GHS

DES

Version 304

Reference Manual

15. June 2005

GreenHouse
Software & Consulting
Karl-Heinz Weber
Heinrichstraße 12
D-45711 Datteln/Horneburg

This is the most recent DES implementation of GreenHouse, optimized to run on NSK.

It comes with these files:

File name in distribution location[1]	Legend	File name on NSK	File code on NSK
CBCDES.700	Native CBD DES encode/decode procedures	CBCDES	700
COMPTEST.101	TACL Macro to compile TestSRC	COMPTEST	101
DES.700	Native DES procedure	DES	700
DESDECS.101	DES external declarations	DESDECS	101
DESLIB.100	NON Native DES procedure library	DESLIB	100
DESTEST.100	NON Native DES test program, ready to run (based on DESLib.100 and DESTESTS.101)	DESTEST	100
DESTEST.700	Native DES test program, ready to run (based on native procedures and DESTESTS.101)	DESTESTN	700
DESTESTS.101	DES test program, source	DESTESTS	101
ECBDES.700	Native ECB DES encode/decode procedures	ECBDES	700
INITDES.700	Native INITDES procedure	INITDES	700
INITKEY.700	Native INITKEY procedure	INITKEY	700
PWHASH.700	Native PWHash procedure	PWHASH	700
TEST.100	NON Native but accelerated test program (based on DESLib.100 and TestSRC.101)	TEST100	100
TEST.700	Native test program (based on native procedures and TestSRC.101)	TEST700	700
TESTSRC.101	Test program to check all available DES versions	TESTSRC	101
TOECBDES.700	Native Trio ECB DES encode/decode procedures	TOECBDES	700
TRCBCDES.700	Native Triple CBC DES encode/decode procedures	TRCBCDES	700
TRECBDES.700	Native Triple ECB DES encode/decode procedures	TRECBDES	700
VALIDATE.101	DES triples to validate the DES code	VALIDATE	101

Test code

To test the DES code, perform these steps:

1. Upload DESTEST.100 in binary mode onto the NSK system, name the file DESTEST, and change the file code to 100 (**`FUP ALTER DESTEST,CODE 100`**).
   To get the native program, upload DESTEST.700 in binary mode onto the NSK system, name the file DESTESTN, and change the file code to 700 (**`FUP ALTER DESTEST,CODE 700`**)

2. Upload VALIDATE.101 in ASCII mode onto the NSK system into the same location as DESTEST and name the file VALIDATE

3. Run DESTEST

Check Functionality

To check the functionality of all available DES procedures do:

1. Upload TEST.100 or TEST.700 in binary mode onto your NSK, name the file TEST, and change the file code to 100 or 700 respectively.

2. Run TEST

[1] The file name extension reflects the NSK file code

TEST.100 as well as TEST.700 are derived from the source file TESTSRC.101.

To compile TESTSRC do the following:

1. Upload the TACL Macro COMPTEST.101 in ASCII mode into the same location as TESTSRC and name the file COMPTEST.

2. Upload DESDECS.101 in ASCII mode onto the NSK system, and name the file DESDECS.

3. Upload DESLIB.100 in BINARY mode onto the NSK system, and name the file DESLIB.

4. Upload TESTSRC.101 in ASCII mode onto the NSK system, and name the file TESTSRC.

5. Upload the following files in BINARY mode into the same location as TESTSRC as well as COMPTEST:

File name	File name on NSK	File Code on NSK
DES.700	DESN	700
ECBDES.700	ECBDES	700
INITDES.700	INITDES	700
INITKEY.700	INITKEY	700
PWHASH.700	PWHASH	700
TOECBDES.700	TOECBDES	700
TRCBCDES.700	TRCBCDES	700
TRECBDES.700	TRECBDES	700

6. Execute the COMPTEST TACL Macro.
   It compiles a NON native version of the test program (Test.100) as well as a native version (Test.700).

Beside the DES code, I provide the ISO Hash based password hashing procedure PWHash: It hashes a user supplied string of up to 2**15 bytes to a 16 byte string, based on ISO 10118-2.

The DES procedures come in two flavors:

1. NON native for e.g. TAL programs
2. Native for e.g. pTAL as well as C programs

NON native library
The non native library is named DESLIB.100. It holds all procedures, defined in DESDECS. Use the ?SEARCH compiler directive to bind the library at compile time, or BINDER.

Native library
The NLD utility is not as clever as BIND, and binds ALL code it finds rather than the really required. To allow a DES function specific NLD bind, all native DES functions are shipped in separate files. Use NLD to bind these files to your code.

Procedure dependencies of native procedure calls:

DES Function	Native Procedure	Required functions
Cipher Block Chaining	CBCDES	INITDES, INITKEY, DES
Basic ECB DES	DES	INITDES, INITKEY
Electronic Code Book	ECBDES	INITDES, INITKEY, DES
DES Initialization	INITDES	Basic function
Key Initialization	INITKEY	Basic function
Password Hash	PWHASH	INITDES, INITKEY, DES
Trio ECB DES	TOECBDES	INITDES, INITKEY, DES
Triple CBC DES	TRCBCDES	INITDES, INITKEY, DES
Triple ECB DES	TRECBDES	INITDES, INITKEY, DES

e.g. to use the ECBDES function, you also need INITDES, INITKEY and DES.

This DES code performs best in Native code.

A sample run of TEST.100 (NON Native, but accelerated code) on a S7000 shows these numbers:

```
$GHS1 NEWDES 4>test100
       ECB DES with   1048576 bytes takes 00:00'03,626.107
Triple ECB DES with   1048576 bytes takes 00:00'10,770.003
  Trio ECB DES with   1048576 bytes takes 00:00'10,829.124
       CBC DES with   1048576 bytes takes 00:00'04,221.248
Triple CBC DES with   1048576 bytes takes 00:00'12,109.238
          Hash with     16384 bytes takes 00:00'02,633.885
$GHS1 NEWDES 5>
```

The Native program TEST.700 is by some 15% faster:

```
$GHS1 NEWDES 5> test700
       ECB DES with   1048576 bytes takes 00:00'03,212.522
Triple ECB DES with   1048576 bytes takes 00:00'09,583.699
  Trio ECB DES with   1048576 bytes takes 00:00'09,581.116
       CBC DES with   1048576 bytes takes 00:00'03,696.043
Triple CBC DES with   1048576 bytes takes 00:00'10,696.983
          Hash with     16384 bytes takes 00:00'02,078.661
$GHS1 NEWDES 6>
```

In other words: This DES implementation, running in ECB mode on a S7000, processes one (1) MB of data in some 3.2 seconds.

DES Functions and Procedure Calls

Release History

Date	Changes
11. February 2004	The LEN fields in the compound functions now treated as unsigned integers, allowing to represent values between 0 and 65535.

Basic DES Functions

ECB DES Encode/Decode

The Electronic Code Book mode of DES processes the input data as follows:

1. Encode/Decode 8 byte with 8 bytes of key

Key length is 8 bytes, where 7 bit per bytes are used = 56 bit real key length.

Triple ECB DES Encode/Decode

The Triple Electronic Code Book mode of DES processes the input data as follows:

1. Encode/Decode 8 byte with 1st 8 bytes of key
2. Decode/Encode result with 2nd 8 bytes of key
3. Encode/Decode result with 1st 8 bytes of key

Key length is 16 bytes, where 7 bit per bytes are used = 112 bit real key length.

Trio ECB DES Decode/Encode

The Trio Electronic Code Book mode of DES processes the input data as follows:

1. Encode/Decode 8 byte with 1st 8 bytes of key
2. Decode/Encode result with 2nd 8 bytes of key
3. Encode/Decode result with 3rd 8 bytes of key

Key length is 24 bytes, where 7 bit per bytes are used = 168 bit real key length.

CBC DES Encode/Decode

The Cipher Block Chaining method of DES processes the input data as follows:

Encode:
1. XOR 8 bytes with ICV
2. Encode with key
3. ICV := Result

Decode:
1. Decode 8 bytes with key
2. XOR result with ICV
3. ICV := Result

Key length is 8 bytes, where 7 bit per bytes are used = 56 bit real key length.

Triple CBC DES Encode/Decode

The Cipher Block Chaining method of DES processes the input data as follows:

Encode:
1. XOR 8 bytes with ICV
2. Encode 8 byte with 1st 8 bytes of key
3. Decode result with 2nd 8 bytes of key
4. Encode result with 1st 8 bytes of key
5. ICV := Result

Decode:
1. Decode 8 byte with 1st 8 bytes of key
2. Encode result with 2nd 8 bytes of key
3. Decode result with 1st 8 bytes of key
4. XOR result with ICV
5. ICV := Result

Key length is 16 bytes, where 7 bit per bytes are used = 112 bit real key length.

Literals

The DES code needs a few literals, which are defined in the DESLiterals section of DESDecs:

```
?Section DESLiterals
!
! Literals
!
Literal Decode^        =  0, ! decode crypted information to plain information
        Encode^        =  1, ! encode plain information to crypted information
        NoError        =  0, ! operation OK
        DatanotModulo8 = -1; ! Error used by compound functions
```

CBCDESDecode

The CBCDESDecode function is a compound function, which decodes a CBC[2] DES encoded string of data.

NON native code: DESLIB
Native Code: CBCDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc CBCDESDecode(In:Len,Out,Key,ICV,SEGAddr) Variable;

  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT Key;      ! String:ref:8    ;input
 [String .EXT ICV;      ! String:ref:8    ;input]
  Int(32)     SEGAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes[3]
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, CBCDESDecode returns a -1 (data not modulo 8).
OUT String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:8	input	User supplied key.
ICV String:EXT:Ref:8	input	User supplied Initial Chaining Vector Defaults to all zeros.
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

[2] CBC = Cipher Block Chaining
[3] The Len-values is treated as an unsigned integer, representing values between 0 and 65,535

CBCDESEncode

The CBCDESEncode function is a compound function, which encodes a string of data using the CBC DES method.

NON native code: DESLIB
Native Code: CBCDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc CBCDESEncode(In:Len,Out,Key,ICV,SEGAddr) Variable;

  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT Key;      ! String:ref:8    ;input
 [String .EXT ICV;      ! String:ref:8    ;input]
  Int(32)     SEGAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, the procedure returns a -1 (data not modulo 8).
OUT String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:8	input	User supplied key.
ICV String:EXT:Ref:8	input	User supplied Initial Chaining Vector Defaults to all zeros.
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

DES

NON native code: DESLIB
Native Code: DES

DES is a basic function of this DES implementation.
It is the cryptographic engine, used by all compound functions such as ECBDESEncode, CBCDESDecode, PWHash etc.
DES processes 8 bytes at a time.

```
Proc DES(In,Out,Key_,DESOp,SEGAddr);

  Int    .EXT In;      ! Int:EXT:Ref:4    input     data in
  Int    .EXT Out;     ! Int:EXT:Ref:4    output    data out
  Int    .EXT Key_;    ! Int:EXT:Ref:64   input     key table
  Int         DESOp;   ! Int:Value:1      input     operation mode
  Int(32)     SEGAddr; ! Int(32):Value:1  input     DES Segment base address
```

IN Int:EXT:Ref:4	input	Defines a 4*16 bit word (= 8 bytes) array that is to be processed.
OUT Int:EXT:Ref:4	output	Is a 4*16 bit word array where the operations outcome is returned.
Key_ Int:EXT:Ref:64	input	Key vector, that is build by Init_Key_
DESOp Int:Value:1	input	Defines the operation mode: 0 = decode <> 0 = encode
SegAddr Int(32):Value:1	input	DES Segment base address, returned by Init_DES_

ECBDESDecode

The ECBDESDecode function is a compound function, which decodes an ECB[4]DESEncode encoded string of data.

NON native code: DESLIB
Native Code: ECBDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc ECBDESDecode(In:Len,Out,Key,SEGAddr);

  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT Key;      ! String:ref:8    ;input
  Int(32)     SEGAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, the procedure returns a -1 (data not modulo 8).
Out String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:8	input	User supplied key.
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

[4] ECB = Electronic Code Book

ECBDESEncode

The ECBDESEncode function is a compound function, which encodes a string of data using the ECB DES method.

NON native code: DESLIB
Native Code: ECBDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc ECBDESEncode(In:Len,Out,Key,SEGAddr);

  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT Key;      ! String:ref:8    ;input
  Int(32)     SEGAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, the procedure returns a -1 (data not modulo 8).
OUT String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:8	input	User supplied key.
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

InitDES

InitDES is a basic function of this DES implementation.
It has to be called once before the first DES operation is performed.
This procedure call allocates a user defined flat segment and returns its base address, which has to be passed to all subsequent DES procedure calls.
Flat segments require D30 or better.

NON native code: DESLIB
Native Code: INITDES

```
Int Proc InitDES(DES_Segment,SegAddr);

  Int          DES_Segment;   ! Extended Segment Number of DES Table to be used
  Int(32) .EXT SegAddr;       ! Segment address
```

Error Int	Returned value	0 = OK <> 0 = error number returned from Segment_Allocate_
DES_Segment Int:Value:1	Input	Segment number to be used. This number should be < 1024, and exclusively used in the calling program.
SegAddr Int(32):EXT:Value 1	output	DES Segment base address, build by Init_DES_ and used by all compound functions.

InitKey

InitKey is a basic function of this DES implementation.
It initializes a key field with key related variables, which has to be passed to the basic procedure DES.
It is implicitly used by all compound functions.

NON native code: DESLIB
Native Code: INITKEY

```
Proc InitKey(Key,Key_);

  Int    .EXT Key;       ! Int:EXT:Ref:4   input
  Int    .EXT Key_;      ! Int:EXT:Ref:64  output; to be passed to DES
```

Key Int:EXT:Ref:4	input	Key to be used for the DES operation.
Key_ Int:EXT:Ref:64	output	Key table, to be passed to the DES_ procedure

PWHash

A user supplied string (pass phrase) is compressed to a 16 byte hash value using the ISO 10118-2 hashing function.
The initial chaining vector of the function can be initialized with the users name. This ensures, that different users using the same pass phrase generate different hash values.

NON native code: DESLIB
Native Code: PWHASH

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc PWHash(PW:PWLen,User:UserLen,Result,SEGAddr) Variable;

  String .EXT PW;         ! String:EXT:Ref:* ;input   PW to hash
  Int         PWLen;      ! Int:value:1      ;input   length of PW (1 .. 65535)
  String .EXT User;       ! String:EXT:Ref:* ;input   User
  Int         UserLen;    ! Int:Value:1      ;input   length of User (1 .. 32)
  String .EXT Hash  ;     ! String:EXT:Ref:16;output  Hash result
 [Int(32)     SegAddr;    ! Int(32):Value:1  ;input   DES segment address]
```

Error Int	Returned value	0 = OK 1 = password out of range (length is zero [0]) 2 = user name out of range (smaller than 1 or larger than 32)
PW String:EXT:Ref:*	input	Data string to be hashed The maximum size is 65,535 bytes
PWLen Int:Value:1	input	Number of bytes in string PW. This string is case sensitive!
User String:EXT:Ref:*	input	Users name; used as ICV This string is case sensitive!
UserLen Int:Value:1	input	Number of bytes in User. Has to be 0 .. 32
Hash String:EXT:Ref:16	output	16 byte hash result
SegAddr Int(32): Value:1	input	DES Segment base address, build by Init_DES_ and used by all compound functions.

TrioECBDESDecode

The TrioECBDESDecode function is a compound function, which decodes a data string created by TrioECBDESEncode.

NON native code: DESLIB
Native Code: TOECBDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc TrioECBDESDecode(In:Len,Out,LKey,SegAddr);
  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT LKey;     ! String:ref:24   ;input
  Int(32)     SegAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, the procedure returns a -1 (data not modulo 8).
OUT String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:24	input	User supplied key
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

TrioECBDESEncode

The TrioECBDESEncode function is a compound function, which encodes a data string using the trio ECB DES method.

NON native code: DESLIB
Native Code: TOECBDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc TrioECBDESEncode(In:Len,Out,LKey,SegAddr);
  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT LKey;     ! String:ref:24   ;input
  Int(32)     SegAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, the procedure returns a -1 (data not modulo 8).
OUT String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:24	input	User supplied key
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

TripleCBCDESDecode

The TripleCBCDESDecode function is a compound function, which decodes a Triple CBC DES encoded string.

NON native code: DESLIB
Native Code: TRCBCDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc TripleCBCDESDecode(In:Len,Out,Key,ICV,SEGAddr) Variable;

  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT Key;      ! String:ref:16   ;input
 [String .EXT ICV;      ! String:ref:8    ;input]
  Int(32)     SEGAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, CBCDESDecode returns a -1 (data not modulo 8).
OUT String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:16	input	User supplied key.
ICV String:EXT:Ref:8	input	User supplied Initial Chaining Vector Defaults to all zeros.
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

TripleCBCDESEncode

The TripleCBCDESEncode function is a compound function, which encodes a data string using the Triple CBC DES method.

NON native code: DESLIB
Native Code: TRCBCDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc CBCDESEncode(In:Len,Out,Key,ICV,SEGAddr) Variable;

  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT Key;      ! String:ref:16   ;input
 [String .EXT ICV;      ! String:ref:8    ;input]
  Int(32)     SEGAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, the procedure returns a -1 (data not modulo 8).
OUT String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:16	input	User supplied key. 16 bytes
ICV String:EXT:Ref:8	input	User supplied Initial Chaining Vector Defaults to all zeros.
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

TripleECBDESDecode

The TripleECBDESDecode function is a compound function, which decodes a data string using the triple ECB DES method.

NON native code: DESLIB
Native Code: TRECBDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc TripleECBDESDecode(In:Len,Out,LKey,SegAddr);
  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT LKey;     ! String:ref:16   ;input
  Int(32)     SegAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, the procedure returns a -1 (data not modulo 8).
OUT String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:16	input	User supplied key. 16 Bytes
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

TripleECBDESEncode

The TripleEBCDESEncode function is a compound function, which encodes a data string using the triple ECB DES method.

NON native code: DESLIB
Native Code: TRECBDES

Required procedures:
- DES
- INITDES
- INITKEY

```
Int Proc TripleECBDESEncode(In:Len,Out,LKey,SegAddr);
  String .EXT In;       ! String:ref:*    ;input
  Int         Len;      ! Int:value       ;input
  String .EXT Out;      ! String:ref:*    ;output
  String .EXT LKey;     ! String:ref:16   ;input
  Int(32)     SegAddr;  ! Int(32):Value:1 ;input
```

Error Int	Returned value	0 = operation OK -1 = Len not modulo 8
In String:EXT:Ref:*	input	Defines an array of data to be processed. The maximum length processed is 65,535 bytes
Len Int:Value:1	input	Number of bytes to be processed. Has to be a multiple of 8. When Len does not fit this, the procedure returns a -1 (data not modulo 8).
OUT String:EXT:Ref:*	output	Is a byte array where the operations outcome is returned.
Key String:EXT:Ref:16	input	User supplied key. 16 Bytes
SegAddr Int(32):Value:1	input	DES Segment base address, build by Init_DES_

Example TAL program

```
!
!Source in the DES declarations, and point the compiler to the procedures
!
?Source DESDECS    ! DES declarations
?Search DESLIB     ! DES procedure library
!
! Sample procedure
!
Proc Crypt Main;
----------------
  Begin

    String .Data[0:n];
    Int     DataLen;
    String .Result[0:n];

    String .User[0:31] := "A";
    Int     UserLen    := 1;

    String .PassPhrase[0:127];
    Int     PassPhraseLen;
    String .Key[0:15];

    Int(32) SegmentAddress;
!
! Procedure Body
!
    Initializer;                          ! get all start-up stuff
    InitDES(1,SegmentAddress);            ! central DES initialization
    .
    . -- lots of code, e.g. get the users pass phrase into
    . -- variable PassPhrase etc.
    .
!
! Hash user supplied pass phrase down to 8 bytes.
!
    PWHASH(PassPhrase:PassPhraseLen       ! user supplied pass phrase
          ,User:UserLen                   ! users name used as ICV
          ,Key,SegmentAddress);

    User I;
      For I := 0 to 7 do Key[I] := Key[I] XOR Key[I+8];
    Drop I;
    .
    . -- lots of additional code, e.g. data preparation
    .
```

```
!
! Run the Encode function
!
    ECBDESEncode(Data:DataLen,Result,Key,SegmentAddress);  ! encode Data
   .
   .
   .
!
! Running the Decode function on the result of the Encode function should
! create the original data!
!
.   ECBDESDecode(Result:DataLen,Data,Key,SegmentAddress);  ! decode Result

  End;
```

To get additional programming hints, please have a look at the DESTEST and TESTSRC source files.

11. February 2004
GreenHouse Software & Consulting
Carl Weber
Carl.Weber@GreenHouse.de