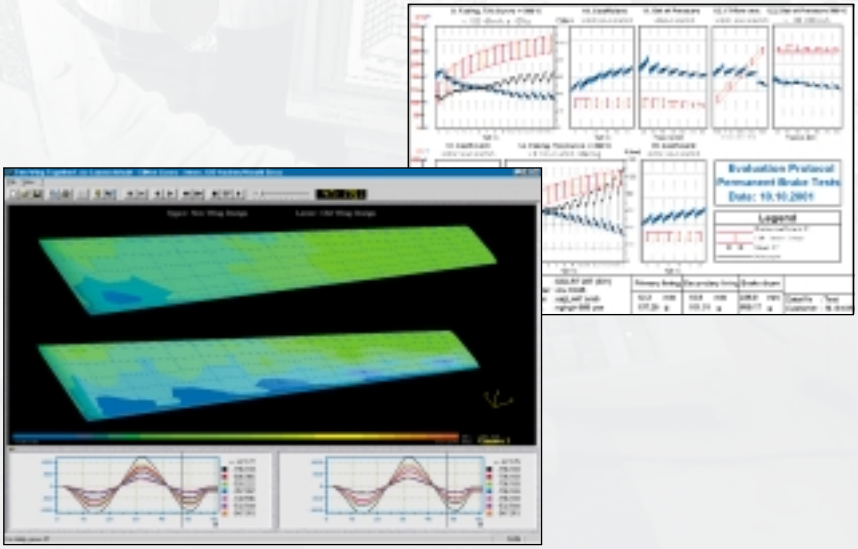
DIAdem™

Script DAC Driver



ni.com/diadem

NATIONAL INSTRUMENTS™

DIAdem™

Script DAC Driver Description

November 2003 Edition

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA
Tel: 001 512 683 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Calgary) 403 274 9391,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427,
Hong Kong 2645 3186, India 91805275406, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 27 73 00,
Poland 48 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886,
Spain 91 640 0085, Sweden 08 587 895 00, Switzerland 056 200 51 51,
Taiwan 02 2377 1200, United Kingdom 01635 523545

For further support information, see the *Technical Support* appendix. To comment on the documentation, send e-mail to techpubs@ni.com

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

Patents

For patents covering National Instruments products, refer to the appropriate location: Help»Patents in your software, the patents.txt file on your CD, or ni.com/patents.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED “SYSTEM FAILURES”). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

1 Introduction

This document is an introduction to the Script DAC Driver. You should have received this document together with the latest version of the following components:

- Script DAC Driver (GfSVBSDR.DLL, GfSVBSDR.G5D)
- UDI-Active-X Control (SFD_OBJECTS.OCX)
- Example files.

You should copy the files of the Script DAC Driver into the directory ADDINFO under the DIAdem installation directory.

Please copy the UDI-Active-X Control into the DIAdem-installation directory and the example files into the DAC directory and the "User“ directory of DIAdem.

This introduction, which is complete in itself, will simplify your getting started with the DIAdem-Script DAC Driver.

If you have already used earlier versions of this document, please pay particular attention to Chapter 8 "List of changes“.

2 General Notes

The Script DAC Driver is a component within the device DIAdem-DAC. It provides the facility of integrating external measurement devices, which are connected to the computer via a communications interface, in DIAdem quickly and without too much effort

The entire Script DAC Driver consists of three sub-components:

- The driver itself, which is addressed in the form of a block within the device DAC. This sub-component is the same for all the devices and is integrated in DIAdem as a GPI-DLL. The DLL is a component of the DIAdem installation.
- A script file that is defined by the user and describes the actual logic of the device linkage. This file is generated for every device type, in order to take into account the special features of the device like the protocols used.
- An Active-X Control, over which the interfaces of the computer (serial, GPIB and TCP/IP) can be addressed. In the text that follows, this control is also called the UDI Control. UDI stands for "Universal Device Interface“, a software interface with which the various communication interfaces of the computer can be addressed in the same way. This sub-component is the same for all devices. This Active-X Control is a component of the DIAdem installation.

By using the Microsoft Visual Basic Script, or VBS in short, as the script language, simple device links can be created quickly and easily and complex protocols can be implemented.

2.1 Installation and Preconditions

Installation of the GPI-DLL

The Script DAC Driver is registered as a DIAdem GPI-driver.

To register, please select, in the dialog for registering a GPI driver, the driver with the name "GfSVBSDR.DLL“ from the directory DIAdem\Addinfo. More details on the installation of a GPI driver can be found in the DIAdem online Help.

After the driver has been registered and loaded in DIAdem, the icons for the input and output blocks can be placed on the module and function group toolbars of DIAdem-DAC. The existing block diagram must first be deleted for this purpose. Thereafter, from the menu Settings » Single-value processing » Configure driver » New entry, the new driver with the name VBScript is selected. Subsequently, in the module and function group toolbar, the blocks "ScriptIn" and "ScriptOut" are available for selection for the block diagrams.

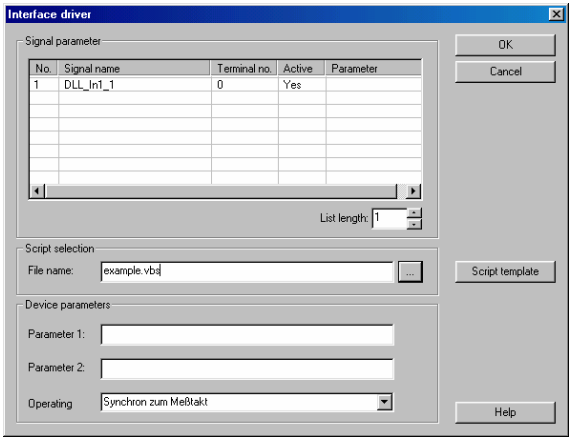
VBScript Runtime Environment

For operating the DIAdem-Script DAC Driver, it is necessary that the runtime environment of VBScript should be installed on your computer. This component is automatically installed at the time of installation of Windows 2000. For Windows 95, Windows 98 and Windows NT, it is installed with the Internet Explorer (from Version 4 onwards).

If the component is not installed, you can download VBScript from the Website "http://msdn.microsoft.com/scripting/“ free of cost.

2.2 Operating with the Script DAC Driver

After an input or output block has been inserted in a block diagram, a double-click on it causes the following dialog for parameterizing the driver to be displayed:



The dialog for the Script DAC Driver contains a number of parameters, which can be divided into two groups:

On the one side, there are parameters that are absolutely necessary for operating the driver. This includes the parameters **File name, Operating mode** and **Signal name**.

The other parameters are optional. These parameters are freely available to the developer of the concerned script. The developer of the script defines their significance and he should document it in detail. Furthermore, it is his task to carry out, in the script, a check of the parameters required and if necessary output clear, informative error messages from the script.

The significance of the individual parameters of the dialog is explained below.

Signal parameter: In the upper part of the dialog is the signal list, in which the individual signals with their parameters are listed. The individual columns of this list contain the following information:

No: A running serial number of the signal is displayed here. This is only used for the information of the user and cannot be changed.

Signal name: The names of the individual signals, which are also used in the block diagram, are displayed in this column. These signal names have to be unique within an Script DAC Driver block and should not exceed the length of 16 characters.

Port-No: This is a freely available parameter that serves to identify a measurement point in the script. This parameter is generally used to specify an input or output of a connected device. Integer values are permissible as the value for this parameter.

Active In this column, it is possible to purposefully enable or disable individual channels for the upcoming measurement. Channels that are disabled serve as placemarkers within the dialog. They are not considered in the measurement and at the time of measurement, no kind of information on these channels is passed to the relevant script.

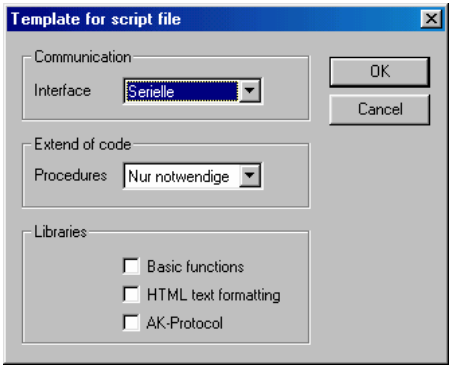
Parameters Here, any parameter, at will, can be passed to the script for every channel. The importance of these parameters is defined in the relevant script and can be taken from the documentation for the script.

List length: The number of signals can be defined using the field List Length. The number can be between 1 and 255.

Script selection: In the middle of the dialog is an input field in which the name of the script to be used has to be specified. By clicking the button " ", you will see a dialog for selecting the desired script.

Device parameters: The device parameters are specified in the lower part of the dialog. These are two parameters that are passed to the script and one parameter for setting the operating mode of the driver. Parameters 1 and 2 are optional parameters whose significance is defined by the developer of the relevant script. The parameter "Operating mode" is used to define in which mode the driver is to be operated. More details on the operating modes of the driver can be obtained from the page Operating Modes of the Script DAC Driver.

Script Template: The button "Script Template" generates a template for a script file and copies that template into the clipboard. After clicking this button, another dialog for parameterizing the contents of the new script file is displayed:



The following terms can be selected in the selection list “Interface”:

Term	Generated script contents
none	No templates are generated for access to a port.
Serial	An example for the parameterization and opening of a serial port is generated
GFS-GPIB	An example for the parameterization and opening of a GPIB device with NEC µPD 7210 processor is generated
NI-GPIB	An example for the parameterization and opening of a GPIB device from National Instruments is generated
TCP/IP	An example for the parameterization and opening of a TCP/IP port is generated

The term "Only required" or "All" can be generated in the selection list "Procedures". If "Only required" is selected, only those procedures are generated, which are absolutely necessary. Otherwise, all the procedures are generated, which have been invoked by DIAdem.

Under "Libraries“, you can select the basic library as well as the library for the AK-protocol.

The generated template can be picked up from the clipboard with any text editor using "Paste". The prepared procedures, which DIAdem calls before, during and immediately after the measurement, can be equipped with the necessary functionality by the user. The functions that the Script DAC Driver does not require can be deleted in the script.

2.3 Script Procedures

DIAdem calls the procedures present in the script file during the various phases of measurement in succession. The following table lists all the functions called by DIAdem:

Procedures of an Input Block

Phase	Procedure	Use	Optional
INIT	SFD_Init	Initialization of the connected device	Yes
	SFD_InitInChannel	Initialization of an input channel	Yes

START	SFD_Start	Start of the measurement or changing a trigger sequence	Yes
MEASUREMENT	SFD_GetScan	Reading a complete scan from the connected device	Yes
	SFD_ReadChannel	Passing of a single measurement value from the script to the Script DAC Driver	No
STOP	SFD_FinalStop	End of the measurement	Yes
DEINIT	SFD_DeInit	De-initialization of the connected device	Yes
	SFD_DeInitInChannel	De-initialization of an input channel	Yes

The script of an input block should therefore at least have the procedure "SFD_ReadChannel“, in which values are returned from the script to DIAdem.

Procedures of an Output Block

Phase	Procedure	Use	Optional
INIT	SFD_Init	Initialization of the connected device	Yes
	SFD_InitOutChannel	Initialization of an output channel	Yes
START	SFD_Start	Start of the measurement or changing a trigger sequence	Yes
MEASUREMENT	SFD_SendScan	Transferring a complete scan to the connected device	No

Phase	Procedure	Use	Optional
	SFD_WriteChannel	Passing of a single measurement value from the Script DAC Driver to the script	No
STOP	SFD_FinalStop	End of the measurement	Yes
DEINIT	SFD_DeInit	De-initialization of the connected device	Yes
	SFD_DeInitOutChannel	De-initialization of an output channel	Yes

The script of an output block must thus contain at least the procedures "SFD_SendScan“ and "SFD_WriteChannel“.

Parameters with the same name always have the same meaning for the function. Thus, e.g. the parameter ErrorP is used to pass an error message from the script to the Script DAC Driver. This applies for all procedures in which the variable ErrorP is used.

If a text is assigned to this parameter within a function, the measurement that is running is ended and the Script DAC Driver outputs an error message in which the value of the parameter ErrorP is output.

3 Procedures

3.1 Acquisition and Output of Data

The procedure SFD_GetScan

The function SFD_GetScan() is called once per measurement cycle. Its task consists of prompting for the measurement data of a complete scan (the data of all the channels that are involved in the measurement) from the connected device and saving it in globally defined variables of the script. With this function, it is possible to receive data from a measurement device that always provides the data for a complete scan.

Syntax : SFD_GetScan(ErrorP)

Parameter:

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note Optional function for the data acquisition

The procedure SFD_ReadChannel

The procedure SFD_ReadChannel() is read to pass the measurement data of an individual channel from the script to DIAdem. This procedure is called once with every measurement cycle for every active measurement channel of the script.

In the case of this function, it must be remembered that the value ParamP, which is returned by the function to the Script DAC Driver, must always be different from VT_Empty. If the value VT_Empty (e.g. a non-initialized variable) is returned here, the Script DAC Driver assumes that this function has not yet been ended. Therefore, SFD_ReadChannel is not called any more, and the driver cannot return any more values. The running DIAdem measurement cannot then be ended properly or aborted.

Syntax :
SFD_ReadChannel(ChannelNumberP, ParamP, DataP, ErrorP)

Parameters:

ChannelNumberP The port number of the signal from the dialog of the Script DAC Driver is passed to the function in this parameter (if it has not been changed by other functions of the scripts). This parameter serves to identify the desired measurement channel.

ParamP The signal parameter from the user interface of the Script DAC Driver is passed to the function in this parameter (provided it has not been changed by other functions of the script).

DataP The measurement data is passed from the script to the Script DAC Driver via the parameter DataP. Here, it is absolutely essential that for every call to this procedure, a value is assigned to this parameter, since this procedure is not otherwise considered to have been completed and the driver is blocked.

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: The implementation of this function in the script is absolutely necessary for data acquisition with the Script DAC Driver.

The procedure SFD_WriteChannel

The function SFD_WriteChannel() is used to pass data from the Script DAC Driver to the script for outputting. This data can be passed in this function either directly to the connected device, or saved in the corresponding variables of the script, so that they can be transferred as a complete scan to the device using the function SFD_SendScan(). This function is called once with every measurement cycle for every active output channel.

Syntax :

SFD_WriteChannel(ChannelNumberP, ParamP, DataP, DoneP, ErrorP)

Parameters:

ChannelNumberP The port number of the signal from the dialog of the Script DAC Driver is passed to the function in this parameter (if it has not been changed by other functions of the scripts). This parameter serves to identify the desired output channel.

ParamP The signal parameter from the user interface of the Script DAC Driver is passed to the function in this parameter (provided it has not been changed by other functions of the script).

DataP The measurement data is passed from Script DAC Driver to the script via the parameter DataP. This data can be transferred directly to the device or saved for transmission in the function SFD_SendScan in the corresponding variables of the script.

DoneP The parameter DoneP is used to signal to the Script DAC Driver that the processing of the function is complete. For this reason, it is absolutely necessary that at the end of a function, any random value should be assigned to this parameter. If no value is assigned to this parameter, the procedure is not called again and the driver is blocked.

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: The implementation of this function is imperative for scripts that have to output data.

The procedure SFD_SendScan

By using the function SFD_GetScan(), the data of a complete scan can be transmitted to a device in one go. This function is called once per measurement cycle.

Syntax:

SFD_SendScan(DoneP, ErrorP)

Parameter:

DoneP The parameter DoneP is used to signal to the Script DAC Driver that the processing of the function is complete. For this reason, it is absolutely necessary that at the end of the function, any random value should be assigned to this parameter. If no value is assigned to this parameter, this function is not called again by the Script DAC Driver and the driver hangs.

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: This function is optional for outputting data.

3.2 Initializing/de-initializing the devices and signals

The procedure SFD_Init

The function SFD_Init() can be used for the basic initialization and connection setup to the device. The function is called once during the measurement preparation.

Syntax:

SFD_Init(DeviceParam1V, DeviceParam2V, ErrorP)

Parameters:

DeviceParam1V,

DeviceParam2V Both these parameters correspond to the device parameters "Parameter 1" and "Parameter 2" from the dialog for the Script DAC Driver. Information on the interface to be used, for example, or on the settings of the connected device can be passed to the script in these parameters.

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: Optional function for basic initialization.

The procedure SFD_DeInit

The function SFD_DeInit() can be used for de-initializing the connected devices and for setting up the connection. The function is called once during the assessment of the measurement.

Syntax: SFD_DeInit(ErrorP)

Parameters:
ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: Optional function for de-initialization

The procedure SFD_InitInChannel

The function SFD_InitInChannel() serves to initialize the required measurement channels. It is called once for every active measurement channel during the measurement preparation. The individual measurement channels can be initialized (e.g. setting the measurement range) and if required, scalings for the Script DAC Driver defined, in this function.

Syntax:

SFD_InitInChannel(ChannelNumberP, ParamP, ErrorP [, ScalingFactorP, ScalingOffsetP] [, ChannelCountV])

Parameters:

ChannelNumberP Here, the port number of the input from the dialog for the DIAdem-Script DAC Driver is passed to the script. This parameter can be used in the script as a pointer to the physical input of the connected device.

ParamP The signal parameter ("Parameters in the dialog for the interface parameter") can be passed to the script in this parameter. This parameter can be used in the script e.g. to set the measurement range of an input.

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

ScalingFactorP With this parameter, a scaling factor can be returned from the script to the Script DAC Driver; using this factor, the measurement values of the channel are calculated automatically at the time of the measurement. This parameter is optional, but can be used only in conjunction with the parameter ScalingOffsetP.

ScalingOffsetP With this parameter, a scaling offset can be returned from the script to the Script DAC Driver; using this factor, the measurement values of the channel are calculated automatically at the time of the measurement. This parameter is optional, but can be used only in conjunction with the parameter ScalingFactorP.

ChannelCountV This optional parameter is used to inform this function how many active channels are present and have to be processed by the script. This information can be used e.g. to create sufficiently large fields for placing measurement data.

Note: Optional function for initializing the inputs

The procedure SFD_DeInitInChannel

The function SFD_DeInitInChannel() serves to de-initialize the inputs used. This function is called once for every active measurement channel during the assessment of the measurement.

Syntax:

SFD_DeInitInChannel(ChannelNumberV, ErrorP)

Parameters:

ChannelNumberV Here, the port number of the input is passed to the script from the dialog of the DIAdem-Script DAC Driver (provided it has not been changed by other functions of the script). This parameter can be used in the script as a pointer to the physical input of the connected device.

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: Optional function for de-initializing the inputs

The procedure SFD_InitOutChannel

The function SFD_InitOutChannel() serves to initialize the required outputs. It is called once for every active output during the measurement preparation. The individual outputs can be initialized (e.g. setting the output range) and if required, scalings for the Script DAC Driver defined, in this function.

Syntax:

SFD_InitOutChannel(ChannelNumberP, ParamP, ErrorP [, ScalingFactorP, ScalingOffsetP] [, ChannelCountV])

Parameter:

ChannelNumberP Here, the port number of the output from the dialog for the DIAdem-Script DAC Driver is passed to the script. This parameter can be used in the script as a pointer to the physical output of the connected device.

ParamP The signal parameter ("Parameters in the dialog to the interface parameter") can be passed to the script in this parameter. This parameter can be used in the script e.g. to set the output range of an output.

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter,

the measurement is immediately ended with an error message that contains this error text.

ScalingFactorP With this parameter, a scaling factor can be returned from the script to the Script DAC Driver; using this factor, the values of the channel to be output are calculated automatically at the time of the measurement. This parameter is optional, but can be used only in conjunction with the parameter ScalingOffsetP.

ScalingOffsetP With this parameter, a scaling offset can be returned from the script to the Script DAC Driver; using this factor, the values of the channel to be output are calculated automatically at the time of the measurement. This parameter is optional, but can be used only in conjunction with the parameter ScalingFactorP.

ChannelCountV This optional parameter is used to inform this function how many active channels are present and have to be processed by the script. This information can be used e.g. to create sufficiently large fields for placing output data.

Note: Optional function for initializing the outputs

The procedure SFD_DeInitOutChannel

An active output channel can be de-initialized again with this function. It is called once for every active output during the measurement assessment.

Syntax:

SFD_DeInitOutChannel(ChannelNumberV, ErrorP)

Parameter:

ChannelNumberV Here, the port number of the output is passed to the script from the dialog of the DIAdem-Script DAC Driver (provided it has not been changed by other functions of the script). This parameter can be used in the script as a pointer to the physical output of the connected device.

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: Optional function for de-initializing the outputs

The procedure SFD_FinalInit

The function SFD_FinalInit() can be used for the final initialization of the connected device and the variables of the scripts. The function is called once at the time of ending the measurement preparation.

Syntax:

SFD_FinalInit(ErrorP)

Parameter:

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: Optional function for final initialization of the script and the connected device.

3.3 Procedures that are called during the start or end of the measurement

The Procedure SFD_Start

This function is called every time the triggering sequence is changed. It makes it possible to start the measurement afresh on the connected device every time the triggering sequence is changed.

Syntax:

SFD_Start(SamplingRateV, ErrorP)

Parameters:

SamplingrateV The polling rate that has been set is passed to the script in this parameter. It can be used to start a measurement with the desired polling rate on the connected device.

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: Optional function for starting a measurement on the connected device.

The procedure SFD_FinalStop

The function SFD_FinalStop() is called once when the relevant clock system is ended. This function can be used to end the communication to a device or a running measurement on a device, if not required any more for the subsequent measurement.

Syntax:

SFD_FinalStop(ErrorP)

Parameters:

ErrorP This parameter serves to return an error message to the Script DAC Driver of DIAdem. If text is assigned to this parameter, the measurement is immediately ended with an error message that contains this error text.

Note: Optional function that is called at the end of a measurement.

4 The UDI-Control

The UDI Control (UDI stands for "Universal Device Interface“) forms a neutral software interface to the communications interfaces of the computer (RS-232, GPIB etc.). Here, the functions for reading and writing the data as well as the

parameters of the individual functions are independent of the communications interface used, so that on changing the communications interface, no changes are necessary to the calls of the communications functions.

The UDI functions that can be used from the script are described below.

The parameters that are given in [ ] are optional and can be omitted if required. To be able to use the UDI functions in the driver script, an object of type UDI Control should be created at the start of the script file. This is done with the following lines:

```
Dim oUDI
Set oUDI = CreateObject("DIAdem.SFD.UDI")
```

Then, the functions of UDI can be accessed by means of the object "oUDI".

Example:

```
SBuffer = oUDI.Read( Count, Mode )
Call oUDI.Open( "COM", "COM1")
```

Note: In the examples, the object of type "UDI-Control“ is mostly designated as "oUDI“. This designation has been selected at random and can be customized according to the task. If, e.g. the device is addressed via the serial interface, the object could be meaningfully called "oSerialPort".

4.1 Functions for setting up and shutting down the connection

The Open function

The "Open" function opens a communications channel to an interface and takes care of its initialization with the current interface parameters.

Syntax: Open(DriverName, DeviceName)

Parameter:

DriverName Name of the driver for the communications interface to be used. (see Chapter 3.5)

DeviceName Name of the communications channel to be used. (see Chapter 3.5)

Example:

```
Call oUDI.Open( "COM", "COM2")
```

The Close function

An interface that has been previously opened with "Open" can be closed with the function "Close".

Syntax Close()

Parameter None

Example:

```
Call oUDI.Close()
```

4.2 Functions for data transfer

The Write function

The function "Write()" can be used to send data over a communications interface to a connected device.

Syntax Write(Data[, Count[, Mode]])

Parameter:

Data The contents of this variable are output to the UDI device. This string variable can also contain binary elements.

Count Number of bytes that have to be output. This parameter is optional. If the parameter is not specified, the entire contents of the variable Data are output.

Mode Mode in which data should be transferred (optional) he following values are possible: : The data is transmitted without a delimiter being appended. : A delimiter is appended to the data before transmittal. f the parameter is not specified, the default value of 1 is used.

Example:

```
Call oUDI.Write( "Hello", 5, 1 )
```

The Read function

The Read() function is used to read data from the communications interface.

Syntax Read([Count[, Mode]])

Parameter:

Count Number of bytes that have to be read (optional). If this parameter is not specified, all the bytes present in the buffer of the UDI device are read.

Mode Mode in which data should be transferred (optional) he following values are possible: : Read till the desired number of characters has been reached (binary transmission) : Read till the defined delimiter of the desired number of characters has been reached. f the parameter is not specified, the default value of 1 is used.

Return value The bytes read by UDI in the form of a string. This string can also contain binary elements, which are then processed further with the method "Parse".

Example:

```
Recieve = oUDI.Read( 5, 7 )
```

4.3 Changing the Interface-specific Parameters

In the case of these procedures, it should be noted that the call to the procedure changes parameters whose significance depends on the communications interface used. In particular, what applies here is that many parameters only exist for special communications interfaces (the parameter "BAUDRATE" makes sense in conjunction with a serial interface, but does not make any sense in conjunction with GPIB. A detailed list of the parameters that can be addressed for an interface type can be found in chapter 3.5

The ParamSet function

Setting an interface-specific parameter (e.g. Baudrate) for the UDI device. Here, it should be noted that both the names of the parameters as well as their contents are basically expected in the form of strings by UDI and therefore have to be included in "". Furthermore, attention should be paid to upper and lower case.

Since the individual parameters depend on the communications interface used, the possible parameters and their values are described in the documentation for the relevant communications interface (see chapter 3.5).

Syntax: ParamSet(Parameter, Value)

Parameter Name of the parameter that is to be modified.

Value New parameter value

Example:

```
Call oUDI.ParamSet( "BAUDRATE", "9600" )
```

The ParamGet function

An interface-specific parameter can be queried from the UDI device using this function.

Syntax ParamGet(Parameter)

Parameter Name of the parameter whose value is to be determined.

Return value String that contains the value of the queried parameter.

Example:

```
Baudrate = oUDI.ParamGet( "BAUDRATE" )
```

4.4 Help functions

The ErrorTextGet function

The function ErrorTextGet() can be used to call an error description for errors that have occurred during a call to a UDI function from the corresponding UDI device.

Syntax: ErrorTextGet()

Parameter: None

Return value String that contans the text of the error message.

The Parse function

This function is used to evaluate binary and formatted data that has been previously read with the Read command.

Syntax: Parse(Data, FormatDescriptor)

Parameter:

Data Variable with data that has previously been read with "Read".

FormatDescriptor Format string for describing the structure of the data:

The format string consists of a format description for one or more variables, according to which the data in the parameter "Data" is to be interpreted. The syntax used for each variable is as follows:

```
% [ n] (U,L,D,S) [ <"Name">]
```

The "%" character defines the start of a new format description and mustbe specified. "n" stands for the number of bytes that should be interpreted for the variable. Only one of the letters U, L, D, S should be specified. "U" stands for Unsigned Integer "L" stands for Signed Integer "D" stands for a real number "S" stands for a text (string) <Name> stands for the name of the variable. If the name is specified, after calling the command "Parse" with the commands "NamedValueGet“ and "NamedValueSet“, it is possible to access the value of the variables.

Return value Value(s) of the variables

If the description of one variable is specified in vFormat, a single value is returned. If several descriptions are specified (see example 2), a field of values is returned

Note If the procedure "Parse" is used for evaluating formatted numbers, it is assumed that the decimal separator is a period ("."). This is independent of what specifications have been made in the country settings Should you use the VBScript function for converting text into numbers, they react to the specifications made under the country settings.

Example 1:

```
Dim vData, vValue
vData = oUDI.Read()       ' Reading all the existing data
' from the UDI-Object
vValue = Parse(vData,"%2L")       ' Interpretation of 2 bytes from
"vData"
' and returning the value that has
' been read to the variable vData
```

Example 2:

```
Dim vData, vArray
vData = oUDI.Read()       ' Reading all the existing data from the UDI-
Object
vArray = oUDI.Parse(vData,"%2L%4L")       'Reading a total of 6 bytes
from vData and 'returning 2 values in the field vArray
For L = 1 To UBound(vArray)       'Outputting the two values in a
MessageBox
MsgBox(CStr(vArray(L))
Next L

Example 3:
Dim vData, vArray, vScaledValue
vData = oUDI.Read()       ' Reading all the existing data from the
        ' UDI object
vArray = oUDI.Parse(vData,"%2L<Trigger>%4L<Temperature>")
        ' Reading a total of 6 bytes from vData
        ' and returning 2 values to the field
        ' vArray
vScaledValue = oUDI.NamedValueGet("Temperature ")*0.01
MsgBox(vScaledValue)      ' Direct access to the value mentioned
        ' "Temperature" and output of the scaled
        ' value in a message box
```

The Append function

Appending the contents of a variable (value) to the output buffer, taking into account the format Format.

Syntax Append(Buffer, Value,Format)

Parameters:

Buffer Variable to which the contents of the variable Value are appended.

Value Variable whose contents are to be appended to the variable Buffer.

Format Description of the format in which the variable should be appended.

Note: Additional description of the format: see Parse

Example:

see the description of the command "Parse()"

The Sleep function

This function is used to interrupt the execution of the script for a specified number of milliseconds.

Syntax: Sleep(MilliSeconds)

Parameter:

MilliSeconds Number of milliseconds

Example:

```
Sleep(100)       ' Interrupts the execution of the script
        ' for 1/10ths of a second
```

The NamedValueSet function

Setting the value (Value) for the named variable with the designator "Name".

Syntax: NamedValueSet(Name, Value)

Parameter:

Name Name of the variable

Value Value of the variable

Example:

See: NamedValueGet()

The NamedValueGet function

Reading the value (Value) for the named variable with the name "Name".

Syntax: NamedValueGet(Name)

Parameter:

Name Name of the variable

Return value Value of the variable

Example:

```
Dim vScaledValue
vScaledValue = oUDI.NamedValueGet("Temperature")
        ' Reading the value of the variable with
        ' the name "Temperature". The value is
        ' assigned to the local variable
         '
vScaledValue = (vScaledValue-32.145) * 0.01
        ' Offset correction and scaling
        ' of the variable
oUDI.NamedValueSet("Temperature",vScaledValue)
        ' Rewriting the value
```

The attribute ByteSwap

The byte sequence (the sequence in which the binary data is transferred) can be set using the attribute "ByteSwap" of the UDI object. In case of ByteSwap = False (default), the binary data is processed with the command "Parse" in the sequence High Byte - Low Byte, whereas in case of ByteSwap = True, they are processed in the sequence Low Byte - High Byte. To set the byte sequence, either the value 0 (false) or 1 (true) is assigned to the attribute ByteSwap.

Example:

```
oUDI.ByteSwap = 1
```

The attribute Version

The current version of the UDI control can be read with this attribute. The version is returned as text in the form "x.yy". "x" stands for the main version number and "yy" for the two-digit sub-version.

The attribute IsOpen

With this attribute, you can query whether an interface has been opened for the current UDI-Control. If an interface is open, "true" is returned, else "false".

4.5 Supported Interfaces

The UDI-Control supports the following interfaces:

- Serial interface
- GPIB with the two following sub-types:

- GPIB cards from National Instruments. The precondition is that the NI-488-2 driver from National Instruments should be installed on the computer.
- GPIB cards with NEC µPD 7210, which are register-compatible with PCII or PCIIa from National Instruments.
- TCP/IP

Each of these interfaces can be configured with a set of parameters. The parameters are passed to the UDI Control with the command "ParamSet". The following sections describe the interfaces supported by the UDI-Control, or their parameters.

Serial Interface (RS-232)

Driver for operating the serial interfaces

Driver name: COM

Device name: COM1 ... COM9

Supported parameters:

BAUDRATE Baudrate at which the interface has to be operated. Examples for the possible values are 1200, 2400, 4800, 9600, 19200. In addition, other baudrates are also possible as long as the rate is an integer.

PARITY Parity ossible values are NONE (no parity),ODD (odd parity) and EVEN (even parity)

STOPBITS Number of stop bits with which the interface is to be operated (1, 2).

DATABITS Number of data bits with which each character is to be transmitted (7, 8).

TIMEOUT Time in ms for which a transmission function waits for data before it aborts with a timeout error.

DELIMITER Character string that marks the end of a line. This string can be several characters long. The delimiter is passed as a normal character string. Special characters can be input here in the form \HH. "HH" stands for a formatted description of a hexadecimal value. Thus, "\0D" stands for "Carriage-Return" (decimal 13). In this context, it is absolutely essential to keep in mind the text constants that have already been pre-defined in VBScript such as e.g. vbCR, vbLF or vbCRLF.

Example:
The baud rate is set to 9600 baud with the following command.

```
Call oUDI.Open( "COM", ""COM1" )
Call oUDI.ParamSet( "BAUDRATE", "9600" )
```

GPIB (IEEE-488)

GfS GPIB

Drivers for operating GPIB cards with NEC µPD 7210, which are register-compatible with PCII or PCIIa from National Instruments.

Driver name: GfS GPIB

Device name: Dev 1 ... Dev30

Supported parameters:

BASE ADDRESS The base address of the GPIB card that is inbuilt in the computer is specified with this parameter. This base address should not be confused with the device address of the connected devices. The latter is set on the devices and not on the GPIB card in the computer. (See the settings of the interface card)

BOARDTYPE "PCII" (The card is register-compatible with National Instruments PCII.)

"PCIIa" (The card is register-compatible with National Instruments PCIIa.)

TIMEOUT Time in ms for which a transmission function waits for data before it aborts with a timeout error.

DELIMITER Character string that marks the end of a line. This string can be several characters long. The delimiter is passed as a normal character string. Special characters can be input here in the form \HH.

Example:

```
Call oUDI.Open( "GfS GPIB", "DEV 4" )
Call oUDI.ParamSet( "BOARDTYPE", "PCII" )
```

NI GPIB

Driver for operating the GPIB cards from National Instruments. This driver requires that the NI-488-2 driver from National Instruments should be installed on the computer.

Driver name: NI GPIB

Device name: Dev 1 ... Dev30

Supported parameters:

TIMEOUT Time in ms for which a transmission function waits for data before it aborts with a timeout error.

DELIMITER Character string that marks the end of a line. This string can be several characters long. The delimiter is passed as a normal character string. Special characters can be input here in the form \HH.

Example:

```
Call oUDI.Open( "NI GPIB", "DEV14" )
Call oUDI.ParamSet( "TIMEOUT", "100" )
```

TCP/IP

Driver for communicating with devices via the TCP/IP protocol. This driver requires that TCP/IP should be installed on the computer as a protocol.

Driver name: TCP/IP

Device name: <IP-Address>:<Port number>

The IP-address can be specified as an address in point notation e.g "100.100.100.12“ or as a symbolic name

Supported parameters:

DELIMITER Character string that marks the end of a line. This string can be several characters long. The delimiter is passed as a normal character string. Special characters can be input here in the form \HH.

Example:

```
Call oUDI.Open("TCP/IP", "DSM3000:23")
Call oUDI.ParamSet( "TIMEOUT", "100" )
```

5 Internal Operation of the Script DAC Driver

The DIAdem-Script DAC Driver works according to the following principle :

On one side is the DIAdem driver with its user interface and on the other side, VBScript as the programming language and interpreter. During the (offline) parameterization of the driver, only the DIAdem driver is active and forms the interface between the user and the driver. During the measurement preparation, VBScript is started in a separate program thread, the script file is passed by the driver to VBScript and checked for syntax errors. If an error is detected in this phase, the further measurement preparation is aborted with an error message. The measurement can be started afresh after correction of the error. The global section of the script is processed after the syntax check. The global section includes all the instructions and variable definitions that take place outside the procedures and functions.

Finally, the DIAdem driver calls the functions for initializing the devices and the inputs and outputs. After processing the measurement preparation, the measurement is started and the corresponding functions in the script are called. The same applies for outputting data and the subsequent assessment of the measurement. If the error variable (ErrorP) is set in the script during the processing of a function the measurement is aborted and an error message with the contents of the error variable is output.

5.1 Validity of variables

All the scripts that are processed by the DIAdem-Script DAC Driver are processed in a separate context. And as a result, this means that the validity of variables is basically always limited to the framework of the relevant script. Variables that are defined globally within a script are valid in the entire script and retain their values even between the calls to the individual functions. Variables that are defined within a function are only valid within that function. They lose their validity on leaving the relevant function.

Furthermore, the effect of processing of the scripts in a separate context is that there is on accidental overwriting of data if different scripts use variables with the same name. If the same script is

used in two different DIAdem blocks, the effect of this isolation is that these scripts do not influence each other. The Script DAC Driver thus makes it possible to operate several devices at different communications interfaces at the same time and for this purpose, to use the same script with different parameterization.

5.2 Access to DIAdem Variables

It is possible to access all DIAdem variables within a script. The following should be noted thereby:

- For every DIAdem-variable that has to be accessed, a variable of the same name should be created in the global section of the script.

  e.g. `Dim Autodrvuse` The names of the variables should be taken from the DIAdem Help.

- Only read access to the variables is possible.

- During the definition of the variables in the global section of the script, read access can only take place within the procedures that are called by DIAdem.

5.3 Operating modes of the Script DAC Driver

The DIAdem-Script DAC Driver differentiates between two operating modes: "synchron zum Messtakt" and "asynchron zum Messtakt". Both the operating modes are different only in the type and manner in which the functions for acquiring and outputting data in the script are called by the Script DAC Driver. In the case of all other functions, the process on calling the corresponding functions is independent of the selected operating mode, since the processing of these functions is not time-critical. The differences in the two operating modes is described below.

Synchronous with the measurement clock

At the start of the measurement cycle, the Script DAC Driver calls the function SFD_GetScan() in the script. During the processing of

this function, the Script DAC Driver waits till the function has been processed. Subsequently, the Script DAC Driver calls the function SFD_ReadChannel() for every active measurement channel and waits till the processing of the function has been completed. After the data has been acquired from all the channels, it is passed from the Script DAC Driver to the DIAdem measurement kernel.

Here, it can be seen that the Script DAC Driver always waits on the end of the processing of the functions in the script and hence, for that time, blocks the DIAdem measurement kernel and other drivers. As long as the Script DAC Driver is the only active driver in the DAC block diagram and the selected polling rate has been so chosen that there is sufficient time for processing the script functions, there are no problems. Things look different when the processing of the individual functions takes a lot of time and at the same time, there are other DIAdem drivers active. Here, it may well happen that the Script DAC Driver blocks the other drivers or the measurement core for an impermissibly long time and this leads to data loss or buffer overflows in the other drivers. These problems can be worked around by choosing the operating mode "asynchronous to the measurement clock".

Asynchronous with the measurement clock

In contrast to what happens in the operating mode "synchronous with the measurement clock", here, the Script DAC Driver does not wait for the individual functions in the script to be processed, but reads the data from an internal buffer in which the data has been saved by the script, and forwards them directly to the DIAdem measurement kernel. Individually, the processing of a measurement cycle looks like this:

First, the Script DAC Driver checks whether all the functions of the script for data acquisition have been processed (In the function SFD_ReadChannel(), a value was assigned to the parameter DataP during all calls.). If that is the case, the new measurement values are copied to an internal buffer in the Script DAC Driver. Then, the Script DAC Driver calls the functions for measurement data acquisition in the script (SFD_GetScan() and for every active channel, SFD_ReadChannel()), but without waiting on their return. Regardless of the result fo the previous checking, the Script DAC Driver then reads the measurement values from the internal buffer and passes them on to the DIAdem measurement kernel.

This procedure ensures that with every call of the measurement routine of the DIAdem-Script DAC Driver, valid measurement data is available, which can also be passed directly to the measurement

kernel. However, in contrast to what happens in operating mode synchron zum Messtakt, here, there is no guarantee that with every measurement cycle, "new" measurement data will be passed to the DIAdem measurement kernel. But instead, in this operating mode, the Script DAC Driver can be run with any desired polling rates - without considering the execution time of the individual script functions, without influencing the operation of other drivers or of the DIAdem measurement kernel.

5.4 Influencing other drivers

By using a separate program thread for processing the scripts, it is possible to reduce the influence of other drivers and the program kernel to a minimum. More details on this can be found on the page Operating Modes of the Script DAC Driver. Since all the scripts are processed in a single program thread, it is very possible here, that these scripts influence each other's execution and if applicable, block each other. Here, when generating the scripts, care should be taken that they are processed with the optimum possible speed.

6 Examples

This chapter shows examples for solutions with the Script DAC Driver. For this example, there are two files each that are referred to: The file with the DIAdem-DAC block diagram and the file with the script. The names of the files are mentioned at the started of a new example.

6.1 Naming conventions

Within the VBScript, procedure names and variable names can be assigned at will. All variables are of type VARIANT. A VARIANT is a special data type that can save different types of information, depending on how the variable is used. (Further information can be found in the Help on VBScript).

To be able to better detect the desired use of the variables or their variable types right at the time of reading the script, the following prefixes are used in the examples.

„sg“ Text variable

„d“ Double variable (64 bit real number)

„l“ Integer variable (32 bit)

A suffix is used for a further distinction between local (more accurately, procedure-local) variable and a global variable.

„T“ local variable

„M“ global variable

Examples :

sgFilenameM global text variable

dNoValueT local real variable

6.2 Examples without measurement hardware

Example: Simulation block

The files used are Simulation.vbs and Simulation.dac.

In this example, we present a script file that simulates the reading of data. Up to 4 signals are made available. The signals can be requested from the DAC block with the name of the signal type. The name of the signal type is specified in the column "Parameters".

The signals available are:

Signal type name	Function	Range
Sine	Sine function sin(t) "t" is the time since the start of measurement.	-1 to 1
Cosine	Cosine function cos(t). "t" is the time since the start of measurement.	-1 to 1
Random	The "Random" function of VBScript.	-1 to 1
Counter	A sawtooth function that increases from 0 to 50 and then starts again at 0. The value of the function is incremented by a value of 1 for every measurement clock cycle.	-1 to 1

The following table contains a listing of the script (ScriptExample 1.vbs)

General

```
'-----------------------------------------------
' DIAdem-Script-Driver for simulation device
' Implemented  : 13.5.2000 (Ha)
' Tested       : 13.5.2000 (Ha)
'-----------------------------------------------
Option Explicit
'-----------------------------------------------
' Global parameters
'-----------------------------------------------
------------------------------
' dStartTimeM         | Variable to save time of
Skript start
' dNoValueM           | Variable to hold NoValue
' lCounterM           | Global counter, incremented
during each scan
'-----------------------------------------------
------------------------------
Dim      dStartTimeM,dNoValueM,lCounterM,sgErrWSigM
'-----------------------------------------------
' Initialize global variables
'-----------------------------------------------
dStartTimeM = Timer
dNoValueM   = 9.9E+34
lCounterM   = 0
sgErrWSigM  = vbCR +"The selected signal type >%1<
is not supported" + vbCR
sgErrWSigM  = sgErrWSigM + "Please select one of
the following terms :"
sgErrWSigM  = sgErrWSigM + vbCR+vbCR+"Sine,
Cosine,Random, Counter"
```

Initialization

```
'-----------------------------------------------
------------------
' SFD_InitInChannel
' Purpose             : Called for each channel
with user defined values for
'                       channel number and channel
specific string.
'                       Used to check for valid
parameters
'-----------------------------------------------
------------------
' lChannelNumberP     | User defined channel
number
' sgParamP            | User defined string
' sgErrorP            | Text variable to signal
DIAdem that an error has occurred
'-----------------------------------------------
------------------
Sub SFD_InitInChannel( lChannelNumberP, sgParamP,
```

```
sgErrorP )
  Dim bSuccessT
  '-----------------------------------------------
--
  ' Check whether the user set sgParamP to one of
  ' the existing function types. User lowercase
  ' characters for comparison
  '-----------------------------------------------
--
  Select Case LCase(sgParamP)
    Case "sine"   bSuccessT = true
    Case "cosine" bSuccessT = true
    Case "random"  bSuccessT = true
    Case "counter"  bSuccessT = true
    Case Else      bSuccessT = false
  End Select
  '-----------------------------------------------
--
  ' Format error message if necessary
  '-----------------------------------------------
--
  if ( Not bSuccessT ) Then sgErrorP =
Replace(sgErrWSigM,"%1",sgParamP)
End Sub
```

Read data

```
'-----------------------------------------------
-------------------
' SFD_ReadChannel
' Purpose             : Read value for one channel
'                       The channel value is
returned on variable "dDataP"
'-----------------------------------------------
-------------------
' lChannelNumberP     | User defined channel
number
' sgParamP            | User defined string
' dDataP              | Variable to return value
to return.
'                     | The variable type is
"double"
' sgErrorP            | Text-variable to signal
DIAdem that an error has occurred
'-----------------------------------------------
-------------------
Sub SFD_ReadChannel( lChannelNumberP, sgParamP,
dDataP, sgErrorP )
  Dim  dElapseTimeT
  '-----------------------------------------------
--
  ' Calculate time since measurement start
  ' the existing function types. User lowercase
  ' characters for comparison
  '-----------------------------------------------
```

```
--
  dElapseTimeT = Timer - dStartTimeM
  '-----------------------------------------------
--
  ' Calculate value depending on the user defined
  ' parameter
  '-----------------------------------------------
--
  Select Case LCase(sgParamP)
    Case "sine"   dDataP = Sin(dElapseTimeT)
    Case "cosine" dDataP = Cos(dElapseTimeT)
    Case "random"  dDataP = ((CDbl(Rnd)-0.5)*2.)
    Case "counter"
      lCounterM = (lCounterM+1) mod 50
      dDataP    = (CDbl(lCounterM)*.04)-1.
    Case Else      dDataP = dNoValueM
  End Select
End Sub
```

Example for a serial mouse

The files used in this example are Mouse.vbs and Mouse.dac.

The current position and the current state of the button of a mouse is read in this example. The mouse should be connected via the serial port and it should be MS-Mouse compatible. The serial port should be configured with 1200 baud, no parity, 8 databits and 1 stop bit. These parameters are fixed with the mouse, and are therefore directly entered in the script. The port from which the mouse is to be operated can be entered in the block diagram in the block dialog.

Note: The procedure "SFDU_COMInit“ is not listed,

since this procedure can be generated automatically with the function "New Script"

from within the block.

General

```
'-----------------------------------------
' Define global data
' oUDIM         : UDI object to communicate
with serial mouse
' dMouseHM      : Horizontal mouse position.
Forced to a range of 0-1000
' dMouseVM      : Vertical mouse position.
Forced to a range of 0-1000
' dLButtonDownM : State info for left mouse
button. 1=pressed, 0=not pressed
' dRButtonDownM : State info for right mouse
button. 1=pressed, 0=not pressed
-----------------------------------------
Option Explicit
Dim
oUDIM,dMouseHM,dMouseVM,dLButtonDownM,dRButt
onDownM,adBitByte1(24),cDataByteT(3)
Dim  dDeltaXM,dDeltaYM
Set  oUDIM = CreateObject( "DIAdem.SFD.UDI"
)
```

Initialization

```
' SFD_Init
' Purpose             : Initialize UDI
object
'                       Parameters for
serial I/O set to 1200 Baud,No parity,
'                       8 databits, 1
stopbit
'                       Name of parallel
port coming from userdefinition in
'                       block parameters.
Must be in the form "COM1", "COM2"...
'-----------------------------------------
Sub SFD_Init( ByRef sgDeviceparam1, ByRef
sgDeviceparam2, ByRef sgError )
  ' Initialize serial I/O
  Call
SFDU_COMInit(oUDIM,sgDeviceparam1+",1200,N,8
,1")
  Call oUDIM.ParamSet("Timeout",100)
  Call oUDIM.Read(1000,0)
  ' Set timeout to 100 microseconds :
  Call oUDIM.ParamSet("Timeout",1000)
  ' Initialize mouse position and button
state info
  dMouseHM      = 500.
  dMouseVM      = 500.
  dLButtonDownM = 0.
  dRButtonDownM = 0.
End Sub
```

Read data

```
'-----------------------------------------
------------------------
' SFD_GetScan
' Purpose             : Read one scan of data
'                       Mouse data consists
of three bytes. First byte with
'                       button information on
bit's 4 and 5 (counting from 0)
'                       and dx in second byte
dy in third byte
'
' Description of mouse data coming from
serial port :
'
' The data packets are sent at 1200 baud with
1 stop bit and no parity. Each
' packet consists of 3 bytes. It is sent to
the computer every time the mouse
' changes state (ie. the mouse is moved or
the buttons are pressed/released).
'
'                     D7    D6    D5    D4
D3    D2    D1    D0
'                     ---------------------------
--------------------
'       1st byte  |  1     1     LB    RB
Y7    Y6    X7    X6
'       2nd byte  |  1     0     X5    X4
X3    X2    X1    X0
'       3rd byte  |  1     0     Y5    Y4
Y3    Y2    Y1    Y0
'                                   32    16
8     4     2     1
'-----------------------------------------
------------------------
' ErrorP              | Errortext
'-----------------------------------------
------------------------
Sub SFD_GetScan(ErrorP )
  Dim bMouseDataT,dDeltaT,I,K,dValueT,lValueT
  ' Try to get three bytes of data
  cDataByteT(0) = 0
  cDataByteT(1) = 0
  cDataByteT(2) = 0
  bMouseDataT = oUDIM.Read(3,0)
  dDeltaXM      = 0.
  dDeltaYM      = 0.

  if ( 3 = Len(bMouseDataT) ) Then

    dLButtonDownM = 0.
    dRButtonDownM = 0.
    dDeltaXM      = 0.
```

```
    dDeltaYM      = 0.

    cDataByteT(0) =
AscB(Mid(bMouseDataT,1,1))
    cDataByteT(1) =
AscB(Mid(bMouseDataT,2,1)) and 63
    cDataByteT(2) =
AscB(Mid(bMouseDataT,3,1)) and 63

    cDataByteT(1) = cDataByteT(1) +
(cDataByteT(0) and  3)*64
    cDataByteT(2) = cDataByteT(2) +
(cDataByteT(0) and 12)*16


    dDeltaXM = DbleFrom2Compl(cDataByteT(1))
    dDeltaYM =
DbleFrom2Compl(cDataByteT(2))*(-1.)
    dMouseHM = dMouseHM + dDeltaXM*.5
    dMouseVM = dMouseVM + dDeltaYM*.5


    ' Force a maximum range of 0 to 1000 for
x and y
    if (    0. > dMouseHM ) Then dMouseHM =
0.
    if ( 1000. < dMouseHM ) Then dMouseHM =
1000.
    if (    0. > dMouseVM ) Then dMouseVM =
0.
    if ( 1000. < dMouseVM ) Then dMouseVM =
1000.
    ' Check whether a button was pressed
    if ( BitCheck(cDataByteT(0),5) ) Then
dLButtonDownM = 1.
    if ( BitCheck(cDataByteT(0),4) ) Then
dRButtonDownM = 1.

  End If
End Sub

'-----------------------------------------
------------------------
' SFD_ReadChannel
' Purpose             : Called from DIAdem to
get one scan of data. The name
'                       of the channel is
give in sgChannelParam. The
'                       parameter name is
case sensitiv
'-----------------------------------------
------------------------
Sub SFD_ReadChannel(ByRef
```

```
lChannelNumber,sgChannelParam,ByRef
dValue,ErrorP )
  Select Case  sgChannelParam
    Case "Byte1"       dValue =
CDbl(cDataByteT(0))
    Case "Byte2"       dValue =
CDbl(cDataByteT(1))
    Case "Byte3"       dValue =
CDbl(cDataByteT(2))
    Case "DeltaX"      dValue = dDeltaXM
    Case "DeltaY"      dValue = dDeltaYM
    Case "MouseX"      dValue = dMouseHM
    Case "MouseY"      dValue = dMouseVM
    Case "LButtonDown" dValue = dLButtonDownM
    Case "RButtonDown" dValue = dRButtonDownM

    Case Else          dValue = 0.
  End Select
End Sub
```

De-initialization

```
'-----------------------------------------
------------------------
' SFD_DeInit
' Purpose             : "Shut down"
communication. Called once at the end of
'                       a measurement task
'-----------------------------------------
------------------------
Sub SFD_DeInit(ErrorP )
  oUDIM.Close()
  Set oUDIM = Nothing
End Sub
```

Help function

```
'-----------------------------------------
------------------------
' DbleFrom2Compl
' Purpose             : Convert a byte with
number coded as two complement to
'                       a valid double value
'-----------------------------------------
------------------------
Function DbleFrom2Compl(TwoComplByte)
  DbleFrom2Compl = 0.
  Dim   uByteT,lValueT
  uByteT  = TwoComplByte and 255
  uByteT  = CByte(uByteT)
  lValueT = uByteT
  if ( CBool(uByteT and (2^7)) ) Then
    lValueT = CInt(uByteT)
    lValueT = Not lValueT
    lValueT = lValueT and 127
    lValueT = lValueT + 1
    lValueT = lValueT * (-1)
  End If
  DbleFrom2Compl = CDbl(lValueT)
```

```
  Exit Function
End Function

'-----------------------------------------
------------------------
' BitCheck
' Purpose             : Check whether bit
"BitNo" in variable "Value" is set
'                       Return true if bit is
set
'-----------------------------------------
------------------------
Function BitCheck(Value,BitNo)
  BitCheck = CBool(Value and (2^BitNo))
  Exit Function
End Function
```

6.3 Examples with the device simulator

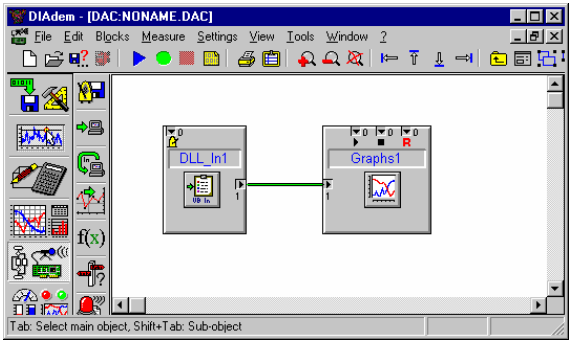
The objective of this document is to introduce the user of the Script DAC Driver to using the DIAdem-Script DAC Driver in a few steps. Here, the possibilities that the DIAdem-Script DAC Driver offers will be illustrated with the help of a number of examples. The examples on the DIAdem-Script DAC Driver are based on the device simulator, whose description can be taken from the corresponding documentation.

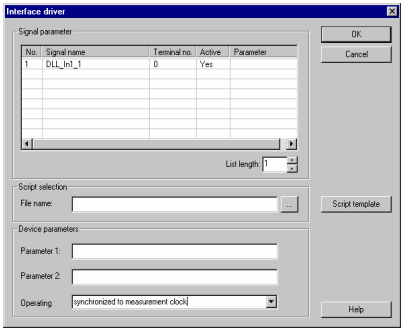
Starting with a simple example, it will be expanded continuously in steps till the entire control of the device simulator is finally being done by the Script DAC Driver.

It is advisable to read through all the documentation on the DIAdem-Script DAC Driver and the device simulator to be able to understand the examples better.

Preparations

The DIAdem-block diagram is deleted to start with. Then, from the module and function group toolbar for the inputs, select a block for the Script DAC Driver. Connect this block with a graph block that you pick up from the module and function group toolbar for the display. Your block diagram should then look something like this:



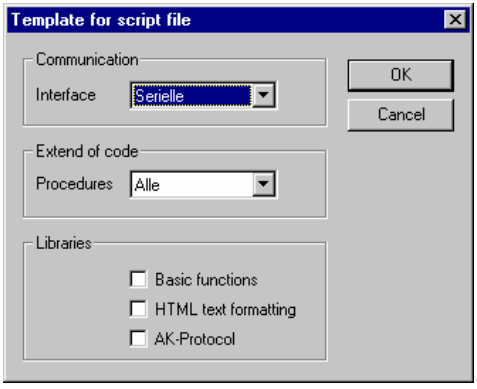


After you have set the default clock for this block diagram to 20 Hz (Menu Options - Single-value Processing - Default Clock), you can save the block diagram under the name "template.dac". This block diagram will later serves as the basis for more examples.

On double-clicking the block DLL_In1, you get the following dialog for setting the parameters for the Script DAC Driver:

Here, first press the button "Script Template".

The following dialog is now displayed:



Please select the following settings:

- Interface = "serial"
- Procedures = "Only essential“

Then click on the button "OK". Thereupon, a template for a VBScript is copied to the clipboard. You can now paste this template in any ASCII editor and edit it. The template contains entry points for all functions that can be called by the Script DAC Driver. Find the function SFD_Init in the file. There, correct the line:

```
Call SFDU_COMInit(oUDIM,"COM1,9600,N,8,1")
```

to

```
Call SFDU_COMInit(oUDIM,"COM2,9600,N,8,1")
```

The serial port COM2 is opened with this command. If you are working with another serial interface this line should be changed accordingly. The settings 9600 baud, no parity, 8 data bits and 1 stop bit are set for communication with the device simulator. At the end of the measurement, the serial port should be closed again. This is done in the function SFD_DeInit with the following line.

```
call oUDIM.Close
```

Please save this file under the name "template.vbs", since it forms the basis of all examples.

After the preparations are complete, the first example for data acquisition with the Script DAC Driver should now be generated.

Simple examples with the device simulator

Example 1, Measurement of channel 1

In this example, measurement data is to be acquired with the Script DAC Driver from Channel 1 of the device simulator. The data is to be transmitted from the device simulator in ASCII format.

To process this example, first start DIAdem and load the block diagram "template.dac" that you have created. Then, load the file "template.vbs" in any ASCII editor and save the file under the name "example1.vbs". Now, switch back to DIAdem and open the dialog for parameterizing the Script DAC Driver with a double-click on the block "DLL_In1". In the field "File name", enter the name "example1.vbs".

In the editor, search for the function SFD_ReadChannel() in the file "example1.vbs. The data of the individual measurement channels is acquired and transferred to DIAdem in this function.

Supplement the definition of the variable "sgRespondT" in the first line.

```
Dim   sgRespondT
```

Then, add the command for querying the channel 1 of the device simulator. The channel 1 is queried with the line

```
Call oUDIM.Write( "MSV?1")
```

Thereupon, the response of the device is read with

```
sgRespondT = oUDIM.Read()
```

```
DataP = CDbl( oUDIM.Parse(sgRespondT,"%AD"))
```

and converted to the variable DataP. This response can now be processed further in the function and the result passed to DIAdem. You can now save the finished script under the name "example1.vbs" and then start the device simulator. Here, activate channel 1. Then, switch to DIAdem and start the measurement there. After you have ended the measurement, you can save the block diagram under the name "example1.dac".

Example 2, Activating the required measurement channels

Building up on example 1, the script should be expanded to such an extent that not only data can be read from channel 1 of the device simulator, but that the data can be queried from any channels of the device simulator. To do so, during the measurement preparation, all the channels should be deactivated and then the necessary channels should be specifically activated again. The parameter "Port no." from the dialog of the Script DAC Driver should be used as the parameter for which channels should be measured. The script "example1.vbs" serves as the basis for this further development. This example is implemented in two steps. First, in the function SFD_InitInChannel(), the necessary channels are activated. To do so, add the function SFD_InitInChannel() in your script.

```
Sub SFD_InitInChannel( ChannelNumberP, ParamP, ErrorP )
End Sub
```

In this function, the activation command is transferred to the device simulator.

```
Call oUDIM.write("ACH" + CStr(ChannelNumberP)+",1" )
```

Since the device simulator responds to the command with a status message, it should subsequently be read.

oUDIM.Read()

This response can then be evaluated further and if applicable, used for outputting an error message. However, in this example, the response is not to be evaluated further.

Finally, delete the following line in the function SFD_ReadChannel

Call oUDIM.Write("MSV?1")

and replace it with the following line:

Call oUDI.Write("MSV?"+CStr(ChannelNumberP))

Then, you can save the script under the name "example2.vbs" and register it in the block for the Script DAC Driver in DIAdem . Now, you can expand the list of measurement channels to be acquired in the block for the Script DAC Driver and then start the measurement. You can view the completed example in the files "example2.dac" and "example2.vbs".

Task:

Enhance the function SFD_Init() with a loop in which all the channels are first deactivated.

Solution:

To deactivate all the measurement channels at the start of a measurement, simply insert a loop in the function SFD_Init(), in which all the channels of the device simulator are deactivated.

```
For lChannelNumberT = 0 to 9
    Call oUDIM.Write("ACH"+CStr(lChannelNumberT)+",0")
    Call oUDIM.Read()
Next
```

Do not forget to insert the definition of the variable lChannelNumberT at the start of the procedure SFD_Init():

Dim lChannelNumberT

The script with the solution for this task is called "solution2.vbs"; the block diagram file is called "solution2.dac".

Influence of the data format on the execution speed

If the last example is run with several channels, you will quickly find that the polling rate of 20 Hz that has been set cannot be reached with the Script DAC Driver. The causes of this will be examined in further detail in the examples below, and we will show how the script for the driver can be optimized. One reason for the bad performance of the Script DAC Driver should firstly be sought in the connected device (here, the device simulator). This device can, if required, process the incoming commands with only a certain speed. This limit is device-dependent and cannot be influenced further by the Script DAC Driver. Another point that greatly influences the performance is the interface used and its performance. This parameter, too, can only be influenced to a limited extent (e.g. by selecting a higher transfer rate). Another parameter that can greatly influence the operating speed is the quantity of data to be transferred. The influence of the quantity of data to be transferred on the performance of the driver can sometimes be influenced by cleverly using the command set of the device. In the last example, 15 bytes are to be transferred between the computer and the device simulator over the serial port per measurement value. (The measurement command "MSV?x<CR><LF>" = 7 characters and the data "x.xxxx<CR><LF> = 8 characters). With the setting of the serial port that is being used (8 data bits, 1 stop bit, 1 start bit), 10 bits are transferred over the serial port per byte. From this, we can conclude that a total of 150 bits have to be transferred for every measurement value. At a transmission rate of 9600 baud, this results in a maximum summation polling rate of 64 Hz. This limit represents the theoretical upper limit for the summation polling rate and cannot be exceeded with the specified transfer parameters. Setting a higher summation polling rate is thus meaningless from the start.

The polling rate that has been currently reached can be determined with the block diagram "samplingrate.dac". Here, the block diagram exploits the fact that in the measurement processes that are run in the foreground, the slowest measurement process determines the maximum polling rate of a DAC task. In the sub-block diagram, the time difference between 10 measurement values is determined and the current polling rate is determined from it and displayed. This block diagram should serve as the basis for the subsequent examples and show what influence the following optimizations of the script have on the achievable polling rate.

Example 3, Using different formats

A. ASCII

To start with, we will examine what the influence of the use of a certain data format can have on the achievable polling rate. To do so, first open the block diagram "samplingrate.dac" and in the block for the Script DAC Driver, input the file "example2.vbs" as the script to be executed. Now, determine the achieved polling rate with this block diagram.

Since, in the examples that follow, the output format of the device simulator will be changed, you should include a command in the function SFD_Init() of the script, which sets the output format of the device simulator to the desired format. To do so, in the script "example2.vbs" in the function SFD_Init, insert the following lines and then save the script under the name "example3a.vbs".

```
Call oUDIM.Write( "COF 0" )
Call oUDIM.Read()
```

With these commands, the output format of the device simulator now gets set to "ASCII".

B. ASCII with Status

Next, we will examine the output format "ASCII with Status". To do so, change the command for setting the output format in the function SFD_Init() correspondingly, and save the script under the name "example3b.vbs". The output format "ASCII with status" first returns the number of the measurement channel followed by a semicolon. This is followed by the actual measurement value, followed by a <CR><LF>. For this reason, the function for data acquisition (SFD_ReadChannel()) should be matched to the new data format. After the response has been read by the device, the measurement value should not be filtered out of the response. This is done with

```
Response = Right( Response, Len(Response) - 2 )
```

After the script "example3b.vbs" has been saved, it can be tested with DIAdem. For this purpose, in the block for the Script DAC Driver, the script "example3b.vbs" is specified as the script to be executed. During the subsequent measurement, it is seen that this script runs somewhat slower than the script "example3a.vbs". This

is not really surprising, since here, 2 more characters have to be transmitted per measurement value.

C. Binary Data

We will now test a binary data format. For this purpose the output format of the device simulator will be set to "binary 2 byte MSB-LSB with status" (COF 5). With this format, the device simulator transmits the following 3 byte-long string per measurement value: 1 byte channel number, 2 bytes measurement value The data is not delimited with ,<CR><LF>. For this reason, the read routine in the function SFD_Readchanngl() is changed as follows:

```
sgRespondT = oUDIM.Read(3,0)
' The response now exists as a <1-byte status> <2-byte measurement
value>
aDataT = oUDIM.Parse(sgRespondT,"%1L%2L")
DataP = aDataT(1)
```

In the variable Count, we first store how many characters are to be read. Then, the data is read with the function oUDI.Read(). The second parameter (mode = 0) specifies that the data should be transmitted in binary form here (without an end-of-line character). After reading, the binary data is available as a 3-character string in the variable Response (Response) for further processing.

This binary data cannot be directly processed by VBScript, so that it must first be prepared with a Help function (oUDI.Parse). To start with, the channel number (1 byte) is deleted from the string. Then, by means of the function oUDI.Parse, the measurement value is converted from the string into a format that is legible for VBScript, assigned to the variable DataP and then passed to DIAdem. This script is then saved under the name "example3c.vbs".

When you execute a measurement with the script, you will observe two things: Firstly, the achievable polling rate is somewhat higher than in the previous examples and secondly, the measurement data spills out of the depiction area of the graph display. The reason for the latter behavior is that in the binary transmission, the data is transmitted unscaled. The range of values of the signal (-1 to +1) is mapped to the range -32767 to 32767. After re-scaling the display, the measurement data is once again depicted correctly qualitatively.

This example has shown that the selection of the transmission format of the data can have an effect on the performance of the Script DAC Driver. In this case, however, this effect is relatively minor. However, a comparison of the theoretically possible polling rate with the actually achievable summation polling rate 64

Hz/about 12 Hz) shows that in this case, the data transmission is probably not the factor effecting the slowing down.

But before we analyze other possibilities for accelerating the script, we should supplement the last example with two points:

- Transmission of scaled values with binary data transmission
- Parameterizable data format for the data output of the device simulator

Parameterization and Scaling

Example 4, Scaling of binary data

There are a few different possibilities for scaling the binary data that is read to the measurement quantities. One possibility is in scaling the data that has been read in the DIAdem block diagram by means of a suitably configured scaling block. In this case, however, at the time of changing the measurement range on the device, a corresponding adjustment of the parameterization of the scaling block would be required. Another alternative is to multiply the measurement data in the script by the corresponding scaling factor before it is passed on to DIAdem . Even with this method, it is necessary to make the corresponding changes in the script with every change in the measurement range on the device.

A more elegant method of scaling is to query the current measurement range of the corresponding channel for the device in the measurement preparation (SFD_Init() or SFD_InitChannel()) and from it, determine the suitable scaling parameters. These parameters can be saved in a field of suitable size and used for calculation in the function SFD_ReadChannel() with the binary data that has been read. Another possibility is to use the optional parameters FaktorP and OffsetP of the function SFD_InitChannel() for saving the scaling parameters of a channel.

The latter possibility will now be implemented in this example. To do so, load the example "example3c.vbs" in your editor and there, search for the function SFD_InitInChannel(). Now supplement the parameter list of this function with the two parameters FaktorP and OffsetP. With the command AMP?, you can now query the amplitude that has been set on the device simulator for a channel. You can determine the required scaling factor by dividing this value by 32767.

First, insert a new first line within SFD_InitInChannel():

```
Dim     dAmplitudeT

Then, at the end of the procedure, add the following lines:
Call oUDIM.Write("AMP?"+CStr(ChannelNumberP))
dAmplitudeT = CDbl(oUDIM.Read())
FaktorP = dAmplitudeT/32767.
OffsetP = 0.
```

Please save this example under the name "example4.vbs". In this example, it should be noted that here, the measurement range (amplitude) of the signal must remain constant across the entire measurement duration, since the scaling factor is determined only once in the measurement preparation. If it has to be possible for the measurement range to be changed during the running of a measurement (e.g. auto-range), it would have to be continuously determined afresh during the measurement and used for calculations with the measurement data. In this case, however, it is not meaningful to use such an unscaled data format for the data transmission, since the advantages of the binary data format (less data to be transmitted) are neutralized by the additional query of the current measurement range.

Example 5, Setting the data format as a device parameter

In this example, we will show how the device parameters of the Script DAC Driver can be used for configuring the driver. Here, with the help of a parameter, a certain data format should be set for the data output of the device simulator and in case of an invalid input, an error message should be output. The example "example4.vbs" serves as the basis for this example.

At first, in the function SFD_Init, the parameter "DeviceParam1V" is evaluated and if required, an error message is output. The evaluation of the parameter could look like this:

```
'-------------------------------------------------
' Interpretation of the user-defined
' format designation
'-------------------------------------------------
sgFormatTypeT = UCase(DeviceParam1V)
lFormatM      = 0
Select Case sgFormatTypeT
  Case  "ASCII 1"
    lFormatM = 0
  Case  "ASCII 2"
    lFormatM = 1
  Case  "BINÄR"
    lFormatM = 5
  Case  Else
    ErrorP = "An invalid parameter has been input."
End Select
```

The parameter DeviceParam1V is checked for the different keywords. To prevent problems with upper/lower case, the

parameter "DeviceParam1V" is first converted to upper case. If a valid keyword is found, the corresponding output format of the device simulator is assigned to the variable "IFormatM". If an invalid keyword is specified, an error message is assigned to the parameter ErrorP.

Finally, the command for setting the output format also has to be adjusted. This is done with the following commands:

```
Call oUDIM.Write( "COF "+CStr(lFormatM))
Call oUDIM.Read()
```

Since the variable lFormatM is also required in the functions SFD_InitInChannel() and SFD_ReadChannel(), care should be taken that it is defined globally. For this purpose, the variable is defined outside the procedures, in the header of the file.

In the function SFD_InitInChannel(), there has to be a check whether the data is present in binary format. In this case, the scaling factor should be determined and passed to DIAdem.

```
FaktorP = 1
OffsetP = 0.
if ( lFormatM = 5 ) Then
  ' Query the amplitude that has been set for the channel :
  Call oUDIM.Write("AMP?"+CStr(ChannelNumberP))
  ' Read and evaluate the response :
  dAmplitudeT = CDbl(oUDIM.Read())
  FaktorP = dAmplitudeT/32767.
Endif
```

Finally, in the function SFD_ReadChannel(), the different functions for evaluating the different data formats should be implemented. This can happen, e.g. in the following Select Case instruction:

```
DataP = 0.
  Select Case lFormatM
    Case  0
      '-------------------------------------------------
      ' The data is read as ASCII. If "Parse"
      ' is called without the first parameter and
      ' the data is present as ASCII, then in "Parse"
      ' reading from the interface continues till the
      ' delimiter is reached
      '-------------------------------------------------
      DataP = oUDIM.Parse(,"%AD")
    Case  1
      '-------------------------------------------------
      ' The data is read as ASCII. The
      ' status and the measurement value, delimited by a semi-
      ' colon are read. If "Parse" is called without the first
      ' parameter set and the data is present in ASCII ,
      ' then in the "Parse", reading from the
      ' interface continues till the delimiter is
      ' reached
      '-------------------------------------------------
      aDataT = oUDIM.Parse(,"%AD%1C%AD")
      DataP  = CDbl(aDataT(1))
    Case  5
      '-------------------------------------------------
      ' The response is now available as a 1-Byte status 2-Byte value
      ' The 1-byte status value is defined as a comment
```

```
    ' and skipped. If "Parse" is called without
    ' the first parameter, the length of the data
    ' block to be read is automatically determined
    ' from the format description and the data is
    ' read from the port
    '-------------------------------------------------
    DataP = CDbl(oUDIM.Parse(,"%1C%2L") )
End Select
```

After you have saved this example as "example5.vbs", you can test it with the block diagram "samplingrate.dac". In the block for the Script DAC Driver, input one of the keywords "ASCII 1", "ASCII 2" or "binary" as Parameter1 and observe the reactions of the device simulator. Depending on the output format of the device simulator that has been set, the correct measurement data is always displayed now.

Optimization by exploiting the command set

The previous examples have shown that - at least in the case of the device simulator - by changing the data format for the data transmission, no significant improvement in the performance could be achieved. On comparing the theoretically possible polling rate with the actually reached polling rate (64/12), it was also seen that in this device, the bottleneck should not be sought in the actual data transmission. For this reason, the command set of the device simulator should be checked to see whether there are possibilities to increase the performance of the script with suitable commands of the device simulator. The device provides the possibility to use the command (TRG) to ask for the measurement data of all active channels in one go. This promises, first, a significant reduction of the data to be transferred, since here, for requesting the data, only 5 bytes per scan have to be transmitted (TRG<CR<LF). In the previous examples, it was always n*7 characters (MSV?x<CR><LF>)

Example 6, Setting the measurement mode

Here, the script "example5.vbs" is to be expanded so that by using Parameter2 of the block of the Script DAC Driver, the measurement mode can be set. As alternatives, the mode "Single value", that has already been implemented, and the mode "Scan" should be adjustable. To start with, therefore, in the function SFD_Init(), Parameter2 is checked and if required, an error message is output. The active measurement mode is saved in the global variable "Measurement mode". The initialization of the individual measurement channels remains unchanged. However, a few changes are necessary for the acquisition of the measurement

data. In the examples so far, the measurement data was requested for one channel each in the function SFD_ReadChannel() and evaluated. This also remains the same for the measurement mode "Single value". This part of the program is always executed in the function SFD_ReadChannel() whenever the measurement mode Single-value has been selected. For this reason, the function SFD_ReadChannel() is enhanced with the following statements:

```
DataP = 0.
if ( 0 = lMessModusM ) Then
  ' Requesting a measurement value from Channel1
  Call oUDIM.Write("MSV?"+CStr(ChannelNumberP))
  Select Case lFormatM

    The "old" instructions are here

  End Select
else

End If
```

The 'else' branch of this query initially remains empty. The driver initially works only in the measurement mode "Single value". In the measurement mode "Scan", the driver currently returns the value 0 for all channels (initialization value of DataP). You can now save the script under the name "example6.vbs" and, say, test it with the block diagram "samplingrate.dac".

The next step now consists of implementing the data acquisition in the measurement mode "Scan". Since the command "TRG" triggers the device simulator into simultaneously transmitting the data of all the active channels simultaneously, the data should be prompted for here not in the function SFD_ReadChannel(), but in the function SFD_GetScan(). In contrast to the function SFD_ReadChannel(), which is called once per scan for every channel, SFD_GetScan() is only called once per scan. SFD_GetScan() is called before the first call to SFD_ReadChannel().

Now, enhance the script with the body for the function SFD_GetScan.

```
Sub SFD_GetScan(ErrorP )End Sub--
```

Since this function is only to be processed for the measurement mode "Scan", the following line should be added first at the start of the procedure:

```
If ( 0 == lMessModusM ) Then Exit Sub
```

Thus, if the measurement mode is not 1 (for "Scan"), the procedure is quite again immediately.

Within the procedure, the measurement command "TRG" must first be transferred to the device. Then, the measurement data is read

from the device. Depending on the transfer format that has been set, the data should be read and prepared in different ways. Therefore, the further processing of the data is done within a Select statement in which the different data formats are processed.

```
  ' Requesting a scan :
  Call oUDIM.Write("TRG")
  ' Processing the data depending on the format
  Select Case lFormatM
    Case  0

    Case  1

    Case  5

  End Select
```

In this function, the measurement data is be read and prepared for further processing. The data should be saved in a script-global field and then returned to DIAdem in the procedure ReadChannel. The field is to be called "dMeasurementDataM".

Since the format "ASCII with status" transfers all the required data for sorting the measurement data in the clipboard, this format should be processed first. To start with, the measurement data should be read (as ASCII text). Then, this text should be evaluated. Depending on how many signals are active and whether the transmission is to be done with or without status, 1 or more values are to be evaluated. To keep the script readable, an additional procedure is generated, which carries out this evaluation.

```
Sub Simulator_Values(sgResponse, aData)
  Dim       sgaDataT
  Dim       lCountT,lValueT
  '-------------------------------------------------
  ' First, the text that has been read is split up
  ' according to the delimiter character (";")
  '-------------------------------------------------
  sgaDataT = Split(sgResponse,";")
  '-------------------------------------------------
  ' Then, a check is made whether the result is
  ' a single value or a field of values :
  '-------------------------------------------------
  if ( Not IsArray(sgaDataT) ) Then
    lCountT = 1
    Redim aData(1)
    sgaDataT = Trim(sgaDataT)
    aData(0) = CDbl(sgaDataT)
  Else
    lCountT = UBound(sgaDataT)
    Redim aData(lCountT)
    For lValueT = 0 To lCountT
      sgaDataT(lValueT) = Trim(sgaDataT(lValueT))
      aData(lValueT) = CDbl(sgaDataT(lValueT))
    Next
  End If
End Sub
```

In the procedure "SFD_GetScan","Simulator_Values" is called and then, the converted values are sorted in the field "dMesurementValuesM":

```
..
Case  1
  sgRespondT = oUDIM.Read()
  Call Simulator_Value(sgRespondT,aDataT)
  ' Umkopieren der Meßdaten
  For lValueT = 0 To UBound(aDataT) Step 2
    lIndexT              = aDataT(lValueT)
    dMesurementValuesM(lIndexT) =  CDbl(aDataT(lValueT+1))
  Next
Case  5
..
```

Now, in the function SFD_ReadChannel, the data is merely read from the clipboard and passed to DIAdem.

```
else
  DataP = dMesurementValuesM(ChannelNumberP)
end if
```

In the binary format, the same information is transmitted as in the transfer format "ASCII with status". Now, the problem with the binary format is that the script must determine, before reading the measurement data, how many bytes are transmitted by the device and have to be read. If the number of bytes specified in the read command is too large, the read command runs into a timeout and the achievable polling rate reduces drastically. As against that, if too little data is read, the measurement data in the channels gets mixed up and the data is unusable.

For counting the measurement values, it should be remembered that the same channel is possibly measured several times. This in turn results in the procedure SFD_InitInChannel being called several times with the same channel number. The counting of the channels that are actually transmitted by the device must be done correspondingly carefully. We do this in two steps:

In the first step, the channels should only be marked as active.

To do so, the field bActiveChannels is created in the global section of the script. Then, in the procedure "SFD_Init", the entries in this field are set to "false":

```
For lChannelNumberT = 0 to 9
  Call oUDIM.Write("ACH"+CStr(lChannelNumberT)+",0")
  Call oUDIM.Read()
  ' Mark channel as disabled
  bActiveChannels(lChannelNumberT) = False
Next
```

Finally, the active channels should be marked. To do so, the following line is added in the procedure "SFD_InitInChannel":

' Mark channel as active :

```
bActiveChannels(ChannelNumberP) = True
```

Then, we insert the new procedure "SFD_FinalInit". This procedure is called once for ending the initialization phase. In this procedure, we can now count the active channels in the field "bActiveChannels" and enter the count in a global variable "INumberChannelsM". The procedure then looks like this:

```
Sub SFD_FinalInit( ErrorP )
  ' Counting the active channels
  lNumberChannelsM = 0
  For lChannelNumberT = 0 to 9
    if ( bActiveChannels(lChannelNumberT) ) Then lNumberChannelsM =
lNumberChannelsM + 1
  Next
End Sub
```

Now, in the function SFD_GetScan(), the data can be read and evaluated in a purposeful manner.

First, we calculate how much data should be read. This data is then read and divided into channel information and measurement data. The results are then saved to the clipboard.

```
Case  5
      '--------------------------------------------------
      ' Binary
      '--------------------------------------------------
      sgRespondT = oUDIM.Read(lNumberChannelsM*3,0)
      sgFormatT  = CStr(lNumberChannelsM)+"(%1L%2L)"
      aBinDataT  = oUDIM.Parse(sgRespondT,sgFormatT)
      ' Recopying the measurement data
      For lValueT = 0 To UBound(aBinDataT) Step 2
        lIndexT              = aBinDataT(lValueT)
        dMesurementValuesM(lIndexT) =  CDbl(aBinDataT(lValueT+1))
      Next
End Select
```

Finally, the data acquisition for the data format "ASCII without status" should be implemented here. In this data format, all the measurement data, delimited by semicolons, is transferred in one data row. In this format, there is no information on which channels the relevant data belongs to. The device simulator returns a measurement value for every active measurement channel. Thus, e.g. if 5 channels are read, only the information on which channels are involved is currently missing. However, this information can be easily determined in the procedure "SFD_FinalInit". To start with, another global field "IChannelIndexM" is defined.

Then, the procedure "SFD_FinalInit" is extended in such a way that the following sequence is obtained:

```
Sub SFD_FinalInit( ErrorP )
  ' Counting the active channels
  lNumberChannelsM = 0
  For lChannelNumberT = 0 to 9
    if ( bActiveChannels(lChannelNumberT) ) Then
      ' The channel index is entered first :
      lChannelIndexM(lNumberChannelsM) = lChannelNumberT
      lNumberChannelsM = lNumberChannelsM + 1
    End If
  Next
End Sub
```

The function SFD_GetScan() is expanded as follows:

```
Case  0
  '-------------------------------------------------
  ' ASCII without Status
  '-------------------------------------------------
  sgRespondT = oUDIM.Read()
  Call Simulator Values(sgRespondT,aDataT)
  ' Recopying the measurement data
  For lValueT = 0 To UBound(aDataT)
    lIndexT              = lChannelIndexM(lValueT)
    dMesurementValuesM(lIndexT) =  CDbl(aDataT(lValueT))
  Next
Case  1
```

The measurement data is extracted from the device response by calling "Simulator_Values". Finally, the clipboard oDataT is re-copied, taking into account the channel indices in the field dMesuarmentDataM.

Evaluation of the Optimizations

Now, when you carry out some measurements with different settings with the block diagram "samplingrate.dac" and the script "example6.vbs", you will find that different polling rates can be achieved, depending on the combination of parameters. The slowest variant is undoubtedly the variant "Single value" with the data format "ASCII 2", whereas the variant "Scan" with the data format "binary" facilitates the highest polling rate. Whereas, in the case of data acquisition in mode "Single value", the increase in performance by selecting a suitable data format was comparatively small at 10%, in the scan-wise transmission, a much more significant improvement in performance of about 35% was reached. Comparisons of the different results show that in this case, the best performance increase was achieved by converting the algorithm for data acquisition. The fact that the performance can once again be improved significantly by a suitable choice of the transfer format shows that often, only a combination of suitable

optimization strategies can result in an optimum solution for the specific application.

Parameterization of the channels

The examples so far have shown, among other things, how the two device parameters of the Script DAC Driver can be used to control the general behavior of a device. Here, it will now be shown how even the individual channels of a device can be parameterized by Script DAC Driver.

Example 7, Setting the signal form

In this example, the signal form of a channel of the device simulator will be set with the help of the Script DAC Driver. The signal forms "Sine", "Rectangle" and "Triangle" can be selected on the device simulator. Alternatively to the setting of the signal form, it should also be possible to retain the signal form that has currently been set on the device simulator, if the parameter is empty. To do so, the contents of the parameter ParamP should first be analyzed in the function SFD_InitInChannel(). If a valid keyword is found, the corresponding command for setting the signal form is generated and then transmitted to the device simulator. If the parameter is empty, no command is transmitted to the device simulator either. If no valid keyword is found, an error message is generated. The corresponding expansion of the function SFD_InitInChannel() could look like this:

```
Command = "WAV" + CStr( ChannelNumberP ) + ","
Select Case ParamP
  Case "Rectangle"
    Command = Command + "1"
  Case "Triangle"
    Command = Command + "2"
  Case "Sine"
    Command = Command + "0"
  Case ""
    Command = ""
  Case Else
    Command = ""
    ErrorP = "An invalid value was specified as parameter for
channel "
             " + CStr( ChannelNumberP ) + _." + vbCrLf + _
             "Valid values are:" + vbCrLf + "Sine" + vbCrLf + _
             "Rectangle" + vbCrLf + "Triangle"
end Select

if Command <> "" then
  Call oUDI.Write( Command )
  Response = oUDI.Read()
end if
```

Example 8, Administering several parameters

The fact that for every block of the Script DAC Driver, there are only two parameters available and for every signal, only one parameter, for setting purposes, points foremost to a big restriction in the parameterizability of this driver. That this is not the case is shown with this example. All the parameters that are input in the block for the Script DAC Driver and passed to the script are strings of any desired size. This makes it possible to accommodate several setting parameters of the connected device in each of these parameters. In this example, example 7 will now be expanded in such a way that apart from the signal form, the amplitude of the signal can also be set via the parameters of the Script DAC Driver. For this purpose, the querying of the parameters will first be expanded to include the query for the keywords "Signal form=" and "Amplitude=". The individual parameters are delimited from each other by a semicolon. Spaces and tabs should not be permitted in the parameter. Since the amplitude that has been set at the relevant channel is queried during the course of channel initialization at the device simulator, the evaluation of the parameter should be done at the start of channel initialization.

You can view the implementation of this example in the script "example8.vbs". Here, the parameter of a channel is evaluated directly at the start of the function SFD_InitInChannel().

Task:

Enhance the "Example8a.vbs" with the settings of the parameter "Frequency", and "Unit".

Solution:

One possible solution of this task can be found in the file "solution8.vbs".

6.4 Devices-Examples

The examples in the following chapters show the connections of real measurement devices with the Script DAC Driver. Each example is preceded by a short explanation in which the device and the protocol with which the device is to be addressed is described in brief. Finally, we will point out special features of each script.

Initialization of the function generator HP 33120A

In this very simple example, a HP 33120A function generator from Hewlett Packard is to be set to the following parameters at the start of measurement :

Signal form: Sine

Frequency: 60 Hz

Amplitude: 2.5

The device is connected to the serial COM1 port. The transmission parameters set on the device side are 9600 baud, no parity and 8 data bits. One start bit and two stop bits are the factory settings.

The command for initializing the function generator to the values specified above is : "APPLy:Sin 60, 2.5". Information on this command as well as specifications for more setting options, transmission parameters and the mapping of the connection cable can be found in the manual of the device.

The driver should be so generated that the serial port can be set from the surface of the block. To do so, the name of the interface (e.g. COM1) should be given on the first parameter.

The script was defined for one output block. For this reason, the script must contain both the procedures SFD_SendScan and SFD_WriteChannel, even if both these procedures do not have any function. The device is parameterized only within the function SFD_Init. The following listing shows the entire script up to the procedure SFDU_COMInit, which can be automatically generated with the function "New Script".

The script with the solution for this task is called "Hp33120a_init.vbs"; the block diagram file is called "hp33120a_init.dac".

```
'--------------------------------------------------------------------
' Global parameters
'--------------------------------------------------------------------
Dim     oUDIM
'--------------------------------------------------------------------
' Function for initializing a device
'--------------------------------------------------------------------
'--------------------------------------------------------------------
' SFD_Init
' Purpose             : This procedure is called during the start of
measurement
'--------------------------------------------------------------------
' DeviceParam1V       | First parameter that can be input by the user
'                     | in the DAC block
' DeviceParam2V       | Second parameter that can be input by the user
'                     | in the DAC block
```

```
' ErrorP              | Variable for returning an error message. If
this
'                     | variable is set, DIAdem stops the measurement.
'-----------------------------------------------------------------------
Sub SFD_Init( sgCOMName, DeviceParam2V, ErrorP )
  ' Creating the UDI-object
  Set oUDIM=CreateObject("DIAdem.SFD.UDI" )
  '-------------------------------------------------
  ' Opening and initializing the serial
  ' port
  '-------------------------------------------------
  Call SFDU_COMInit(oUDIM,sgCOMName+",9600,N,8,2")
  ' Setting delimiters
  Call oUDIM.ParamSet("DELIMITER",vbLF)
  ' The remote command has to be sent first !
  Call oUDIM.Write("System:Remote")
  ' Set signal form, frequency (60 Hz) and amplitude (2.5):
  Call oUDIM.Write("APPLy:Sin 60, 2.5")
  ' Set device back to local operation
  Call oUDIM.Write("System:Local")
  ' Close port
  oUDIM.Close()
  ' Delete object :
  Set oUDIM = Nothing
End Sub
'-----------------------------------------------------------------------
' SFD WriteChannel
' Purpose               : This method remains without a function in
this example
'-----------------------------------------------------------------------
' DoneP               | Flag that shows whether the value could be
output
'-----------------------------------------------------------------------
Sub SFD WriteChannel( ChannelNumberP, sgParamName, DataP, DoneP,
ErrorP )
  DoneP = 1
End Sub
'-----------------------------------------------------------------------
' SFD_SendScan
' Purpose              : This method remains without a function in
this example
'-----------------------------------------------------------------------
' DoneP               | Flag that shows whether the value could be
output
'-----------------------------------------------------------------------
Sub SFD_SendScan( DoneP, ErrorP )
  DoneP = 1
End Sub
```

Online parameterization of the function generator HP 33120

In this second example for the function generator HP 33120A from Hewlett Packard, during the measurement, the user should be able to directly set the following parameters from DIAdem in the specified ranges:

Signal form : Sine, rectangular, sawtooth, triangular

Frequency: 0-250 Hz

Amplitude: 0-7

The device is connected to the serial COM1 port. The transmission parameters set on the device side are 9600 baud, no parity and 8 data bits. One start bit and two stop bits are the factory settings.

The driver should be so generated that the serial port can be set from the surface of the block. To do so, the name of the interface (e.g. COM1) should be given on the first parameter.

The script was defined for one output block. The following listing shows the entire script up to the procedure SFDU_COMInit, which can be automatically generated with the function "New Script".

The script with the solution for this task is called "Hp33120a_DAC.vbs"; the block diagram file is called "Hp33120a_DAC.dac".

```
'--------------------------------------------------------------------
' Global parameters
'--------------------------------------------------------------------
Dim     oUDIM
Dim     dFrequencyM
Dim     dAmplitudeM
Dim     bChangedValuesM
Dim     lWaveformM
Dim     sgaWaveformNames(3)
' Initialize to invalid values
dFrequencyM         = vbEmpty
dAmplitudeM         = vbEmpty
lWaveformM          = vbEmpty
bChangedValuesM     = true
' Default-set the names of the signal forms
sgaWaveformNames(0) = "Sin" ' Sine-Form
sgaWaveformNames(1) = "SQU" ' Square (Rectangle)
sgaWaveformNames(2) = "TRI" ' Triangle
sgaWaveformNames(3) = "RAMP"' Sawtooth
'--------------------------------------------------------------------
' Functions for outputting data
'--------------------------------------------------------------------
'--------------------------------------------------------------------
' SFD_WriteChannel
' Purpose             : Outputting a value for the channel number
"ChannelNumberP"
'--------------------------------------------------------------------
' ChannelNumberP      | Channel number from the Block dialog
' ParamP              | User-defined variable from the Block dialog
' DataP               | Value to be output
' DoneP               | Code that indicates whether the value could be
output
'                     | This variable must be set to a valid value
(e.g 1) .
'                     | Else, DIAdem stops the measurement
' ErrorP              | Variable for returning an error message. If
this
'                     | variable is set, DIAdem stops the measurement.
'--------------------------------------------------------------------
Sub SFD WriteChannel( ChannelNumberP, sgParamName, DataP, DoneP,
ErrorP )
  Select Case LCase(sgParamName)
    '-------------------------------------------------
    ' New value for the signal frequency
    '-------------------------------------------------
```

```
    Case "frequency"
      if ( dFrequencyM <> DataP ) Then
        bChangedValuesM = true
        dFrequencyM     = DataP
      End If
    '-------------------------------------------------
    ' New value for the signal amplitude
    '-------------------------------------------------
    Case "amplitude"
      if ( dAmplitudeM <> DataP ) Then
        bChangedValuesM = true
        dAmplitudeM     = DataP
      End If
    '-------------------------------------------------
    ' New value for the signal form
    '-------------------------------------------------
    Case "waveform"
      if ( lWaveformM <> CInt(DataP) ) Then
        bChangedValuesM = true
        lWaveformM      = CInt(DataP)
        lWaveformM      = CInt(SFDU Min(lWaveformM,3))
        lWaveformM      = CInt(SFDU_Max(lWaveformM,0))
      End If
  End Select
  DoneP = 1
End Sub
'--------------------------------------------------------------------
' SFD SendScan
' Purpose              : Output of a "Scan". For multi-channel
devices, this
'                        procedure can be used for writing a scan of
all the
'                        values. The data is previously passed to the
script
'                        in the procedure SFD WriteChannel of DIAdem.
'--------------------------------------------------------------------
' DoneP               | Flag that indicates whether the value could be
output.
'                     | This variable must be set to a valid value
(e.g 1).
'                     | Otherwise, DIAdem stops the measurement.
' ErrorP              | Variable for returning an error message. If
this
'                     | variable is set, DIAdem stops the measurement.
'--------------------------------------------------------------------
Sub SFD_SendScan( DoneP, ErrorP )
  Dim     sgMessageT,sgWaveform
  ' Have the values changed ?
  if ( bChangedValuesM ) Then
    sgMessageT = "APPLy:"+sgaWaveformNames(lWaveformM)+"
"+CStr(dFrequencyM)+", "+CStr(dAmplitudeM)
    Call oUDIM.Write(sgMessageT)
    ' The values are now up-to-date again
    bChangedValuesM = false
  End If
  DoneP = 1
End Sub
'--------------------------------------------------------------------
' Function for initializing a device
'--------------------------------------------------------------------
'--------------------------------------------------------------------
' SFD Init
' Purpose             : This procedure is called during the start of
measurement
'--------------------------------------------------------------------
' DeviceParam1V       | First parameter that can be input by the user
'                     | in the DAC block
```

```
' DeviceParam2V       | Second parameter that can be input by the user
'                     | in the DAC block
' ErrorP              | Variable for returning an error message. If
this
'                     | variable is set, DIAdem stops the measurement
'-----------------------------------------------------------------------
Sub SFD Init( sgCOMName, DeviceParam2V, ErrorP )
'-----------------------------------------------
  ' Creating the UDI object
  '-----------------------------------------------
  Set oUDIM=CreateObject("DIAdem.SFD.UDI" )
  '-----------------------------------------------
  ' Opening and initializing the serial
  ' port
  ' The following parameters are variable on the device-side
  ' - Transmission rate
  ' - Number of data bits
  ' - Parity
  'The following parameters are always the same
  ' - 1 Start-Bit
  ' - 2 Stop-Bits
  '-----------------------------------------------
  Call SFDU_COMInit(oUDIM,sgCOMName+",9600,N,8,2")
  '
  Call oUDIM.ParamSet("DELIMITER",vbLF)
  Call oUDIM.ParamSet("TIMEOUT"  ,"1000")
  ' The Remote command has to be transmitted first!
  Call oUDIM.Write("System:Remote")
End Sub

'-----------------------------------------------------------------------
' Procedure for de-initializing the devices
'-----------------------------------------------------------------------
'-----------------------------------------------------------------------
' SFD DeInit
' Purpose             :
'-----------------------------------------------------------------------
' ErrorP              | Variable for returning an error message. If
this
'                     | variable is set, DIAdem stops the measurement
'-----------------------------------------------------------------------
Sub SFD DeInit(ErrorP )
  ' Reset device to operation via Panel
  Call oUDIM.Write("System:Local")
  ' Close port
  oUDIM.Close()
  ' Onjekt loeschen
  Set oUDIM = Nothing
End Sub

'-----------------------------------------------------------------------
' Two small utility functions (Min- and Max-determination)
'-----------------------------------------------------------------------
Function  SFDU Min(Value1,Value2)
  SFDU Min = CDbl(Value2)
  if ( CDbl(Value1) < CDbl(Value2) ) Then SFDU_Min = CDbl(Value1)
End Function

Function  SFDU_Max(Value1,Value2)
  SFDU_Max = CDbl(Value2)
  if ( CDbl(Value1) > CDbl(Value2) ) Then SFDU Max = CDbl(Value1)
End Function
```

Reading a GPS receiver (Motorola binary format)

In this example, a GPS that is constructed on a Motorola ONCORE is read by a receiver. The device transmits the data in Motorola binary format.

The transmission parameters are 9600 baud, no parity, 8 data bits, 1 start bit and 1 stop bit.

This example uses the facility of the function "Parse" for assigning names to the values during the reading. Subsequently, the data can be accessed by their names.

The driver should be so constructed that at first, all the information from the GPS receiver is read.

The user should then be able to select the data that he wishes to measure with their names in the parameterization dialog of the Script DAC Driver.

During the initialization of the measurement, the format description for the "Parse" function is "assembled" as follows:

```
vPFormat =            "%1L<month>%1L<day>%2L<year>"
vPFormat = vPFormat +
"%1L<hours>%1L<minutes>%1L<seconds>%4L<fract.Seconds>"
vPFormat = vPFormat +
"%4L<latitude>%4L<longitude>%4L<GPS Height>%4L<MSL Height>"
vPFormat = vPFormat + "%2U<velocity>%2L<heading>"
vPFormat = vPFormat + "%2L<CurrentDOP>%1L<DOPType>"
vPFormat = vPFormat + "%1L<SatVisible>%1L<SatTracked>"
vPFormat = vPFormat +
"8(%1L<SatID>%1L<ChnTrackMode>%1L<SignalStrength>%1L<ChannelStateFlag>
)"
```

The details of the format are explained in further detail with some examples:

In the first line, the first section of the format is defined as "%1L<month>%1L<day>%2L<year>". When this format is interpreted by the procedure "Parse" to evaluate the data, then the first data byte ("1L") is interpreted as an integer value and internally assigned to a variable with the name "month". The next data byte (1L) is also interpreted as an integer value and assigned to the variable "day". Both the following data bytes ("2L") are interpreted as integral values and assigned to the variable "year". The last part of the format "8(%1L<SatID>.....%1L<ChannelStateFlag>)" describes the section of the data that is the same for each of the maximum 8 evaluated satellites. Format sections that are repeated can simply be described with "*Repeat factor(format*)".

In this case, therefore, the format section within the brackets is repeated eight times. The specified variable names in brackets are automatically extended into a unique name by appending a number. Thus, the ID of the first satellite (variable name specified in the format with "SatID") is saved in the variable "SatID1", the ID of the second satellite in the variable "SatID2" etc.

The following listing shows the entire script up to the procedure SFDU_COMInit, which can be automatically generated with the function "New Script".

The script with the solution for this task is called " GPS_ONCORE_BINARY.vbs"; the block diagram file is called " GPS_ONCORE_BINARY.dac".

```
'--------------------------------------------------------------------
' Global parameters
'--------------------------------------------------------------------
Dim                      oUDIM
Dim     sgFormatM,aGlobalScanM
'--------------------------------------------------------------------
' SFD_Init
' Purpose              : Function called when starting measurement
'--------------------------------------------------------------------
' sgCOMName           | Name of the serial interface at which the GPS
'                     | receiver is connected
' DeviceParam2V       | No function
' ErrorP              | Variable with which error messages can be
returned to
'                     | DIAdem
'--------------------------------------------------------------------
Sub SFD_Init( sgCOMName,DeviceParam2V, ErrorP )
  ' Creating the UDI object
  Set oUDIM=CreateObject("DIAdem.SFD.UDI" )
  ' Opening and initializing the interface
  Call SFDU_COMInit(oUDIM,sgCOMName+",9600,N,8,1")
  '-----------------------------------------------
  ' Initializing the format description
  '-----------------------------------------------
  ' Datum
  sgFormatM =            "%1L<month>%1L<day>%2L<year>"
  ' Zeit
  sgFormatM = sgFormatM +
"%1L<hours>%1L<minutes>%1L<seconds>%4L<fract.Seconds>"
  ' Postion
  sgFormatM = sgFormatM +
"%4L<latitude>%4L<longitude>%4L<GPS_Height>%4L<MSL_Height>"
  ' geschwindigkeit
  sgFormatM = sgFormatM + "%2U<velocity>%2L<heading>"
  ' Geometrie
  sgFormatM = sgFormatM + "%2L<CurrentDOP>%1L<DOPType>"
  ' Sateliten Status
  sgFormatM = sgFormatM + "%1L<SatVisible>%1L<SatTracked>"
  ' Sateliten Daten
  sgFormatM = sgFormatM +
"8(%1L<SatID>%1L<ChnTrackMode>%1L<SignalStrength>%1L<ChannelStateFlag>
)"
End Sub

'--------------------------------------------------------------------
' SFD_GetScan
```

```
' Purpose             : Reading a data block from the GPS-receiver
'-------------------------------------------------------------------
' ErrorP              | Variable for returning an error message. If
this
'                     | variable is set, DIAdem stops the measurement.
'-------------------------------------------------------------------
Sub SFD_GetScan(ErrorP )
  Dim vData
  Call GPS_DataBlockRead(oUDIM,vData)
  CAll GPS_ProcessData(oUDIM,vData)
End Sub
'-------------------------------------------------------------------
' SFD_ReadChannel
' Purpose             : Reading a value
'-------------------------------------------------------------------
' ChannelNumberP      | The channel number from the dialog
' sgValuename         | The name of the value desired by the user
' DataP               | The variable in which the value is returned to
DIAdem
' ErrorP              | Variable using which error messages can be
returned to
'                     | DIAdem.
'-------------------------------------------------------------------
Sub SFD_ReadChannel( ChannelNumberP, sgValuename, DataP, ErrorP )
  DataP = 0
  '-------------------------------------------------
  ' If no name has been specified for the value,
  ' 0 is returned
  '-------------------------------------------------
  if ( IsEmpty(sgValuename) ) Then Exit Sub
  ' Ask for the value using the name of the UDI object
  DataP = oUDIM.NamedValueGet(sgValuename)
End Sub
'-------------------------------------------------------------------
' Function for evaluating the GPS Data
'-------------------------------------------------------------------
'-------------------------------------------------------------------
' GPS_DataBlockRead
' Purpose             : Reading the data from the GPS receiver
'                       The command "Ea" is used to ask for data from
'                       the receiver
'-------------------------------------------------------------------
' vData               | Buffer to which the data that has been read is
returned
'-------------------------------------------------------------------
Sub GPS_DataBlockRead(oUDI,vData)
  Dim vCommand
  ' Send a command to ask for the data
  vCommand = "Ea"+chr(0)
  vCommand = "@@"+vCommand + chr(GPS_CheckSum(vCommand))+vbCR+vbLF
  Call oUDI.ParamSet("Delimiter","")
  Call oUDI.Write(vCommand)
  '-------------------------------------------------
  ' Read till "@@Ea" indicates the start of the
  ' transmitted data
  '-------------------------------------------------
  Call oUDI.ParamSet("Delimiter","@@Ea")
  Call oUDI.Read()
  Call oUDI.ParamSet("Delimiter","")
  ' Read the entire data block (72 bytes)
  vData = oUDI.Read(72,0)
End Sub
'-------------------------------------------------------------------
' GPS_ProcessData
' Purpose             : Evaluating the data that has been read with
the procedure
```

```
'                       "Parse" followed by execution of all the
necessary
'                       scaling.
'                       The scaling parameters (Offset, factors) have
been
'                       taken from the documentation for the device
'-------------------------------------------------------------------
' vData                | Data read from I/O
'-------------------------------------------------------------------
Sub GPS_ProcessData(oUDI,vData)
  Call oUDI.Parse(vData,vPFormat)
  '-------------------------------------------------
  ' Scaling, offset displacement and rounding of the
  ' decimal places are carried out in the function
  ' GPS_ScaleValue
  '-------------------------------------------------
  Call GPS_ScaleValue(oUDI,"Latitude",  0.,(90./324000000.),2)
  Call GPS_ScaleValue(oUDI,"Longitude", 0.,(180./648000000.),2)
  Call GPS_ScaleValue(oUDI,"GPS_Height",0.,0.01,2)
  Call GPS ScaleValue(oUDI,"MSL Height",0.,0.01,2)
  Call GPS ScaleValue(oUDI,"velocity",  0.,0.01,2)
  Call GPS_ScaleValue(oUDI,"heading",   0.,0.1, 1)
  Call GPS_ScaleValue(oUDI,"CurrentDOP",0.,0.1, 1)
End Sub
'-------------------------------------------------------------------
' GPS ScaleValue
' Purpose             : Scaling, offset displacement and rounding of
the decimal
'                       places of the data that has been read.
'-------------------------------------------------------------------
' Name                | Name of the variable to be scaled
' Offset              | Offset value
' Scale               | Scaling factor
' Digits              | Number of decimal places to which rounding is
to be done
'-------------------------------------------------------------------
Sub GPS_ScaleValue(oUDI,Name,Offset,Scale,Digits)
  vValue = oUDI.NamedValueGet(Name)
  vValue = (vValue + Offset)*Scale
  vValue = Round(vValue,Digits)
  Call oUDI.NamedValueSet(Name,vValue)
End Sub
'-------------------------------------------------------------------
' GPS_CheckSum
' Purpose             : Calculation of the checksum defined for the
Motorola
'                       binary format
'-------------------------------------------------------------------
' vData               | Data for which the checksum should be
calculated
'-------------------------------------------------------------------
Function GPS_CheckSum(vData)
  CharAct = Asc(Mid(vData,1,1))
  For L = 2 to Len(vData)
    CharAct = CharAct Xor Asc(Mid(vData,L,1))
  Next
  GPS_CheckSum = CharAct
  Exit Function
End Function
```

Liquid level indicator with a sensor from TEMIC

The liquid level indicator from TEMIC is used, among other things, to continuously acquire the liquid level in the fuel tanks on test rigs in the automobile industry.

The device is addressed via the serial port. Since the parameters of the serial port are always set to 14400 bauds, no parity, 8 data bits and one stop bit for communication with this device, these parameters have been coded in the script as constants.

Only the number of the serial port via which the device is to be addressed can be set by the user via Parameter1 of the device parameters in the properties of the block in the block diagram (more on this in chapter 1.2 ("The Operation of the Script DAC Driver").

The device is initialized by sending an "R". The device should ask for the data (current measurement values) by transmitting a " " (space). The device responds with 8 lines in the following form :

U=1<Tab>T=0.001<Tab>L=0.002<Tab>Q=0.003<Tab>P=0.004<CRLF>

Only the first two lines should be evaluated.

The procedure SFDU_COMInit was generated automatically using the function "New Script". This function has been deliberately omitted in the following print of the script.

```
'--------------------------------------------------------------------
' Global parameters
'--------------------------------------------------------------------
' oUDIM               | UDI-Object to communicate with device
'--------------------------------------------------------------------
Dim   oUDIM,dValues1M(4),dValues2M(4),dNoValueM
Set     oUDIM = CreateObject( "DIAdem.SFD.UDI" )
dNoValueM = 9.9E+34
'--------------------------------------------------------------------
' SFD_Init
' Purpose             : Initialize UDI object
'                       Parameters for serial I/O set to 14400 Baud,No
parity,
'                       8 data bits, 1 stop bit
'                       Name of parallel port coming from user
definition in
'                       block parameters. Must be in the form "COM1",
"COM2"...
'--------------------------------------------------------------------
' sgComName           | Name of serial port (COM1, COM2...) defined in
'                     | DIAdem block dialog
' sgDeviceparam2      | not used
' sgError             | not used
'--------------------------------------------------------------------
Sub SFD Init( ByRef sgComName, ByRef sgDeviceparam2, ByRef sgError )
  Dim     iValueT
  ' Initialize serial I/O
  Call SFDU_COMInit(oUDIM,sgComName+",14400,N,8,1")
```

```
  ' Read all remaining "junk" from port
    Call oUDIM.Read(1000,0)
  ' Set timeout to 100 microseconds :
  Call oUDIM.ParamSet("Timeout",1000)
  '-------------------------------------------------
  ' Set delimiter to carriage-return + linefeed
  '-------------------------------------------------
  Call oUDIM.ParamSet("Delimiter",(vbCR+vbLF))
  '-------------------------------------------------
  ' Request for data in relative mode
  ' Write one byte without delimiter !
  '-------------------------------------------------
  Call oUDIM.Write("R",,0)
  '-------------------------------------------------
  ' Read answer ....
  '-------------------------------------------------
  Dim sgEchoT
  sgEchoT = oUDIM.Read(1000,0)
  '-------------------------------------------------
  ' Set desired time out
  '-------------------------------------------------
  Call oUDIM.ParamSet("Timeout",1000)
  ' Initialize data
  For iValueT = 0 To 3
    dValues1M(iValueT) = 0.
    dValues2M(iValueT) = 0.
    ' For Testing :
    dValues1M(iValueT) = CDbl(iValueT)
    dValues2M(iValueT) = dValues1M(iValueT) + 4.
  Next
End Sub
'---------------------------------------------------------------------
' SFD DeInit
' Purpose             : "Shut down" communication. Called once at the
end of
'                       a measurement task
'---------------------------------------------------------------------
Sub SFD_DeInit(ErrorP )
  oUDIM.Close()
  ' Free active-x object
  Set oUDIM = Nothing
End Sub
'---------------------------------------------------------------------
' SFD_GetScan
' Purpose             : Read one scan of data
'                       Mouse data consists of three bytes. First byte
with
'                       button information on bit's 4 and 5 (counting
from 0)
'                       and dx in second byte dy in third byte
'---------------------------------------------------------------------
' ErrorP              | Errortext
'---------------------------------------------------------------------
Sub SFD GetScan(ErrorP )
  Dim   sgLine1T,sgLine2T,sgJustSkipT
  Dim   lCountT
  ' Request for data :
  Call oUDIM.Write(" ",,0)
  ' Read up to delimiter :
  sgLine1T = oUDIM.Read(1000,1)
  sgLine2T = oUDIM.Read(1000,1)
  '-------------------------------------------------
  ' Device sends 8 lines (one for each sensor)
  ' reading lines 3-8
  '-------------------------------------------------
  For lCountT = 3 To 8
    sgJustSkipT = oUDIM.Read(1000,1)
```

```
  Next
  '-------------------------------------------------
  ' Convert strings to data
  '-------------------------------------------------
  Call TEMIC_Values(sgLine1T, dValues1M, dNoValueM)
  Call TEMIC_Values(sgLine2T, dValues2M, dNoValueM)
End Sub
'---------------------------------------------------------------------
' TEMIC_Values
' Purpose             : Parse values from string
'---------------------------------------------------------------------
' sgTEMICData         | String read from TEMIC device
' daValues            | Array to put values in (size is 4)
' sgError             | not used
'---------------------------------------------------------------------
Sub TEMIC_Values(sgTEMICData, daValues, dNoValue)
  On Error resume Next
  Dim   sgDataT,aArrayT,lSizeT,iTokenT,lPosT,sgTokenT
  '
  sgDataT        = Trim(sgTEMICData)
  if ( 0 >= Len(sgDataT) ) Then Exit Sub
  ' Split string into token :
  aArrayT = Split(sgDataT,vbTab, -1, 1)
  lSizeT = UBound(aArrayT)
  if ( 4 < lSizeT ) Then lSizeT =4
  '-------------------------------------------------
  ' Initialize result
  '-------------------------------------------------
  For iTokenT = 0 To 4
    daValues(iTokenT) = dNoValue
  Next
  '-------------------------------------------------
  ' Loop through all tokens
  '-------------------------------------------------
  For iTokenT = 0 To lSizeT
    '-----------------------------------------------
    ' Trim token and remove "??="
    '-----------------------------------------------
    sgTokenT = Trim(aArrayT(iTokenT))
    lPosT = InStr(1,sgTokenT,"=")
    if ( 0 < lPosT ) Then sgTokenT = Right(sgTokenT,Len(sgTokenT)-
lPosT)
    '-----------------------------------------------
    ' Convert from string to double
    '-----------------------------------------------
    daValues(iTokenT) = CDbl(sgTokenT)
  Next
End Sub
'---------------------------------------------------------------------
' SFD_ReadChannel
' Purpose             : Called from DIAdem to get one scan of data.
The name
'                       of the channel is give in sgChannelParam. The
'                       parameter name is case sensitiv
'---------------------------------------------------------------------
Sub SFD_ReadChannel(ByRef lChannelNumber,sgChannelParam,ByRef
dValue,ErrorP )
  Select Case  sgChannelParam
    Case "Temperature1" dValue = dValues1M(1)
    Case "Level1"       dValue = dValues1M(2)
    Case "Quality1"     dValue = dValues1M(3)
    Case "P1"           dValue = dValues1M(4)
    Case "Temperature2" dValue = dValues2M(1)
    Case "Level2"       dValue = dValues2M(2)
    Case "Quality2"     dValue = dValues2M(3)
    Case "P2"           dValue = dValues2M(4)
    Case Else           dValue = dNoValueM
```

```
  End Select
End Sub
```

CO_2 Measurement device NGA 2000 from FISHER-ROSEMOUNT

The measurement apparatus NGA 200 from Fisher-Rosemount is used to acquire CO2 concentrations. The device is addressed via the serial port. Communications with the device are done using the standardized AK protocol. The script for this example can be found in the file NGA2000_MLT.VBS.

The procedures AK_Query, AK_Values, AK_ReplyRead, AK_HeaderStrip and SFDU_COMInit were generated automatically using the function "New Script“. These functions have been deliberately omitted in the following print of the script.

```
'------------------------------------------------------------------
' Global parameters
'------------------------------------------------------------------
' vValueBuffer        | Buffer for measured values
'                     | (0) = "current level "
'                     | (1) = "filling rate"
'                     | (2) = "consumption"
' vValueValid         | Flags indicating whether the values in
vValueBuffer
'                     | are valid
'------------------------------------------------------------------
Dim
daPressureM(),daTemperature(),daConcentration(),daFlowThrough(),dNoVal
ueM
Dim   oUDI,STX,ETX,DontCare,oDBGList
Dim   ErrorState,ReplyData,dNoValue

STX       = Chr(2)
ETX       = Chr(3)
DontCare  = Chr(32)
dNoValueM = 9.9E+34
'------------------------------------------------------------------
' SFD ReadChannel
' Purpose             : Read one value (one channel)
'------------------------------------------------------------------
' ErrorP              | Errortext
' ChannelNumberP      | channelnumber (set from DIAdem, not used in
this driver)
' ParamP              | Parameter which is defined from the user for
each channel
'                     | Used to access the measured values by name
'                     | valid values are "Temp(1)"-"Temp(4)","CO2(1)"-
"Co2(4)"
'------------------------------------------------------------------
Sub SFD ReadChannel( ChannelNumberP, ParamP, DataP, ErrorP )
  ' This function must be implemented for data acquisition!
  DataP = dNoValueM
  Select Case ParamP
    Case "Temp(1)"
      DataP = daTemperature(0)-273.
    Case "Temp(2)"
      DataP = daTemperature(1)-273.
    Case "Temp(3)"
      DataP = daTemperature(2)-273.
```

```
    Case "Temp(4)"
      DataP = daTemperature(3)-273.
    Case "CO2(1)"
      DataP = daConcentration(0)/10000.
    Case "CO2(2)"
      DataP = daConcentration(1)/10000.
    Case "CO2(3)"
      DataP = daConcentration(2)/10000.
    Case "CO2(4)"
      DataP = daConcentration(3)/10000.
    Case "Flow(1)"
      DataP = daFlowThrough(0)
    Case "Flow(2)"
      DataP = daFlowThrough(1)
    Case "Flow(3)"
      DataP = daFlowThrough(2)
    Case "Flow(4)"
      DataP = daFlowThrough(3)
    Case "Press(1)"
      DataP = daPressureM(0)/1.E6
    Case "Press(2)"
      DataP = daPressureM(1)/1.E6
    Case "Press(3)"
      DataP = daPressureM(2)/1.E6
    Case "Press(4)"
      DataP = daPressureM(3)/1.E6
        End Select
End Sub


'--------------------------------------------------------------------
' SFD_GetScan
' Purpose             : Read one scan of data
'--------------------------------------------------------------------
' ErrorP              | Errortext
'--------------------------------------------------------------------
Sub SFD_GetScan(ErrorP )
  Dim lErrorStateT,sgReplyDataT,sgMessageT
  ' Request for "Druck" :
  CALL AK Query(oUDI,"ADRU",0,"",true,lErrorStateT,sgReplyDataT)
  Call AK Values(sgReplyDataT, daPressureM,dNoValueM)
  ' Request for "Temperatur" :
  CALL AK_Query(oUDI,"ATEM",0,"",true,lErrorStateT,sgReplyDataT)
  Call AK_Values(sgReplyDataT, daTemperature, dNoValueM)
  ' Request for "Konzentration" :
  CALL AK_Query(oUDI,"AKON",0,"",true,lErrorStateT,sgReplyDataT)
  Call AK Values(sgReplyDataT, daConcentration, dNoValueM)
  ' Request for "Durchfluss" :
  CALL AK_Query(oUDI,"ADUF",0,"",true,lErrorStateT,sgReplyDataT)
  Call AK_Values(sgReplyDataT, daFlowThrough, dNoValueM)
End Sub

'--------------------------------------------------------------------
' SFD Init
' Purpose             : Functions to Initialize the device interface
'--------------------------------------------------------------------
' DeviceParam1V       | First description string from script block
(entered by user)
' DeviceParam2V       | Second description string from script block
(entered by user)
' ErrorP              | Errortext
'--------------------------------------------------------------------
Sub SFD_Init( DeviceParam1V, DeviceParam2V, ErrorP )
  Dim       ErrorStateT,ReplyDataT
  ' Initialize global UDI object :
Set     oUDI     = CreateObject( "DIAdem.SFD.UDI" )
  Call SFDU_COMInit(oUDI,DeviceParam1V)
```

```
  Call oUDI.ParamSet("Timeout",1000)
  ' Set delimiter to ETX
  Call oUDI.ParamSet("Delimiter",ETX)
  ' Initialize device
  CALL AK_Query(oUDI,"SRES",0,"",true,ErrorState,ReplyData)
  ' Check for error :
  if ( 0 <> ErrorState ) Then ErrorP = "Error while initializing
device NGA 2000 !"
End Sub
'--------------------------------------------------------------------
' SFD_DeInit
' Purpose             : Close device
'--------------------------------------------------------------------
' ErrorP              | Errortext
'--------------------------------------------------------------------
Sub SFD_DeInit(ErrorP )
  ' Close device
  Call oUDI.Close()
End Sub
```

Fuel weighing scale 733 S from AVL

The fuel consumption of an internal combustion engine can be determined with the fuel weighing scale 733S from M/s AVL. The weighing scale works according to the gravimetric principle of measurement. The engine is supplied fuel from a measurement container whose weight is determined continuously.

The weighing scale is connected to the serial port. Communications with the device are done using the standardized AK protocol. The script for this example can be found in the file AVL_733S.VBS; the block diagram in the file AVL_733S.DAC.

In this simple example, the weighing scale is initialized with the commands "STBY" (cancel all the running procedures) and "SINT" (start or restart measurement). During the measurement, the "AWRT" command is used to read the current filing level and the momentary value.

In a real test rig environment, the weighing scale is additionally calibrated, washed and if required, individual channels are further parameterized. The relevant command sequences are not present here, but can be easily inserted in the procedure FSD_Init.

The procedures AK_Query, AK_Values, AK_ReplyRead, AK_HeaderStrip and SFDU_COMInit were generated automatically using the function "New Script“. These functions have been deliberately omitted in the following print of the script.

```
'--------------------------------------------------------------------
' Global parameters
'--------------------------------------------------------------------
Dim     oUDIM
Dim     STX,ETX,DontCare
```

```
Dim     dNoValueM
Dim     daActValuesM()
STX       = Chr(2)
ETX       = Chr(3)
DontCare  = Chr(32)
dNoValueM = 9.9E+34
'--------------------------------------------------------------------
' Functions for reading data
'--------------------------------------------------------------------
'--------------------------------------------------------------------
' SFD_GetScan
' Purpose             : Reading the current values. The AK-command
AWRT
'                       is used to read the current values for the
(filling)
'                       level and the current value
'--------------------------------------------------------------------
' ErrorP              | Variable for returning an error message. If
this
'                     | variable is set, DIAdem stops the measurement
'--------------------------------------------------------------------
Sub SFD_GetScan(ErrorP )
  Dim     lErrorStateT,sgReplyDataT
  ' Request values
  CALL AK_Query(oUDIM,"AWRT",0,"",true,lErrorStateT,sgReplyDataT)
  ' Evaluate ...
  Call AK Values(sgReplyDataT, daActValuesM, dNoValueM)
End Sub

'--------------------------------------------------------------------
' SFD_ReadChannel
' Purpose             : Reading a value for the channel number
"ChannelNumberP"
'--------------------------------------------------------------------
' ChannelNumberP      | Channel number from the Block dialog (Not
used)
' sgValueName         | Name of the value
' DataP               | Variable for returning the new channel value.
' ErrorP              | Variable for returning an error message.
'--------------------------------------------------------------------
Sub SFD ReadChannel( ChannelNumberP, sgValueName, DataP, ErrorP )
  Select Case LCase(sgValueName)
    Case "level"
      DataP = CDbl(daActValuesM(0))
    Case "Curantvalue"
      DataP = CDbl(daActValuesM(1))
    Case Else
      '-------------------------------------------------
      ' If an unsupported value name is specified
      ' a "NoValue" is returned here
      '-------------------------------------------------
      DataP = dNoValueM
  End Select
End Sub

'--------------------------------------------------------------------
' SFD_Init
' Purpose             : Initialization of the device
'--------------------------------------------------------------------
' sgCOMName           | Name of the serial port to which the device
has been
'                     | connected
' DeviceParam2V       | Not used
' ErrorP              | Variable for returning an error message.
'--------------------------------------------------------------------
Sub SFD_Init( sgCOMName, DeviceParam2V, ErrorP )
  Dim     lErrorStateT,sgReplyDataT
```

```
  ' Creating the UDI object
  Set oUDIM=CreateObject("DIAdem.SFD.UDI" )
  '-------------------------------------------------
  ' Opening and initializing the serial
  ' port
  '-------------------------------------------------
  Call SFDU COMInit(oUDIM,sgCOMName+",9600,N,8,1")
  '-------------------------------------------------
  ' Initializing the device
  '-------------------------------------------------
  CALL AK_Query(oUDIM,"STBY",0,"",true,lErrorStateT,sgReplyDataT)
  CALL AK_Query(oUDIM,"SINT",0,"",true,lErrorStateT,sgReplyDataT)
End Sub

'--------------------------------------------------------------------
' SFD_DeInit
' Purpose             :   Carry out de-initialization, delete UDI
objects etc.
'--------------------------------------------------------------------
' ErrorP              | Variable for returning an error message.
'--------------------------------------------------------------------
Sub SFD_DeInit(ErrorP )
  ' Close communication channel
  oUDIM.Close()
  ' Delete UDI object
  Set oUDIM = Nothing
End Sub
```

7 The Device Simulator

7.1 General

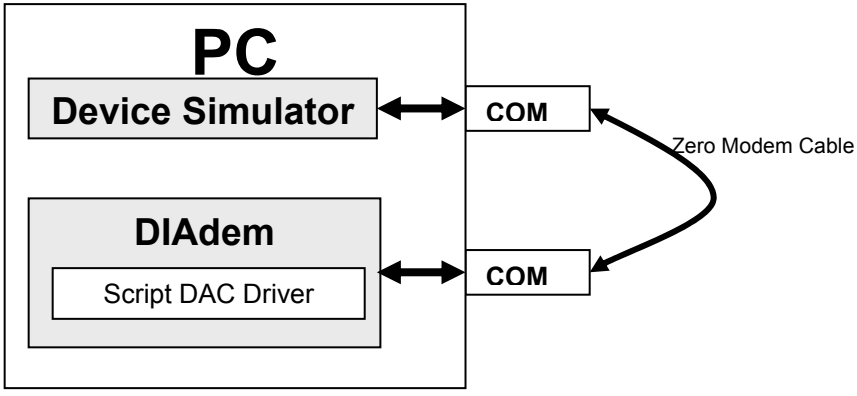
The device simulator is an application to simulate a measurement device that communicates with a connected computer via a serial port.

What is involved here is not the actual functions of the device, but rather, providing an application in which the induction to the Script DAC Driver can take place.

The device simulator simulates a device with 10 channels, in which every channel can be parameterized individually. The simulator "understands" a set of commands using which it can be parameterized and data can be invoked. In addition to explicitly calling the data, the simulator can be so set that data is transmitted permanently. This mode of the simulator is called the "free-running mode" in the text that follows.

In order to avoid faulty operation of the device simulator as much as possible and to keep the system requirements low, communication with the device simulator is done over the serial ports COM1 or COM2 with a fixed parameterization of the ports.

The following diagram shows the Script DAC Driver, DIAdem and the communication between the two applications over two serial ports:



7.2 System requirements

Operating system: Windows 95,98,NT

Hardware: Two free serial ports COM1 or COM2, Zero modem cable for connecting the serial ports.

7.3 Commissioning the device simulator

On starting, the device simulator tries to open the serial port COM1. If this is not successful, the program (after outputting a corresponding message) tries to open the serial port COM2. If that works, a message it output that the device simulator is using the serial port COM2 for communication. If this is not successful either, an error message is output and the program ends.

For operating the device simulator, the serial port that the simulator has selected for communication is connected with the port that the Script DAC Driver should use in DIAdem for reading data of the simulator. In most cases, this means connecting the first and second serial interface. This connection must be done with a so-called "zero-modem" cable.

The settings of the serial port of the device simulator are:
- 9600 Baud
- no parity
- 8 data bits
- 1 stop bit

The simulator automatically makes these settings at the interface that it has chosen for communication. The port that is addressed from DIAdem must be configured by the Script DAC Driver.

7.4 Operation

The following dialog is displayed after the start of the device simulator.

All the settings that are made manually are made from this dialog.

In the upper part of the dialog ("Channels"), there is a list of all the available channels and the relevant parameters. The number of the

relevant channel is displayed in the first column. This is the number using which the individual can be addressed from DIAdem.

The channels can be individually activated using the checkboxes in the second column ("Active"). Only the activated channels are considered later at the time of a measurement. Just as is the case for adjusting a measurement recorder to a real measurement device, in the next column, the signal form of the simulated measurement signal can be selected. The signal forms "Sine", "Rectangular" and "Triangle" are available.

The signal amplitude in the fourth column ("Amplitude") of the signal list corresponds to the measurement range of the input on a real device.

Using the input fields in the fifth and sixth column, a unit and the frequency of the simulated signal can be assigned to every signal.

The polling rate and the output format can be set in the next section of the dialog ("Global parameters"). The polling rate that has been set is evaluated by the program only if the device simulator has to output data in free mode.

The parameter "Output format" is used to define in which format the data is to be output via the serial port. The possible output formats are described in a separate section.

In the lower section ("Status"), some more status information on the device is displayed:

The checkbox "Remote Control" indicates that the device simulator is in the remote control state. This checkbox is activated as soon as a signal is sent to the simulator. At the end of communication with the device simulator, this checkbox can be unchecked either by means of a command over the serial port or manually.

The checkbox "Running" indicates whether the device simulator is in free running mode. In this mode, the simulator independently transmits data with the polling rate that has been set.

In the line "Last command", the last command that has been received by the serial port is displayed.

All the parameters of the device simulator can be changed both manually as well as by means of a command that is sent via the serial port. If the simulator is in "Remote control" state, the input of some parameters is blocked. Thus, e.g. it is not possible to change the output format if the device simulator is in remote control state.

7.5 Function test of the simulator

After the simulator has been made operational, it is very easy to test whether the connection between the computer and the device simulator is working. If a few characters are sent to the simulator, the device simulator must go into remote control mode (the checkbox "Remote Control" is then checked). After the device simulator has then been brought back into operational state by clicking the checkbox "Remote Control", you can start the data output of the device simulator by clicking the checkbox "Run". It should now be possible to receive data with DIAdem. The data output is ended again by clicking the checkbox "Run" again.

7.6 Output formats

To be able to match the data output as accurately as possible with the output formats of existing devices, the device simulator provides a number of different formats. In addition to the data output in different ASCII formats, there are various binary output formats available , which are described individually here.

ASCII

In the case of data transmission in output format "ASCII", the measurement data of the individual channels is transmitted in plain text. All the measurement data that belongs to one measurement cycle (Scan) is transmitted in one line. The individual values are delimited from each other by semicolons (;) and the line is terminated by <CR><LF>. In this output format, the measurement data is output in scaled form.

Syntax <measurement value>[;<measurement value>]<CR><LF>

Example for 3 channels:

```
1.2345;3.1415;2.1478<CR><LF>
```

ASCII with Status

This output format essentially corresponds to the output format ASCII. However, the number of the relevant channel is transmitted before every measurement value, as additional data.

Syntax <Channel number>;<measurement value>[;<Channel number>;<measurement value>]<CR><LF>

Example for channels 2, 4 and 7

```
2;2.3456;4;5.2837;7;10.0000<CR><LF>
```

Binary, 1 Byte

In this output format, the measurement data is transferred as one byte per measurement value. Here, the measurement range (amplitude) of the signal is mapped to the range -128 to 127. In order to determine the respective measurement value from the measurement data to be transferred, the measurement data should be calculated with the measurement range (amplitude). The individual measurement values are transmitted without a delimiter.

Syntax <measurement value>[<measurement value>]

Example for 3 measurement values (as hex dump)

```
31 22 55
```

This corresponds to the following measurement values: 49, 34, 85 (unscaled).

Binary, 1 byte with status

This format, in its construction, corresponds to the output format Binary 1 byte. However, another byte is transmitted before every measurement value, which contains the channel number of the relevant measurement value.

Syntax <Channel number><measurement value>[<Channel number><measurement value>]

Example for the query on channels 2, 4 and 7 (as Hex dump):

```
02 31 04 22 07 55
```

This corresponds to the following measurement data: 49 (channel 2) , 34 (channel 4) , 85 (channel 7)

Binary, 2 byte MSB-LSB

In this output format, 2 bytes are transmitted per measurement value. Here, the measurement range (amplitude) of the signal is mapped to the range -32768 to 32767. Every measurement value is represented here by two bytes, which are transmitted in the sequence high-byte, low-byte. In order to determine the respective measurement value from the measurement data transferred, the measurement data should be calculated with the measurement range (amplitude).

Syntax <measurement value>[<measurement value>]

Example for 3 measurement values (as hex dump)

```
FCB3 19B2 1267
```

This corresponds to the following measurement values: -845, 6578, 4711

Binary, 2 byte MSB-LSB with status

This format, in its construction, corresponds to the output format "Binary" 2 byte MSB-LSB. However, another byte is transmitted before every measurement value, which contains the channel number of the relevant measurement channel.

Syntax <Channel number><measurement value>[<Channel number><measurement value>]

Example for the query on channels 2, 4 and 7 (as Hex dump):

```
02 FCB3 04 19B2 07 1267
```

This corresponds to the following measurement values: -845 (channel 2) , 6578 (channel 4) , 4711 (channel 7)

Binary, 2 byte LSB-MSB

This format, in its construction, corresponds to the output format "Binary" 2 byte MSB-LSB. Only the sequence in which the

measurement data is transmitted is replaced. Here, the data is transmitted in the sequence Low-Byte, High-Byte.

Syntax <measurement value>[<measurement value>]

Example for 3 measurement values (as hex dump)

```
B3FC B219 6712
```

This corresponds to the following measurement values: -845, 6578, 4711

Binary, 2 byte LSB-MSB with status

This format, in its construction, corresponds to the format 2 byte MSB-LSB with status. However, another byte is transmitted before every measurement value, which contains the channel number of the relevant measurement channel.

Syntax <channel number><measurement value>[<channel number><measurement value>]

Example for the channels 2, 4 and 7 (as Hex dump):

```
02 B3FC 04 B219 07 6712
```

This corresponds to the following measurement values: -845 (Channel 2), 6578 (Channel 4), 4711 (Channel 7)

Binary, 8 byte MSB-LSB

In the case of the output format "Binary 8 byte MSB-LSB", the measurement values are transmitted as 8-byte long data blocks in data format "double". The byte-sequence of the transmission is high-byte low-byte.

Syntax <measurement value>[<measurement value>]

Binary, 8 byte MSB-LSB with status

With this format, before each measurement value, an additional byte is also transmitted, in which the channel number is present. The rest of the construction corresponds to the output format "binary 8 byte MSB-LSB".

Syntax <Channel number><measurement value>[<Channel number><measurement value>]

Binary, 8 byte LSB-MSB

This format corresponds to the output format binary 8-byte MSB-LSB; however, here, the transmission sequence is low-byte, high-byte.

Syntax <measurement value>[<measurement value>]

Binary, 8 byte LSB-MSB with status

This format corresponds to the output format "Binary 8-byte with status MSB-LSB; however, here, the transmission sequence is low-byte, high-byte.

Syntax <Channel number><measurement value>[<Channel number><measurement value>]

7.7 Command set

The command set of the device simulator consists of 3-character long combinations of letters which, if required, are supplemented with parameters All the commands are specified in upper case only.

A command is always terminated by the <LF> character (Line Feed). The combination of characters <CR><LF> is also permissible.

All commands always return a status or the requested data.

The status is either the character "0" (character code 48) or the character "?". (Question mark). The character "0" represents successful completion of a command, whereas the character "?" is returned in case of an error.

For better readability of a command, any number of spaces or TABs can be included in the command. These characters serve only to improve the readability and are ignored by the device simulator.

A list of commands that are understood by the device simulator now follows.

The ACH command

Meaning Activation of a measurement channel.

Syntax ACH <Channel number>,<Status><LF>

Parameters

Channel number Number of the measurement channel (0-9)

Status Activation status (0=not active, 1=active)

Return "0“=OK "?“=Error

Example:

```
ACH 1, 1<LF> (Switches channel 1 to active)
ACH 2, 0<LF> (Switches channel 2 to
inactive)
```

The ACH? command

Meaning Querying the channel status

Syntax ACH?<Channel number><LF>

Parameters

Channel number Number of the channel whose activation status has to be determined (0 - 9)

Return "0“=Channel is not active "1“ =Channel is active " ?“= Error

Example:

```
ACH?1<LF> // Querying the activation status
                    of channel 1
```

The AMP command

Meaning Setting the amplitude (measurement range) of a channel.

Syntax AMP<Channel number><Amplitude><LF>

Parameters:

Channel number Number of the channel whose amplitude has to be set (0 - 9)

Amplitude Amplitude to be set (0.1 - 10.0)

Return "0“=Command executed successfully "?“=Error

Example:

```
AMP 3, 7.5<LF> // Setting the amplitude of
                    channel 3 to 7.5
```

The AMP? command

Meaning Querying the amplitude (measurement range) of a channel (0 - 9).

Syntax AMP?<Channel number><LF>

Parameters:

Channel number Number of the channel whose amplitude has to be queried (0 - 9)

Return Amplitude value "?“=Error

Example:

```
AMP? 5<LF> // Querying the amplitude of
                    channel5
```

The COF command

Meaning Setting the output format. This command defines the output format in which the measurement data is transmitted over the serial interface.

Syntax COF<Output format<LF>

Parameters Output format: Format in which the measurement data should be output. The following inputs are permitted:
„0“ : ASCII
„1“ : ASCII with Status
"2“ : Binary 1 Byte
"3“ : Binary 1 byte with status
"4“ : Binary 2 byte MSB-LSB
"5“ : Binary 2 byte MSB-LSB with status
"6“ : Binary 2 byte LSB-MSB
"7“ : Binary 2 byte LSB-MSB with status
"8“ : Binary 8 byte MSB-LSB
"9“ : Binary 8 byte MSB-LSB with status
"10“ : Binary 8 byte LSB-MSB
"11“ : Binary, 8 byte LSB-MSB with status

Return "0“=Command executed successfully, "?“=Error

Example:

(Setting the output format to "binary 1 byte with status")

```
COF 3<LF>
```

The COF? command

Meaning Querying the current output format The command COF? can be used to query the output format that is currently set. The list of output formats that is available and their coding can be taken from the description of the command "COF".

Syntax COF?<LF>

Parameters None

Return As the return value, this function returns the code of the currently set output format. A "?" is returned in case of an error.

The DCL command

Meaning Ending the remote control state

Syntax DCL<LF>

Parameters none

Return none

The ENU command

Meaning Setting the measurement quantity.

Syntax ENU<Channel number><Unit><LF>

Parameters:

Channel number Number of the channel that is to be set (0 - 9)

Unit Character string that contains the designation of the measurement quantity.

Return "0“=Command executed successfully "?“=Error

Example:

(The unit of channel 7 is set to "Volt".)

```
ENU 7,Volt<LF>
```

The ENU? command

Meaning Querying the unit of a channel

Syntax ENU?<Channel number><LF>

Parameters:

Channel number Number of the channel that is to be set (0 - 9)

Return Character set that contains the current unit of the channel.

Example:

(Querying the unit of channel 5)

```
ENU? 5<LF>
```

The EST? command

Meaning Querying the error status

Syntax EST?<LF>

Parameters none

Return

"0“ = Command executed successfully
"1“ = Syntax error
"2“ = invalid channel
"3“ = too few parameters
"4“ = erroneous parameters

Example:

(Querying the error status)

```
EST?<LF>
```

The FRE command

Meaning Setting the signal frequency

Syntax: FRE<Channel number><Frequency><LF>

Parameters:

Channel number Number of the channel whose frequency has to be set (0 - 9)

Frequency Signal frequency to be set (0.1 - 10.0)

Return "0“=Command executed successfully, "?“=Error

Example:

```
FRE 1, 7.4<LF> // Setting the signal
                  frequency of channel 1 to
                  7.4 Hz
```

The FRE? command

Meaning Querying the signal frequency

Syntax: FRE?<Channel number><LF>

Parameters Channel number: Number of the channel whose frequency is to be queried (0 - 9)

Return Frequency," ?“ = error

Example:

```
FRE? 1<LF>
```

The IDN? command

Meaning Querying the device name

Syntax IDN?<LF>

Parameters none

Return "device simulator", "?“=Error

The ICR command

Meaning Setting the polling rate for the free running mode

Syntax ICR<polling rate><LF>

Parameters:

Polling rate Polling rate to be set for the free-running mode (0.1 - 50.0)

Return "0“=Command executed successfully, "?“=Error

The ICR? command

Meaning Querying the polling rate for the free running mode

Syntax ICR?<LF>

Parameters none

Return current polling rate, "?“ = error

The MSV? command

Meaning Querying the current measurement value of a channel

Syntax MSV?<Channel number>

Parameters:

Channel number Number of the channel that is to be queried (0 - 9)

Return Measurement value, "?“=Error

Example:

(Querying the current measurement value of channel 5)

```
MSV? 5
```

The RUN command

Meaning Starting the free-running mode

Syntax RUN<LF>

Parameters none

Return Measurement data “?“ = error

The STP command

Meaning Stopping the free-running mode

Syntax STP<LF>

Parameters None

Return "0“=Command executed successfully, "?“=Error

The TRG command

Meaning Triggering a scan

Syntax TRG<LF>

Parameters None

Return Measurement data of all active channels "?“=Error

The WAV command

Meaning Setting the signal form of a channel

Syntax WAV<Channel number><Signal form><LF>

Parameters:

Channel number Number of the channel that is to be set (0 - 9)

Signal form

"0“ = Sine

"1“ = Rectangular

"2“ = Triangular

Return "0“=Command executed successfully, "?“=Error

The WAV? command

Meaning Querying the signal form of a channel

Syntax WAV?<Channel number><LF>

Parameters:

Channel number Number of the channel that is to be set (0 - 9)

Return

"0“ = Sine

"1“ = Rectangular

“2“ = Triangular

"?“ = error

8 Other sources

The short chapters that follow have pointers to further, more detailed information that will help you to construct your script for the Script DAC Driver more quickly and effectively.

Documentation for VBScript

More detailed documentation on VBScript can be found at the Website „http://msdn.microsoft.com/scripting“. On this page, select the point "VBScript" and then the sub-point "Documentation".

Windows-Scripting Host

Information on Windows Scripting Host can be found at the Website "http://msdn.microsoft.com/scripting“. On this page, select the option "Windows Script Host".

Script Debugger

Information on Script Debugger can be found at the Website "http://msdn.microsoft.com/scripting“. On this page, select the option "Script Debugger".

9 List of changes

Version 2.1 :

Chapter	Description
2.1	Correction of the formatting of the parameter description

Version 2.2 :

Chapter	Description
5.4.1	New example on the function generator HP 33120A "Initialization of the function generator HP 33120A"
5.4.2	New example on the function generator HP 33120A "Online parameterization of the function generator HP 33120A"

Version 2.3 :

Chapter	Description
5.4.3	New example of the GPS receiver "Reading a GPS receiver in Motorola binary format"
5.4.6	New example for fuel weighing scale AVL 733S "Fuel weighing scale 733 S from AVL"
4.2	The chapter "Access to DIAdem variables“ was enhanced.

Version 2.4 :

Chapter	Description
8.1.1	Description of the attribute "Version"
8.1.2	Description of the attribute "IsOpen"