Inhaltsverzeichnis

Comfort functions (The Standard-UDS Library)

As shown in the extended functions, SKDS has some built-in functions to enable the basic communication to the modules.

But to make it more comfortable, the OEM data [1)] of SKDS includes a library, which supports several helpful functions.

This library is made as a collection of Hyperterp functions. To use these functions in your own scripts, just include the standard-UDS.pas in the beginning of your own scripts by using the skdsc-directive

```
(*#include <standard-UDS.pas>*)
```

and use the skdsc -I command line parameter to define where the include-file can be found:

```
skdsc -I \path_to_includes\standard-UDS.pas file_to_compile.pas
```

User Functions

By including the library, the included „end-user“ functions automatically appear in the function table.

How to transfer module specs into SKDS

Although the standard library already contains a lot of functions to handle the common data of modules, there are tons of module specific values left, which need to be implemented per each module.

To assist this work, the standard library contains some functions which act as a kind of generic interface between a Module specification and SKDS:

```
procedure getNUMPID( hpid: byte ; lpid: Byte ; dlen: Byte ; resolution :
real ; offset : real ; valunit : string )
procedure getPIDBMP( hpid : byte ; lpid : Byte ; bytenr : integer ; bytepos
: Byte ; negstring : String ; posstring : string)
procedure getPIDASC( hpid : byte ; lpid : Byte ; nrofchar : Byte )
procedure setPIDBMP( hpid : byte ; lpid : Byte ; bytenr : integer ; bytepos
: Byte ; negstring : String ; posstring : string)
```

You might notice that the parameters of these functions are equivalent to the type of values given in the module spec. The trick now is to use some Excel formulas to reformat the module spec into Hyperterp function calls.

This can be archived in the following way:

- Make sure you have the module spec available as Excel file
- Download the SKDS-Module_Converter-Sample.xls
- Have a close look at the function of these formulas. Be aware that there are different formulas

for different PID types
- Copy and adapt the formulas shown in the Converter-Sample to your module spec.
    - Make sure that the cell references in the formulas point to the correct columns with the correct data type
    - Crosscheck twice that the writing and formating of the referenced cells is correct (Are Bits are numbered from 0-7? Is the format of hexadecimal numbers correct (0xFF instead of $FF?)? Are all cells filled out and not just blank? Do text fields contain illegal special characters like „ ?)
- Copy the adapted formulas into each affected row
- After having done all that, the last column of the formula area shows the raw Hyperterp program text, but without proper LineFeeds. To archive a correct text formating aswell, do
    - Copy the raw program text into a text editor, which is able to replace special LineFeed characters (e.g. MS-Word)
    - Search and replace the placeholder {n} against LineFeeds (in Word that would mean replace “{n}„ against „^p“)
    - Now, finally copy the resulting text into your SKDS script file

After compiling the script and loading it into SKDS, do a quick run (with the „Update“- Button) through all generated functions. It might quite often happen, that an overseen special character, a blank cell or a wrong number format causes a Hyperterp runtime syntax error. Correct the error in the Excel file and run through the steps above again.

A common problem with the spec files are that not each single row is filled with the complete parameter set. The following autoFill- macro fills the gaps. Assuming you've a table content like this

PID	Byte	Bit
0815	1	0
		1
		2
		3
	2	1
		2
		3
		4
4711	1	0
		1
		2
		3
	2	1
		2
		3
		4

just select that area and let the macro run. The result will be

PID	Byte	Bit
0815	1	0
0815	1	1
0815	1	2

PID	Byte	Bit
0815	1	3
0815	2	1
0815	2	2
0815	2	3
0815	2	4
4711	1	0
4711	1	1
4711	1	2
4711	1	3
4711	2	1
4711	2	2
4711	2	3
4711	2	4

The macro itself:

```
Sub autofill_Part2_Parameter_fiels()
'
' autofill_Part2_Parameter_fiels Macro
' This macro autofills the gaps in a part2 spec
'

Dim Arr() As Variant
Dim area As Range
Dim C As Range
Dim sizeX As Long
Dim sizeY As Long
Dim countY As Long

On Error GoTo EndMacro

Application.ScreenUpdating = False
Application.Calculation = xlCalculationManual
Application.EnableEvents = False

    Set area = Selection
    sizeX = area.Columns.Count
    sizeY = area.Rows.Count
    If sizeX > 1 And sizeY > 1 Then
        sizeX = sizeX - 1
        While sizeX > 0
            For countY = 2 To sizeY
                If area.Cells(countY, sizeX).Value = "" And
area.Cells(countY, sizeX + 1).Value <> "" Then
                    area.Cells(countY, sizeX).Value = area.Cells(countY - 1,
sizeX).Value
                End If
                 'area.Cells(countY, sizeX) = "x"
            Next
```

```
            sizeX = sizeX - 1
        Wend
    End If
EndMacro:
Application.ScreenUpdating = True
Application.Calculation = xlCalculationAutomatic
Application.EnableEvents = True

End Sub
```

1)

not included in the base software pack, only awailable for OEMs