Inhaltsverzeichnis

Extended Script Functions

To establish communication between the integrated script-interpreter and the surrounding main program, the script-language has been extended with additional procedures:

Please note that some procedures are only available in some versions of SKDS.

V1	V2	V3	V4	Procedure
✔	✔	✔	✔	Write(VAR text: String)
✔	✔	✔/[1]	✔	Telegramm(VAR telegram: string; VAR Error: integer)
✔	✔	[1]	✔	Diagnose(OnOff:Boolean; VAR Success: Boolean)
✔	✔	✔	✔	ChangeID(newTitle: String; newID: integer; VAR Success: BOOLEAN)
✔	✔	✔	✔	Readstring(Windowtitle: String; VAR Answer: String)
	✔	[1]	[1]	Download(VAR telegram: String; VAR Error: integer; VAR filename: String ; Address: LONGINT)
	✔	[1]	[1]	Upload (VAR telegram: String; VAR Error: integer; VAR filename: String ; Address: LONGINT)
	✔	[1]	[1]	WriteDatabyLocalID(VAR telegram: String; VAR Error: integer; VAR filename: String ; ID: byte)
	✔	[1]	[1]	ReadDatabyLocalID (VAR telegram: String; VAR Error: integer; VAR filename: String ; ID: byte)
	✔	✔	-	KeyMatchTo(key:string; VAR Success: BOOLEAN)
	✔	✔	✔	md5sum(filename: String; VAR md5: String)
		✔	✔	GetErrorRate(VAR err: byte)
		✔	✔	SendUDS(VAR err: integer)
		✔	✔	GetDTC(HighPid: byte; LowPid: byte; VAR Hex: String; VAR Display: String; VAR Description: String; VAR Obsolete: String)
		✔	✔	GetConfig(system: byte; value: byte; VAR systemString: String; VAR valueString: String; VAR system: String; VAR Description: String)
		✔	✔	UDSGetCall(pos: longint; VAR Value: byte)
		✔	✔	UDSSet(position: longint; Value: byte)
		✔	✔	UDSClear()
		✔	✔	UDSlen(VAR len: longint)
		✔	✔	UDSEditConfig(VAR start: longint; ReadOnly: BOOLEAN)
		✔	✔	loadbuffer(start: longint; VAR filelen: longint; VAR filename: String; VAR err: integer)
		✔	✔	savebuffer(start: longint; VAR filelen: longint; VAR filename: String; VAR err: integer)
		✔	✔	SetBuffer(bufferNr: byte, newSize: longint)
		✔	✔	CopyBuffer(bufferNr: byte)
		✔	✔	BlitBuffer(frombuffer: byte; startpos: longint; topos: longint; blocklen: longint)
		✔	✔	FileDialog(title:String; FileMustExist: Boolean; VAR filename: String)

[1] replaced by the UDS-commands

Data Transfer (Old Single Frame Handling)

Telegram

```
Telegram(VAR telegram: String; VAR error: Integer)
```

This extended Script Function sends a single KWP2000-Telegram to the module and reports the response telegram back.

Unfortunately the Script-to-program interface does not allow Byte-Arrays to be given directly to an external procedure, so the following workaround has to be made with each telegram:

- a telegram consists in the sample-scripts of t : Array[0..9] of byte
- t[0] describes the length of the telegram in bytes
- t[1] .. t[n] contains the data bytes of the telegram itself

Important: The telegram contains only the data content of the telegram! The Length/Physical Address Byte, Tester address and ECU-Address at the beginning and also the checksum at the end is automatically added and handled by SKDS itself!

In the CAN-Versions the Byte t[9] is used to set and read the PCM-Byte.

If t[9]=0, then the PCM-Byte is automatically created as a single frame message, containing the length of the message.

But if t[9]<> 0, then the value of t[9] is taken as a PCM-Byte, so the PCM-Byte can be directly set through t[9].

The received telegram is also stored in the t[0..9] Byte array, where the received PCM-Byte is stored in t[0], the received data bytes are in t[1..8]

Please note that this functionality is replaced by the UDS-Commands in Version 3.

Starting with Version 3, SKDS does not need to handle long data packets as single telegrams anymore. These packets are now just „one big telegram“ with a length of up to 4095 Bytes.

In contrast to previous versions, where the whole telegram content was handled in the script itself (as that t[]- Array), now the telegram is kept internally in SKDS itself, and the scripts only offer functions to create, transfer and modify this telegram buffer.

A typical communication flow would be like this:

Sample

```
UDSclear;
UDSSet(0,$19)
UDSSet(1,$02)
UDSSet(2,$08)


sendUDS(err)
UDSlen(len)
```

```
if  err > -1  then
    if ( UDSGet(0) = $59 ) then
    ....
    endif
else
    switch err of
        case 0 : res := 'Bustransfer ok';endcase
        case -1 : res := 'Bus error';endcase
        case -2 : res := 'No Answer from Module';endcase
        case -4 : res := 'File not found';endcase
        case -5 : res := 'File too big';endcase
        case -6 : res := 'Couldn not open file';endcase
        case -7 : res := 'Unexpected Answer';endcase
        else
            res := 'Unknown Busserror'
        endcase
    endswitch
endif
```

SendUDS

```
SendUDS(VAR err: integer)
```

Send the actual telegram buffer to the module, already splitted into multiframes, if necessary.

After the transmission SendUDS reports an error code back in „err“. If no error has occured, the answer of the module is contained in the SKDS telegram buffer.

UDSGetCall

```
UDSGetCall(pos: longint; VAR  Value: byte)
```

Read the value 'Value' at position 'Pos' out of the telegram buffer

As a comfort function, the UDSGetCall-function is normally not called directly, it is wrapped by the UDSGet-function (as part of the standard-UDS.pas - library), which returns the wanted value as a function call.

UDSSet

```
UDSSet(position: longint; Value: byte)
```

Writes the value 'Value' at position 'Pos' into the telegram buffer. If necessary, the length value of the buffer is automatically adjusted.

UDSClear

```
UDSClear()
```

Fills the complete telegram buffer with 0 and sets the length to 0

UDSlen

```
UDSlen(VAR  len: longint)
```

returns the actual length of the telegram in „len“

Tester Present Messages

Diagnose

```
Diagnose(OnOff:Boolean; VAR Success: Boolean)
```

This extended Script Function sends in a 3-sec interval a regular „Request-Parameter-by-PID“ - Telegram to the module to keep the link alive.

This function is only needed in SKDS-Versions <3. From Version 3 onwards SKDS listens to the data transfer and recognizes itself, if a module reports an operational state and switches the tester present message accordingly.

It also distinguishes, whether the request was done through an old KWP- or a new UDS-command and sends the tester present accordingly with the module address or as broadcast.

Hint: If a module is controlled in a script by a mix of old KWP and new UDS commands, please keep in mind that the type of the last send telegram determinates the type of the following tester present messages. This might cause the effect that after using an UDS- command for an old KWP module the tester present message seems not to work anymore (because it was switched from physical to broadcast address). In such cases, just add an „empty“ KWP tester present message after your UDS sequence. This will make SKDS to switch back to KWP tester presents after the normal command execution.

The variable OnOff defines whether the Interval-Timer has to be switched on or off (On= TRUE)

The variable Success gave the feedback whether the request was successful or not (Successful=TRUE), but it is not used any more. For compatibility reasons it is always TRUE.

The module answer is shown in the top left column of the status bar:

- ?? - the module has not reported its state yet
- $xx Diagnostic - The Module is currently in State $xx

Logging and Text Input

Write

```
Write(VAR text: String)
```

This function writes „text“ into the Log Window.

ReadString

```
ReadString(title: String; VAR answer: String)
```

This extended Script-Function is used to request inputs from the user.

By calling this function, a window will be opened, in which the „title“ is shown as window title, and the variable „answer“ is shown in an edit-cell, where the user can modify this value of „answer“ up to a total length of 250 chars.

Module Addressing

ChangeID

```
ChangeID(newTitle: String; newID: integer; VAR Success: BOOLEAN)
```

This extended Script-Function is used to set a new Module-ID for the actual script. The „newTitle“ gives the new caption of the window, the „new“ID„ gives the new Module- ID. The value of „Success“ tells whether the ID has been changed or not.

File Input/output

Download

```
Download(VAR telegram: String; VAR Error: integer; VAR filename: String ;
Address: LONGINT)
```

and

Upload

```
Upload (VAR telegram: String; VAR Error: integer; VAR filename: String ;
Address: LONGINT)
```

These extended Script-Functions start the RequestDownload-Mode or the RequestUpload-Mode.

The file name identifies the file which will be transferred to or from the ECU. If the filename is "„ (empty), then a file selection window will be opened, from which a file can be selected. The selected file name will be returned in „filename".

The functions try to run through the complete up/download protocol. Whenever this sequence is stopped by reaching the end or through an error case, the last received telegram from the ECU is returned as „telegram". As in the telegramm()-function the telegram is returned as coded string, so the answer needs to be converted to its original values in the hyperterp-script first.

In the variable „err" further information about the progress will be returned.

The address (range 0-65535) identifies the memory address inside the ECU, from/to where the data should be transferred.

The size of a block transfer is limited by the protocol to 4095 Bytes.

Please note that these functions are replaced by the UDS-Commands in Version 3.

WriteDatabyLocalID

```
WriteDatabyLocalID(VAR telegram: String; VAR Error: integer; VAR filename:
String ; ID: byte)
```

and

ReadDatabyLocalID

```
 ReadDatabyLocalID (VAR telegram: String; VAR Error: integer; VAR filename:
String ; ID: byte)
```

Both these functions are simular to the Up/Download function above, but instead of an address the memory to load from/to is identified by an ID- No.(0-255). Also the block transfer size is limited to 50 Bytes.

Please note that these functions are replaced by the UDS-Commands in Version 3.

In Version 3, the different block transfer commands have been exchanged against a general procedure.

Generic Filetransfer in Version 3

Since Version3 SKDS supports a generic file transfer scheme. A typical file transfer would be like this:

Sample

```
procedure filetest;

var
    filename: string
    start: longint
    len: longint
    errtext: string
    lentext: string
endvar;
UDSclear;
UDSSet(0,$19)
UDSSet(1,$02)
UDSSet(2,$08)
len:=0;
filename:= '';
loadBuffer(3, len,  filename,err);
str(len:0:0,lentext);
str(err:0:0,errtext);
write('load:  Filename ' + filename + ' len '+ lentext + ' err  '+ errtext)
sendUDS(err)
filename:=filename + "_o"
saveBuffer(3, len,  filename,err);
str(len:0:0,lentext);
str(err:0:0,errtext);
write('save:  Filename ' + filename + ' len '+ lentext + ' err  '+ errtext)

endproc
```

loadbuffer

```
loadbuffer(start: longint; VAR filelen: longint;  VAR filename: String; VAR
err: integer)
```

Reads the file „filename“ into the telegram buffer starting at position „start“ by reading „filelen“ number of bytes

The number of bytes read will be returned in „len“.

The following conditions apply:

- If the filename is “„ (empty), then a file selection window will be opened, from which a file can be selected. The selected filename will be returned in „filename“.
- If len is 0, the whole file will be read. If the length exceeds the telegram length, an error will be raised.

savebuffer

```
savebuffer(start: longint; VAR filelen: longint;  VAR filename: String; VAR
```

```
err: integer)
```

Writes the telegram buffer starting at position „start“ by reading „filelen“ number of bytes into the file „filename“.

The number of bytes written will be returned in „len“.

The following conditions apply:

- If the filename is “„ (empty), then a file selection window will be opened, from which a file can be selected. The selected file name will be returned in „filename“.
- If len is 0, the whole telegram buffer will be written.

setbuffer

```
setBuffer(bufferNr : byte; newSize: longint)
```

Changes the actual buffer used to buffer number „buffernr“. SKDS supports 10 buffers, counted from 0 to 9. The startup buffer is nr. 0.

If newsize is <> 0, the old buffer is deleted and new memory with size newmem is allocated. The maximal memsize is 2^31 = 2147483648 bytes

copyBuffer

```
copyBuffer(bufferNr : byte)
```

Copies the content of buffer nr. „bufferNr“ into the actual buffer.

BlitBuffer

```
BlitBuffer(frombuffer: byte; startpos: longint; topos: longint; blocklen:
longint )
```

Copies a memory block from the buffer „frombuffer“ starting at position „startpos“ to the actual buffer to position „topos“ with the length of „blocklen“ bytes.

In case the buffer len needs to be bigger, the buffer len is increased accourdingly.

SetBufferLen

```
SetBufferLen(VAR newSize: longint)
```

All other buffer commands can increase the 'len' of a buffer, but none of them can make a buffer shorter, except SetBufferLen.

SetBufferLen sets the 'len' of the current Buffer to 'newSize'.

The success of the SetBufferLen - operation is returned in the variable newSize itself as follows:

input value of newSize	return value of newSize
< = 0	available size of the buffer in bytes. This can be used to read the real allocated memory size of that buffer
1.. available size	new available size (= requested size)
> available size	available size of the buffer as **negative value** . This is a fault condition

As seen, a negative value given back indicates a fault condition, all other returned values are positive.

Database

Get DTC

```
Get DTC(HighPid: byte; LowPid: byte; VAR Hex: String; VAR Display: String;
VAR Description: String; VAR Obsolete: String )
```

Since Version 3 SKDS has a database for DTCs tosupport OEM data. To read DTC-informations, call up the Get DTC-function with the bytes for HighPid and LowPid.

The information feed backs are:

- Hex: Contains the given DTC in hexadecimal writing
- Display: The DTC as OEM-Number
- Description: Textual description of this DTC
- Obsolete: Set, if this DTC is already set to obsolete in the database

Get Config

```
GetConfig(system: byte; value: byte; VAR systemString: String; VAR
valueString: String; VAR system: String; VAR Description: String )
```

Since Version 3 SKDS has a database to load OEM vehicle config. To read Config-informations, call up the Get Config-function with the bytes for the parameter number (system) and its value.

The information feed backs are:

- systemString: Contains the given systemvalue in hexadecimal writing
- valueString: Contains the given value in hexadecimal writing
- system: Name of the system given by the system-value
- Description: Textual Description of the given value

Miscellaneous

Key Match To

```
Key Match To(key:string; VAR Success: BOOLEAN)
```

This function allows the user to check, if the actually used dongle matches a particual User/Company combination.

The user/company will be defined as described under Key-Management and has been given to the function in „key“. If the actual key matches the given definition, „Success“ returns to TRUE, „No success“ returns to FALSE. This function allows the user to make an execution of script parts, depending on the actual script user.

Since Version 4 this function is not supported anymore and returns always FALSE.

md5sum

```
md5sum(filename: String; VAR md5: String)
```

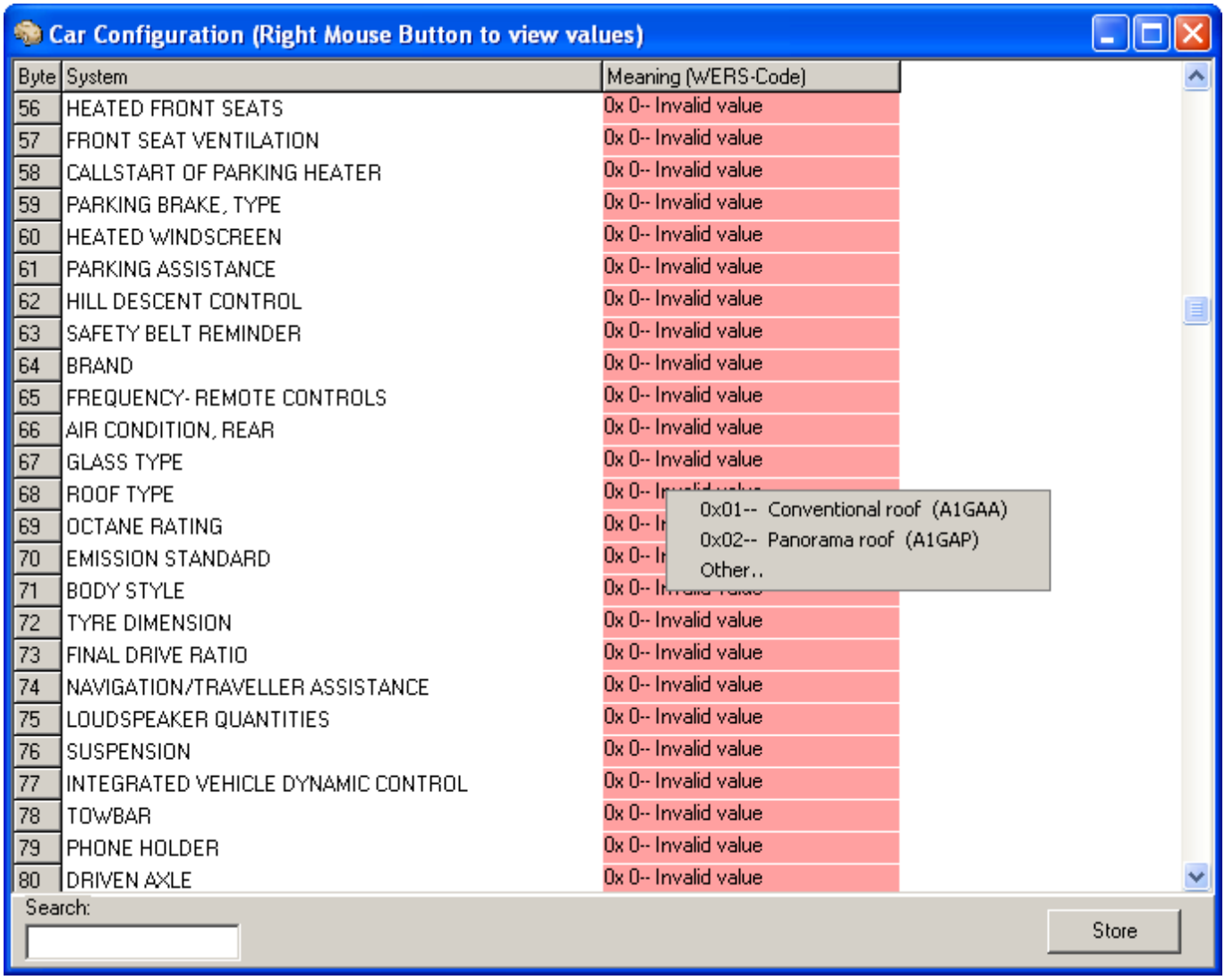
This function allows the verification and comparison of files such as parameter files via their md5-checksum.

If the given file „filename“ could not be read or the file has the length 0, the returned md5-value „md5“ is equal to „d41d8cd98f00b204e9800998ecf8427e“, otherwise the real file checksum returns.

Vehicle Config

UDSEditConfig(VAR start: longint; ReadOnly: BOOLEAN)

This function opens a new window, in which the content of the buffer is shown as Vehicle Configuration, based on the data taken from the Vehicle Config-Database, starting at the buffering position „start“.



If ReadOnly is TRUE, the values can only be read, but not been changed and saved.

FileDialog

```
FileDialog(title:String; FileMustExist: Boolean; VAR filename: String)
```

Opens a file dialog to select a filename

The parameters are

- title: Title of the filedialog- Window
- FilemustExist: If TRUE, only existing files can be selected
- filename: If <>, directory and filename are preselected. Contains full path of a file after selecting a file, or if operation is cancelled.

GetErrorRate

```
GetErrorRate(VAR err: byte)
```

Reports the actual Error Frame Rate back to the Script (in percent 0-100)

Errorcodes

These codes are given back in the var „err“ after any telegram transmission:

- 0 No Error
- -1 Bus-Error or no Bus-driver connected
- -2 No Answer from Module
- -4 File not found
- -5 File too big
- -6 Couldn't open file
- -7 Unexpected Answer