Inhaltsverzeichnis

SKDSC - The SKDS Command-Line-Compiler

Since version 2 of SKDS the generation and the use of a module file is separated. The files are used as binary data in the SKDS Windows application, they are written as ASCII text with a normal text editor, and they are transfered from one type to another with the command-line tool skdsc.

skdsc takes a text File as input, runs through it, includes other files, if it finds an include command, generates the table structure for the main table, encrypts the output into a SCS file with the public key of the recipient.

SKDSC-Options

The program call is

```
skdsc [options] -x user file[.ext]
```

where the options are

-help
Displays short help information

-v Display
program outputs

-u <Username>
Set User name <Username> as author of that file (do not use blanks!)
This user name will appear as the author's name when the file has been loaded in SKDS

-E
Pre-process only to STDOUT; do not encrypt or save

-o <file>
Place the output into <file>

-I <Include-path>
add <includepath> to searchpath
with the -I option(s) one or more search paths can be named. When skdsc finds an include-statement in the input file, it runs through all search paths until a file with the given name is found.

-x <user>
define the user who will be able to execute the script (see Key- Management).

All Options are case independent.

#-commands

Inside a script file some commands are possible. The common format for these commands, embedded in comment-signs, is

```
(*#command [parameter[,parameter]]*)
```

The following #- commands are supported:

menuitem

```
(*#menuitem "*" "U" "Function desciption" "functionname" "Group" "Input"*)
```

This command defines an entry for the main function table. The command needs 6 parameters, enclosed in „ “ , separated with blanks. The parameters are:

1. The inital value for the the Status-Row
2. The update type (see [update], page 4.)
3. The text, which will appear in the description cell
4. The function name: The function, which will be executed, when this item has been selected
5. The group to which this function should belong
6. This function type (input, output or parameter)

include

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

This command searches for the file given as a parameter in the search-path (defined with the -I option) and in the actual directory. If a file with that name is found, the actual text-line is replaced with the content of the file.

Then the new complete text is investigated further for other include-commands.

So it is possible to include first file b into file a, where b also includes file c, so that in the end file a also contains b and c.

Caution: There is no control whether a file has been included before, so if there is, for example, a file b which includes file a, and a includes b, this will result in an infinitive loop and a program stop

There are two kinds of parameters possible, which differ in the enclosing signs:

<file>: The file will first be searched for in the Include-Search-Path and then in the actual directory.

„file“: The file will first be searched for in the actual directory and then in the Include-Search-Path

optinclude

```
(*#optinclude <file_to_include> Dummypart *)
```

Especially to solve the problem, that on one hand confidential information must not be included in the offical SKDS script archive, but on the other hand the strong want from the users not to type in this information at each program start again and again, the command `optinclude` is implemented.

It allows to store sensitive information in a separate file and compile it in, „only if it's there“. Here an example:

In a normal script, the line

```
(*#optinclude <mod_Sec.code>seccode: string =""*)
```

is included.

Now the file is compiled with skdsc. As there's no file „mod_Sec.code“ in the same directory, skdsc inserts the dummypart, so the line becomes to

```
seccode: string =""
```

But if we have now the file mod_Sec.code in the actual directory, containing the following text:

```
 seccode: string ="001122334455";
```

and compiling our previous text again, the line becomes to

```
(* Including optional mod_Sec.code: *)
storedseckey :='1122334455' ;
(* End of optional include of mod_Sec.code *)
```

so the line changes, depending if the file to include is there or not.

This is very helpful during the script development: During testing the engineer stores the key in a separate file, but when he publish the main script file, he don't has to remember to remove the key first out of the source. When also the secret data is stored in files with a unique file extension (like .code), other software like revision controls can automatically ignore this suffix.