Elite EL

Developer Guide v3.4.0.0

SENSESHIELD

Copyrights and Trademarks

The Elite EL with its technical documentation is copyrighted (C) 2017 to present by Senseshield Technology Co., Ltd (Senseshield). All rights reserved.

All products referenced throughout this document are trademarks of their respective owners.

All attempts have been made to make the information in this document complete and accurate. Senseshield is not responsible for any direct or indirect damages or loss of business resulting from inaccuracies or omissions. The specifications contained in this document are subject to change without notice.

Contact

Company Name: Beijing Senseshield Technology Co., Ltd

Address: Suite 510,Block C,Internet Innovation Center,Building 5,
No.10,Xibeiwang East Road,Haidian District,
Beijing China

Tel.: +86-10-56730936
Fax: +86-10-56730936-8007

Sales: sense@sense.com.cn
Website: www.sense.com.cn

License Agreement

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THE CONTENTS THEREOF AND/OR BEFORE DOWNLOADING OR INSTALLING THE SOFTWARE PROGRAM. ALL ORDERS FOR AND USE OF THE Elite AND/OR EL FAMILY PRODUCTS (including but not limited to the Kit, libraries, utilities, diskettes, disc, Senseshield® and/or Senseshield® keys, the software component of Senseshield/or EL and the EL License Guide) (hereinafter “Product”) SUPPLIED BY Senseshield Technology Co., Ltd (hereinafter "Senseshield") ARE AND SHALL BE, SUBJECT TO THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT.

This document is a legally binding agreement between you (either an individual or an entity) and Senseshield®. If you are not willing to be bound by the terms of this agreement, you should promptly (and at least within 3 days from the date you received this package) return the unused developer's kit and the programmer's guide to Senseshield. Use of the software indicates your acceptance of these terms.

- **GRANT OF LICENSE**

  The software of the Product is being licensed to you, which means you have the right to use the software only in accordance with this License Agreement. You may (a) copy the software for internal use, (b) modify the software for the purpose of integrating with your application and (c) merge the software with other programs.

- **NON-PERMITTED USES**

  Except explicitly permitted in this License Agreement, you may not (a) copy, modify, reverse engineering, decompose, assemble the Product in whole or in part, or (b) sell, lease, license, transfer, distribute all or part of the Product or rights granted in this License Agreement.

- **LIMITED WARRANTY**

  After the date of purchase, Senseshield provides 24-month warranty that the Senseshield EL key has no material and manufacturing defects substantially. All the responsibilities of Senseshield Technology Co., Ltd and all the compensation you can get under warranty are: you can require replace/repair the Product or accept other remedial measures.

- **LIMITATION OF LIABILITY**

  Under any circumstances, Senseshield will NOT be liable for any damages arising out of usage or inability of the Product, including but not limited to: loss of data, loss of profits, and other special, incidental, joint, secondary or indirect loss.

  Except for the limited warranty offered to the original buyer, Senseshield is not responsible for providing any insurance to anyone on the product, performance and service including merchantability and fitness for a particular purpose.

  The entire product, including Senseshield EL, the software, the document, other material shipped as accessories, and backups made by you are copyrighted by Senseshield Security GmbH.

- **TERMINATION**

  Your failure to comply with the terms of this License Agreement shall terminate your license and this License Agreement.

Contents

Copyrights and Trademarks ..... **I**
Contact ..... **II**
License Agreement ..... **III**
Contents **4**
Overview **6**
About the Guide 6
What is EL? 6
Code Port Technology 6
Development Flow 7
User Running Environment 7
What is with EL? 8
Development Tool Kit 8
Hardware Device 8
Features 8
Top-secured Chip 8
Automatic Self-locking Mechanism 8
Global Unique Serial Number 9
Built-in Timer 9
Concrete Encryption 9
Multi-applications 9
Start to Know **10**
File Type 10
Executable File 10
Data File 10
Key File 10
Read-Write Attributes 10
File System 11
PIN 11
Developer PIN 11
User PIN 11
Using PINs 11
Prepare to Use **13**
Demo Software 13
Code Porting 14
Create a Project 14
Compile Codes 14
Compile and Test Codes 15
Download Codes to Device 15
Modify Computer Codes 17
Test Protection Results 19
Analyze Demo Structure 19
Developing Codes **21**
Compile Internal Codes 21
Running Modes 21
Internal Code Structure 21
Input & Output 22
Memory Structure 23
Lib Files of System Function 26
Data Type 26
Compilation Suggestions 26
Accessing Device 27
General Level Access 27
Developer Level Access 29

User Level Access 32
Testing Codes 33
Using Compiler 35
Using Keil C51 35
Project Creation and Management 36
Code Compilation and Debugging Configuration 38
Debugging Codes 40
Executable File Format 44
Using Tools 45
Development and Test Tool 45
Device Reconnection 45
Device Reset 46
PIN Management 47
Download Files 48
Empty Directory 49
Execute File 50
PC Keys 51
SES References 53
Flow Control 54
Input & Output 55
File Operation 56
Mathematics 64
Single Precision 64
Double Precision 64
Cryptographic Algorithm 120
Memory Operation 142
Time Control 152
Macro and Auxiliary Function 161
Get Device Info 173
Error Code List of SES 174
Boundary Limitation of Double Precision Float 175
API Reference 177
API List 179
Error Code List of API 208
Appendix A: Cryptographic API 210
Key Operating Functions 211
Cryptographic Algorithm Functions 217
Error Code List of Cryptographic API 224

Overview 1

About the Guide

Type	Model	Hardware Version	Guide Version	Releasing Date
Local	Genii, Std, Rtc	v2.xx, v5.xx	v3.4.0.0	2017.02.21

- **Conventions Used**

The following conventions are used throughout this document:

Italic	Words in italic represent file names and directory names. Parameter of API
Bold	Words in boldface represent keystrokes, menu items, and window names and fields
CAP	Predefined struct and Macro of API
⚠	The caution icon flags some content you have to be careful of.

- **Document Improvement**

Document Writing Team dedicates to insure the accuracy and completeness of context. Your feedback will assist them to make continuous improvement on EL document. Please do not hesitate to email us, sense@sense.com.cn.

What is EL?

EL Series is the most secure and effective dongle against piracy in history. The innovation of combining “Code Port” and “Smart Card” changes the software encryption industry completely, now becomes the trademark of high-end encryption dongles.

Code Port Technology

It is essential to introduce the Code Port technology by the following diagram:

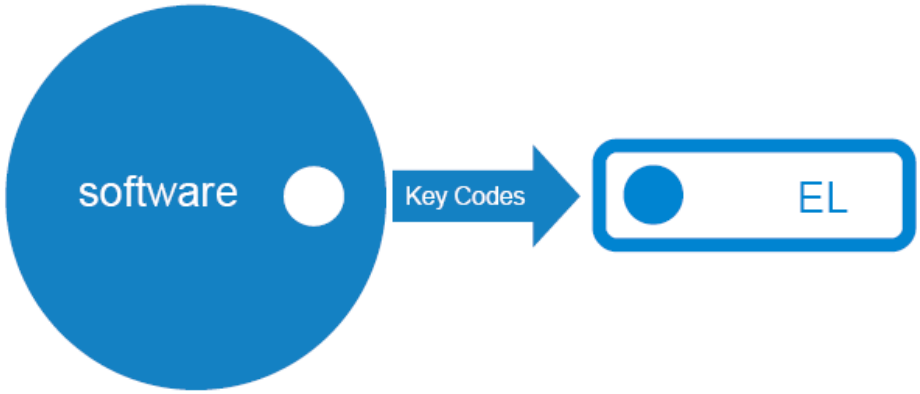


Figure 1 Principle of Code Port

The principle of design is that each ported code segment becomes an independent

"Executable file", termed "EXF". While the software is running and requiring ported codes to accomplish certain tasks, EL shall execute these EXFs with the internal operating system[1]. If the device is missing from being plugged, the incomplete software will become unworkable thoroughly.

In fact, the complexity of EXF is up to the device memory. For instance, the 64K version almost has no limits on number of EXFs (The file header requires storage, therefore, over-numbered EXFs cause wastage of memory). In maximum, it is able to hold approximately 10000 lines of portable codes.

Under most circumstances, you can make EXF by slight modification of codes for EL is supportive to codes compiled by standard C language. If the software is developed in other high-level languages, such as Delphi, it is required to re-write the ported part in C language. Although some efforts are required for the "translation" between high-level languages, if "translating" several hundred-line codes can keep you thoroughly away from piracy, they are definitely paid off. In the subsequent part of this text, we usually term the codes ported into EL "EXF".

Development Flow

1. Develop and debug software with full concentration, disregarding completely the software protection strategy;
2. Select critical code segments from the software and compile these codes via the compiler compatible with EL before writing into device;
3. Add the application of invoking EL to where the codes are dig out from the software;
4. When the software is running, these ported codes will be required to run inside EL and return the designated results;



Despite the fact that Code Port is the most fundamental software protection method of EL, this does not mean you could discard the conventional strategy. Under most circumstances, they are reciprocally complementary, yet it is imperative to remember: the software is only secured by porting codes into EL.

User Running Environment

EL runs in HID mode:

- **HID**

It is called driverless mode. In this mode, EL is recognized by the operating system as a standard HID (Human Input Device) device. In comparison with the USB mode, HID mode needs to sacrifice some advanced functions and versatility (such as multi-process access to the device).

Linux and Mac are supportive under HID mode.

[1] The internal operating system of EL complies with the international standard of the Smart Card operating system. You can get relevant information about "Smart Card Operating System" by consulting ISO7816 serial standards.

What is with EL?

Development Tool Kit

The key to make the best of EL is the IDE (Integrated Development Environment). For EL supports standard C language development, the C51 of Keil Software is highly recommended. You can visit http: //www.keil.com to get information about is powerful compilation software.



There is patch that is essential to apply on Keil C51 (*SDK\IDE\keil\C51_v7.01_ eval.rar*). You can locate it under the directory (*SDK\IDE\keil\Patch*). From its official website, it is available to download the latest evaluation version of C51 which has 2k limitation on operating memory.

Extra tools are also provided in the development tool kit:

- **Development Test Tool.**
  It is used to develop and test codes ported to EL and perform general management;
- **User Test Tool.**
  It can be distributed to software end users, assisting to address the possible configuration errors;

Hardware Device

Developed fully based on smart card technology and employing 32-bit, high-performance CPU, the hardware device is the running environment and storage media for EXFs. EL is offered mainly as a USB interface device. As an option, it can also be offered in the form of standard Smart Card. EL is a highly integrated product for CPU, RAM, FLASH and USB communication module all co-work in one smart card chip, thus enhancing tremendously the security and stability.

Features

Top-secured Chip

ISO made explicit manufacturing norms (ISO/IEC 10373, ISO/IEC 15408) and formulated the international criteria EAL (Evaluation Assurance Level) into 1-7 grades for scientifically evaluating the chip security. The higher of the grade, the securer the chip is. EAL 5+ is a known upper limit for current technical products.

EL is equipped with the most advanced Infineon 32-bit smart card chip reaching EAL 5+ in the global hi-tech sector, that practically enable Elite with the outstanding effectiveness in resisting of major attacks such as Electronic Attack (SPA and DPA), Probe Attack (SiShell), Chip Dissection and Debug Port.

Automatic Self-locking Mechanism

The access to the chip is protected by PIN, and the maximum re-tries is pre-set by software developers. For instance, under a dictionary attack, once re-tries exceed the pre-set value,

the chip will trigger off a self-locking mechanism and block all external running operations.

Global Unique Serial Number

A Global Unique Serial Number (GUSN) which cannot be altered or rewritten is assigned to a smart card chip only. This puts an end to the piracy and enables the effective management on the released chips. Furthermore, by cooperating with Senseshield, software developers are able to gain a unique Customer ID, which in effect addresses the critical issue of the reproduced and fraudulent dongle.

Built-in Timer

Timer is a useful component for software protection that applies to time-limited usage and anti-tracking. EL has a built-in timer lasting more than 10 hours that sustains much longer than similar products of competitors.

Concrete Encryption

EL contains the hardware co-processor of RSA (1024 bits) and DES/TDES cryptography algorithm and offers SHA-1 algorithm function and hardware true random number generator. Using the cryptography algorithms of EL, you can also log on the Windows operating system, encrypt and decrypt email and perform various kinds of complex network security functions.

Multi-applications

Based on these cryptography functions, EL presents a remote updating and sales management platform. The software protected by EL can be billed and updated via the Internet. By this means, the management of software sales channels can be carried out. As EL opens the bottom-layer technical details for remote updating, you can create your own remote updating and sales management platform using the low-level protocol.

Start to Know 2

File Type

EL supports three types of files: Executable Files, Data Files and Key Files, none of which can be read directly from the EL device but created and modified with privileges of Developer PIN.

The files in EL are identified via a 16-bit (2-byte) unsigned integer. Some parts of the ID are reserved by the system. They are *0x0000, 0x0015, 0x0016, 0x0018, 0x001e, 0x3f00, 0x3f01, 0x3f02, 0x3f03, 0x3f04*.

Executable File

An executable file can be loaded and run on the internal operating system of EL, the result of being compiled from ported codes.

It can operate other files in the same directory through the system function in EL device, such as reading and writing data files. When it comes to the operation of being read & written by other executable files, the feasibility of operation depends on its read-write attributes.

Corresponding to the following running mode:

- **VM Executable File**
  It is also known as General Executable File. For the detailed description of the running mode, please refer to Chapter “Developing Codes”.

Data File

The data file is designed only to store binary data. The program on the computer cannot be read directly from the data files from EL. To access the data files in EL, you need to do it via EXF.

Key File

Only the RSA key file is supported now. You can consult Chapter “SES Reference” for more detailed information.

Read-Write Attributes

R: Read W: Write E: Execute N: None

		Developer **PIN L**evel	User **PIN L**evel	**EXF**
Executable File	Unread/Unwritten	W	E	N
	Internal Read-write	W	E	R/W
Data File		W	N	R/W
Key File		W	N	W

Table 1: Read-Write Attributes of EL files

There is specification in detail governing the read- write attributes involved in internal file operation in EL. For details, please refer to the concerned file operation part in Chapter “SES Reference”.

File System

EL supports the “directory-file” structure. Each directory should be assigned a Directory ID, and any directory’s parent, sibling and child directory must not share the same Directory ID.

Only one root directory can be set up in EL. It has maximum three levels of directories and takes up all the entire device memory. Unless it is necessary, we do not recommend the use of multi-level directory to avoid chaos in management.

- **Create a Directory**
  To create a directory, the space occupied by the directory shall be allocated. Once the directory is created, its size is unchangeable.

- **Delete Root Directory**
  The root directory is deletable. When the root directory is deleted, all the contents inside, such as sub-directories and files will be deleted as well.

- **Clear Sub-directory**
  A sub-directory can only be cleared, but not deleted directly. Clearing a sub-directory will delete all the contents within the directory, such as sub-directories, files, etc. After the sub-directory is cleared, its Developer PIN and User PIN will be restored to default values. To wipe out a sub-directory to release all seized space, it is essential to clear its upper-level directory.

PIN

Two levels of PINs are set up in EL for being authorized with leveled privileges:

Developer PIN

Developer PIN is 24-byte long. Its default value is “123456781234567812345678” (*0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38*). Before EL is released, make sure to modify the default Developer PIN.

After the PIN is successfully authenticated, the developer can configure authorization for EL device. Although the possession of Developer PIN does not allow directly reading data in EL, the disclosure of Developer PIN will critically jeopardize the data security.

User PIN

User PIN is 8-byte long. Its default value is “12345678” (*0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38*). The privileges of User PIN can only conduct invoking operations on EXF but accessing other files such as data file. In principle, there is no must-be-secured requirement on User PIN. It is also simply available to verify whether the possession of the EL device is the releaser or not.

Using PINs

- **Independency**
  Both PINs are designed for securing directories in EL, but not for specific EL device. Each directory contains its own Developer PIN and User PIN. The Read-Write attributes among directories are independent; therefore modifying the PIN of the root directory does not

mean the PIN of sub-directories is modified accordingly. Similarly, the acquisition of root directory authorization does not mean the acquisition of relevant authorization of sub-directories automatically.

This feature allows some irrelevant software can be protected by one EL device without interferences.

- **Retries**

  15 continuous failures of retries on Developer PIN of the root directory will lock the EL device permanently. Once it is locked, there is no way to recover it, so it is recommended to keep Developer PINs well-preserved of EL device. It is to secure the data inside to a maximum extent.

  The failure of retries on Developer PIN of sub-directories will only get the directory itself locked. However it can still be deleted by clearing its upper-level directory.

  User PIN Retries will not lock the device or directory in any case.

- **Restoration**

  Whenever the file system is re-initialized, such as deleting and re-creating a root directory, clearing the sub-directory, the two-level PINs of the directory will be both restored to default values, so do remember to modify the Developer PIN to a secret value after the file system is re-initialized.

Prepare to Use 3

This chapter will use a sample to generally explain how to port codes into EL. It will help you understand roughly the total process of deployment of protection scheme.

The full sample is available under the directory of *SDK\Samples\Bubblesort Demo*.

Demo Software

To make the process easier to understand, we have chosen a simple demo program: bubble-sorting algorithm. Before it is ported, the code of the software is as follows:

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/*bubble sort function*/
void bubble_sort(unsigned char *p, int len)
{
	int i,j;
	unsigned char tmp;

	for (i=0; i<len-1; i++)
	{
		for (j=0; j<len-i-1; j++)
		{
			if (p[j] < p[j+1])
			{
				tmp = p[j];
				p[j] = p[j+1];
				p[j+1] = tmp;
			}
		}
	}
}

/*main procedure*/
void main()
{
	unsigned char test[] = {4,3,8,2,9,7,1,5,0,6};
	int len = sizeof(test);
	int i;

	bubble_sort(test, len);

	printf("result: \n");
	for (i=0; i<len; i++)
	{
		printf("%d ",test[i]);
	}
}
```

Create a new Win32 Console Application in Microsoft Visual C++ 6 and add the above code to the VC6 project. In this sample, suppose we save the project as *C: \s4demo\Win32\demo1.dsp*, and the above code as *C: \s4demo\Win32\pc_demo1.c*. Compile and run the above code and it will output the following result:

```
9 8 7 6 5 4 3 2 1 0
```

Let's suppose the sorting algorithm in the software is a code that shall be protected and that shall be ported into device.

Code Porting

Before the code is ported, you need to choose a compiler. In this sample, Keil C51 v3.0 is selected. This chapter describes only necessary parts for the configuration of the Keil C51 compiler. For more details about the usage of the compiler, please refer to Chapter "Using Tools".

Create a Project

Run Keil µ Vision, select **Project**-> **New Project…** on the top menu bar, name the project as **Demo1.uv 2**, and save. In this sample, suppose the project is saved under *C:\s4demo\Demo1.uv2*.

After that, µ Vision will prompt you to select device. Select **Intel**->**8052AH** from the list of **Generic CPU Data Base**. Keil will ask **Copy Standard 8051 Startup Code to Project Folder and Add File to Project?** Click **Yes**. At this stage, a blank project is successfully built. Some additional configurations shall be made before the project can be supported by EL.

Select **Project**->**Options for Target1** from the top menu bar. On the tab **Target**, set **Memory Model** as **Large: variables in XDATA**, and then switch to the tab **Output**, tick **Create Hex File**.

Compile Codes

Now we can add the ported code to the project. Create a C file called *demo1.c* and copy the *bubble-sort()* function in *pc_demo1.c* to *demo1.c* as well as adding the *main()* function of EL. The full contents of *demo1.c* are given below:

```
#include "ses_v3.h"

/*bubble sort function*/
void bubble_sort(unsigned char *p, int len)
{
	int i,j;
	unsigned char tmp;

	for (i=0; i<len-1; i++)
	{
		for (j=0; j<len-i-1; j++)
		{
			if (p[j] < p[j+1])
			{
				tmp = p[j];
				p[j] = p[j+1];
				p[j+1] = tmp;
			}
		}
	}
}

/*EL main procedure*/
void main()
{
	unsigned char *test = pbInBuff;
	int len = bInLen;

	bubble_sort(test, len);

	_set_response(len,test);
	_exit();
}
```

The above compiled code can be run in EL. We should make a brief description of the code before getting started with compilation.

The function of the code in this section is to submit the data and data length (byte count) received at the computer end to the *bubble_sort ()* function of EL for processing and to return the processing results to the computer.

The *pbInBuff* and *bInLen* are two pre-defined macros. The former points to communication buffer, namely, the data sent by the computer while the latter indicates the length of transmitted data.

The code of *bubble-sort* () function is completely identical with that in *pc_demo1.c*. This is a very important feature of EL: supporting the porting of standard C code.

_set_response () and *_exit ()* are two system functions of the EL. The former is designed to transmit data to computer while the latter returns from the application of EL. For the detailed of those two system functions, please go to Chapter “SES Reference”.

For the convenience, *demo1.c* is saved to the directory of *C: \s4demo*.

Compile and Test Codes

Before modifying the code (*pc_demo1.c*) at the computer end, you can compile and test the codes first.

In the just-created project, choose the menu **Project**->**Components, Environment and Books** and click **Add Files** in the dialog box to add *demo1.c* to the project. In the meantime, you can add “*Ses51L.lib*” (under the demo directory) to the project[2].

Choose the menu **Project**->**Build target** to compile codes. If the compilation is successful, a file (*demo1.hex*) will be generated under the directory of *C: \s4demo*. This file is the result of code compilation.

Now you can test or debug codes. For the details, please refer to Chapter “Developing Codes” and Chapter “Using Compiler”.

Download Codes to Device

Plug a EL device to the computer and run the Development Test Tool (*SDK\Tools\Devtest.exe*). If the device is correctly recognized by the operating system, the tool will display the global unique serial number of the connected device. Click **Reconnect** to view the device information as shown below.

[2] It is recommended to copy *ses51L.lib* and *ses_v3.h* to the working directory, namely *C:\s4demo* in this sample.

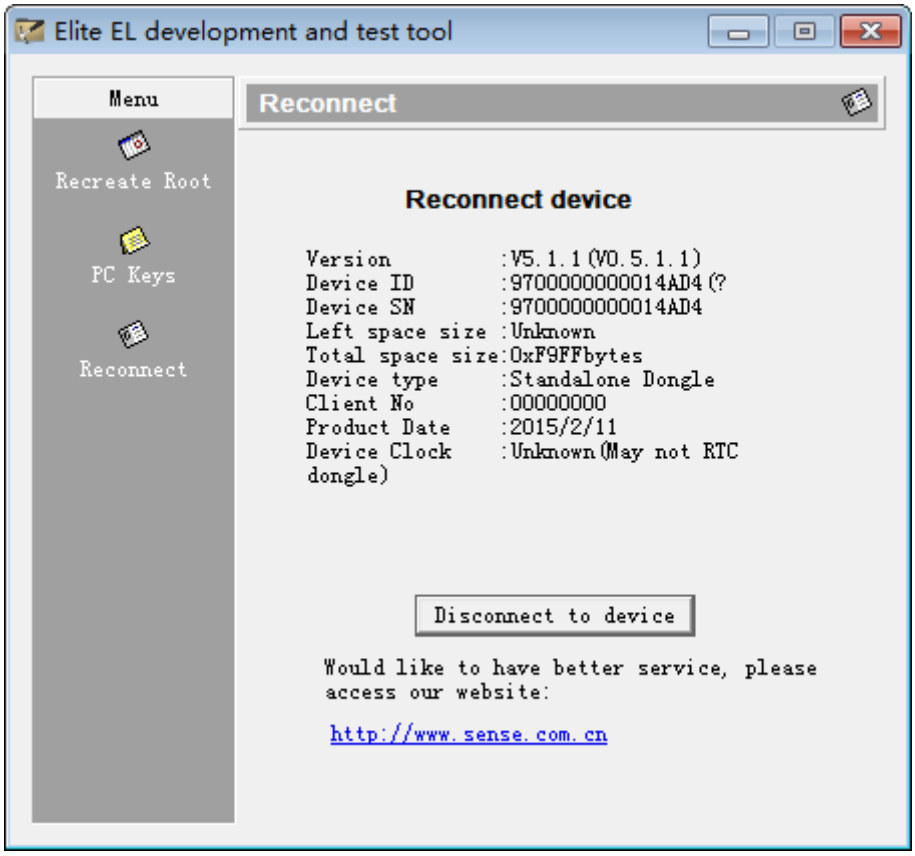


Figure 2: Displaying Device Information

For the convenience of testing, we first re-initialize the device and click **Recreate Root** to create the file system in EL. Assuming that we download the compiled file *demo1.hex* to the root directory, the file ID is *0xd001.* Click **Download** and fill in the file information according to the following.

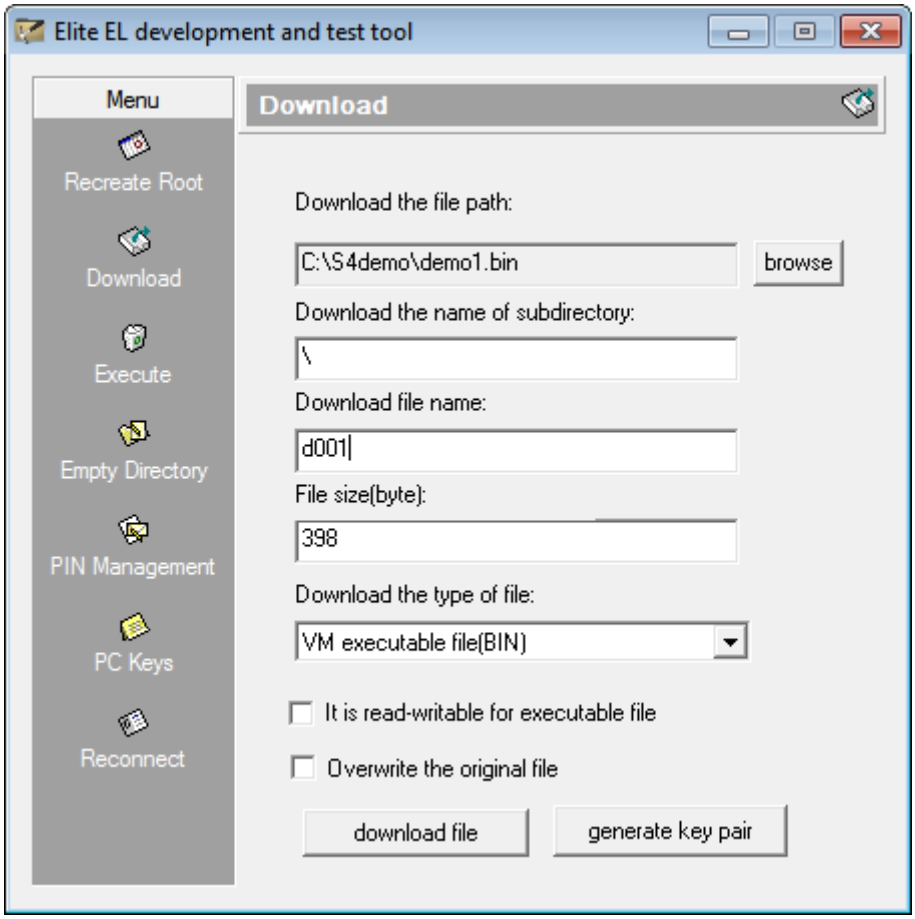


Figure 3: Download File to Device

The software will calculate automatically the size of the file to be created. In this sample, the file shall be at least 398 bytes. You can input a greater value so that when the file grows bigger, the created file can still be used. To write data into the existing file, simply tick the checkbox **Overwrite the original file**.

Modify Computer Codes

Since *bubble_sort()* function is ported to EL, we cancel or delete it in *pc_demo1.c*. In the meantime, we change the part of calling *bubble_sort()* in *main()* function into calling internal codes of device. The contents of the modified *pc_demo1.c* are as follows:

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include “sense4.h”

/*bubble sort function removed. Add EL invoking code. */
void call_sense4(char *, unsigned char *, int);

/*main procedure*/
void main()
{
	unsigned char test[] = {4,3,8,2,9,7,1,5,0,6};
	int len = sizeof(test);
	int i;

	call_sense4(“d001”, test, len);

	printf("result: \n");
	for (i=0; i<len; i++)
```

```
        {
                printf("%d ",test[i]);
        }
}

void call_sense4(char *fid, unsigned char *buff, int len)
{
        /* See remarks for details.*/
}
```

The red, bold text indicates the modification made in contrast with the original codes. There are three parts:

- Add support for access to EL device, including header file (*sense4.h*) and corresponding API function library (*sense4.lib*+ *sense4.dll* or *sense4st.lib*)
- Remove the copy of ported codes in the software and add the call function *call_ sense4 ()* to EL.
- Add the calling of internal codes of EL to fulfill the function of ported codes.

In this sample, the function of *call_ sense4 ()* is to run the EXF whose ID is 0xd001 in EL, taking test and *len* as input data and save the operation result to test. *call_sense4()* is not a universal API function offered in SDK, but an encapsulation of standard API. The purpose of the function is to enhance the readability of the sample codes.

In fact, to execute the EXF in EL, the standard access process should be: Enumerate devices->Open specified device->Choose directory->Verify the PIN->Execute…->Execute->Close the device->Release resources. We do not recommend you to access devices using this function in formal software. For the details and suggested method for device access, please refer to Chapter “API Reference” and Chapter “Developing Codes”.

Copy *sense4.h* and *sense4st.lib* in SDK to the directory of *c: \s4demo\win32*, add it to the project and re-compile the VC6 project.

```
void call_sense4(char *fid, unsigned char *buff, int len)
{
        SENSE4_CONTEXT ctx = {0};
        SENSE4_CONTEXT *pctx = NULL;
        unsigned long size = 0;
        unsigned long ret = 0;

        S4Enum(pctx, &size);
        if (size == 0)
        {
                printf("EL not found!\n");
                return;
        }

        pctx = (SENSE4_CONTEXT *)malloc(size);
        if (pctx == NULL)
        {
                printf("Not enough memory!\n");
                return;
        }

        ret = S4Enum(pctx, &size);
        if (ret != S4_SUCCESS)
        {
                printf("Enumerate EL error!\n");
                free(pctx);
                return;
        }

        memcpy(&ctx, pctx, sizeof(SENSE4_CONTEXT));
        free(pctx);
```

```
        pctx = NULL;

        ret = S4Open(&ctx);
        if (ret != S4_SUCCESS)
        {
                printf("Open EL failed!\n");
                return;
        }

        ret = S4ChangeDir(&ctx, "\\");
        if (ret != S4_SUCCESS)
        {
                printf("No root directory found!\n");
                S4Close(&ctx);
                return;
        }

        ret = S4VerifyPin(&ctx, "12345678", 8, S4_USER_PIN);
        if (ret != S4_SUCCESS)
        {
                printf("Verify User PIN failed!\n");
                S4Close(&ctx);
                return;
        }

        ret = S4Execute(&ctx, fid, buff, len, buff, len, &size);
        if (ret != S4_SUCCESS)
        {
                printf("Execute EL exe failed!\n");
                S4Close(&ctx);
                return;
        }

        S4Close(&ctx);

        return;
}
```

Test Protection Results

After taking the above steps, we have completed the protection of sample codes. Plug the configured EL device, run the re-compiled sample software and you will get the correct output result:

```
9 8 7 6 5 4 3 2 1 0
```

Remove the Elite EL and re-run the sample software and you will get the prompt for error.

```
EL not found!
4 3 8 2 9 7 1 5 0 6
```

Analyze Demo Structure

Before the demo program is protected, the codes contain two major functions: main function *main()* and function *bubble_sort()*. After it is protected, all the codes contain four major functions: main function *main()*, device access function *call_sense4()*, main function *main()* in EL, and function *bubble_sort()* in EL. The following diagram shows difference between unprotected and protected program.

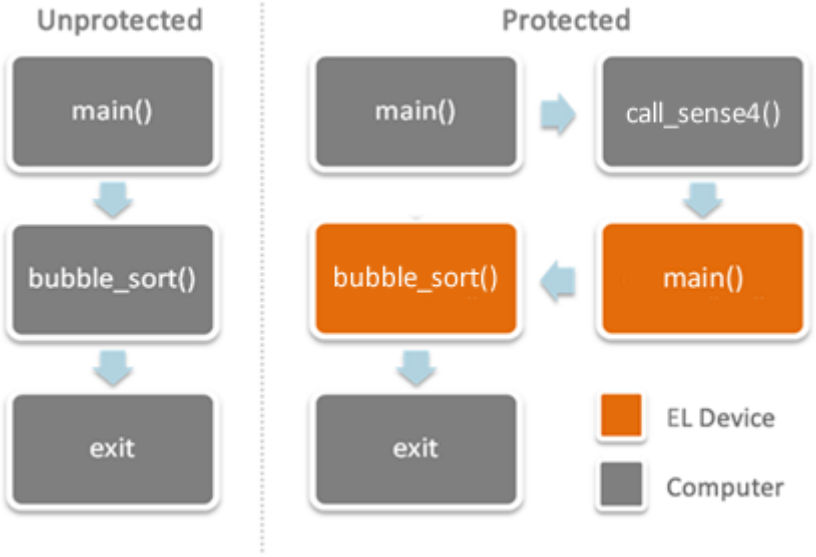


Figure 4: Protection Structure of Demo Program

It can be seen that the protected codes seem to be based on the Client/Server structure. EL is equivalent to a “Service” while functions *call_sense4()* and *main()* in EL plays the role of communication.

Developing Codes 4

From the previous chapters that developing EL applications involves two tasks: compiling internal codes in EL device and compiling the software codes in the computer as well as accessing EL device. This chapter will guide you how to compile and test the internal codes of EL, and operate EL device by API.

Compile Internal Codes

Compiling the internal codes of EL requires basic knowledge of C language. If you have ever developed C programs, it will be much easier for you to go through this section. The C language used for the development of protection scheme is quite simple; it does not require a senior C programmer at all.

Running Modes

Previously stated, there is one type of executable files in EL: VM executable files which correspond to the following running mode:

- **VM Mode (Virtual Machine Mode)**
  Codes compiled by Keil C51 turn into VM files;

  For the convenience of understanding, VM file are taken as internal executable files, termed EXF (Executable File).

  If different compiler/running modes are used, there is a slight difference in the requirements for source codes. For details, please refer to Chapter "Using Compiler". We will present sample codes without being affected by running modes. Notes will be given on the part involving with running modes.

Internal Code Structure

The typical structure of EL internal codes is as follows:

```
#include "ses_v3.h"

void foo()
{
        /* function code here. */
        return;
}

void main()
{
        /* initialization code here. */
        foo();
        _exit();
}
```

A code that is a standard C language program structure consists of a main function *main()*and several sub-functions. The main function is the entry point of a program where codes start to be executed. The description of others is given below:

- **#include "ses_v3.h"**
  ses_v3.h contains the definition of the extension service of EL system. (*SDK\IDE\keil\inc*) You

must include this file in all codes. SES is short for System Extension Service and defined as system functions in this guide, which provide operations on file, clock, input & output, etc. All the SES names are initiated with the underline "_". For description of SES in detail, please refer to Chapter "SES Reference".

- **Program Termination**
  You can add system function *_exit()* to any part of the program to terminate the execution of the program. Notice that the use of *return* sentence in main function cannot guarantee the safe exit of the program. Instead, the *_exit()* system function must be used. This slightly differs from standard C language.

Input & Output

The purpose of compiling internal codes for EL is to provide functional service to the software running on the computer end. The communication between EXF and software is achieved via "Communication Buffer", which is a special memory area in the EL device. The data, sent from the computer to EL or otherwise, will be all saved in the communication buffer.

Two macros are defined in *ses_v3.h* of EL to receive the data from the communication buffer:

- **pbInBuff**
  A byte type pointer pointing to the communication buffer; the other is
- **bInLen**
  A single byte integer, indicating the length of data sent over from the computer.

Due to the limit of internal resources of EL, the communication buffer shall not exceed 250 bytes in size. In other words, the data sent from the computer to EL or otherwise shall not exceed 250 bytes[3] each time. To transmit data larger than 250 bytes requires that the data should be partitioned and transmitted several times. For the implementation method, please refer to "Memory Sharing" in this chapter.

While transmitting data from EL to the computer, the system function *_set_response()* shall be used. It will copy the data automatically to the communication buffer where the computer can receive. It shall be noted that only the last call to *_set_response()* is effective and the previous calls will be disregarded.

The sample as given below illustrates how data are transported between the computer and the communication buffer.

```
#include "ses_v3.h"
#include "string.h"

unsigned char output[256];
unsigned char input[256];

void main()
{
        int input_len = bInLen;
        int output_len = 0;

        memcpy(input, pbInBuff, input_len);

        /* Operate input data here… */

        /* If we get some output data… */
        _set_response((unsigned char)output_len, output);
        _exit();
```

[3]In the current version, when data are transported from EL to the computer, the value can be as big as 252, but there is no guarantee of this value for the subsequent versions.

```
}
```

In this sample, the C function *memcpy()*, which is defined in *string.h* is being used. The communication buffer itself is also a part of the memory, so you can either copy the content in it somewhere else or operate it directly.

Memory Structure

EL offers more than 2K memory space[4], but is still much less than the computer. While developing codes, you need to use it cautiously in case overflow occurs.

The memory structure is a complicated topic in term of the selected compiler. Due to the limited space of context, it is unable to go through all around. If it is hard to understand the memory structure of EL, you can skip this section and directly refer to the end of this Section "Compilation Suggestions".

- **Memory Allocation**

In the VM mode, the memory is divided into two parts: internal and external RAM. The internal RAM has totals 256 bytes while the external RAM is up to 2K bytes. Some compilers may use the internal RAM as stack in compilation results, so under normal circumstances, it is recommended to use the external RAM in case the memory may overflow. In codes, the internal RAM and external RAM are specified respectively by reserved keywords idata and xdata. For example:

```
xdata unsigned long x;
idata unsigned long i;
```

The variable x is defined in the external RAM while the variable i is defined in the internal RAM.

You do not need always to define variables explicitly using those two keywords. The compiler can use the default value. For instance, if the compilation mode is set to large (which is highly recommended by us), all variables without being given a special declaration will be defined in the external RAM. In this case, to define variables in the internal RAM requires using idata explicitly.

- **Memory Address**

The communication buffer is a special memory. Under VM mode, the memory structure is shown as follows:

0 …….. 0xFF 0 …….. 0x7FF 0x8000 0x8001…….. 0x80FA

Internal RAM	External RAM	Data length	Communication buffer

a. VM Mode Memory Structure

Table 2: EL Memory Structure

- **Memory Sharing**

For VM mode, unless EL is reset (for example, the device is out of power or use _mem_set()); Otherwise after EL EXF is executed, the system will not clear automatically the contents of external RAM while the contents of internal RAM will be cleared each time.

Using this feature, the non-auto-erased memory can be taken as "shared memory". This can help share data among different EXFs and buffer data among consecutive calls.

[4] Under different running circumstances (different compilers are chosen), there are slight differences in available memory. For specific values, please refer to Appendix B.

For instance, if a program in EL needs to receive data larger than 250 bytes, it can be done through a process similar to the following sample:

```
#include “ses_v3.h”

typedef struct {
        unsigned short offset;
        unsigned char len;
        unsigned char buff[1];
} IO_PACKAGE;

DEFINE_AT(unsigned char, big_buff[512], 0x400, RAM_EXT);
IO_PACKAGE *input = NULL;

void main()
{
        input = (IO_PACKAGE *)pbInBuff;
        LE16_TO_CC(&input->len);

        if (input->len != 0)
        {
                memcpy(big_buff + input->offset, input->buff, input->len);
                _exit();
        }

        /* now got enough data and you can add operations here… */

        _exit();
}
```

In the codes, two macros are used: DEFINE_AT *and* LE16_TO_CC. The former can define a variable at a specified address of a specified memory area. The latter will be explained later in the Topic “Big-Endian and Little-Endian”; you can ignore it for the moment. In this sample, we have defined a 512-byte array of unsigned char at *0x400* of the external memory. Furthermore, we have also declared a struct where *offset* is designed to specify the location where the data input are stored in the variable *big_buff* this time. *len* indicates the length of the data input this time while *buff* saves the inputted data.

Assume that the 512-byte data would be valid to operate after being transmitted to device memory. The method is to execute the above code four times. At each execution, the following offsets: *0, 128, 256, 384* and corresponding 128-byte data are transmitted in respectively. At the fifth execution, assigning *0* the variable *len,* indicates no additional data needs to be transmitted in. At this point, codes have received 512-byte data and are able to execute the required functions.

- **Code Memory**

Code Memory is not the part of the memory we mentioned above, however it can be operated (Read-only) like common memory in EL. Under some specific circumstances (for example, it is required to store a big consulting table in EL), using Code Memory can bring great convenience.

Code Memory can also be used to store some initial variables, like strings, so as to save the memory usage. For example, for the codes below:

```
code char message[1024] = “This should be a long message…”;
```



None of SES functions can operate Code Memory directly, therefore to operate variables defined in Code Memory by system functions, it is mandatory to use functions like *strcpy()*, *memcpy()*, etc to copy them to the memory area.

- **Alignment Issue**

To transmit data between the computer and EL device by copying structs needs to pay attention on byte alignment issue. In high-level languages, compiler will align structs by default in accordance with the width of machine instruction in order to enhance efficiency. For example, to define the following struct in Visual C++ 6.0:

```
struct {
       char c;
       long l;
} x;
```

The struct takes up 8-byte long. Whereas the same defined struct may occupy 5-byte memory in EL. If data between the computer software and EL are transmitted via same struct, it will cause chaos. In fact, such a problem exists in the sample of memory sharing mentioned above.

The solution is to require the struct defined in computer software should be aligned by 1 byte. For example, in Visual C++ 6.0[5], we can add the compiler directive *#pragma pack (push, 1)* before declaring the struct to force the compiler to align it by 1 byte. Then, after the declaration is made, we add *#pragma pack (pop)* to restore the original setting. For example,

```
#include “windows.h”

#pragma pack(push, 1)

typedef struct _MY_STRUCTURE {
       char c;
       long l;
} MY_STRUCTURE

#pragma pack(pop)
```

- **Big-Endian and Little-Endian**

This topic is about the CPU architecture. First let us explain briefly the meanings of two terms:

Big-Endian stands for “high byte at the low memory address”. In the computer of Big-Endian, the high byte of a variable is in the front;

Little-Endian stands for “low byte at the low memory address”. In the computer of Little-Endian, the low byte of a variable is in the front.

For example, to define the variable *unsigned long x = 0x12345678*, in the computer of Big-Endian, the variable *x* in the memory should be stored like this: *12 34 56 78*; in the computer of Little-Endian, the way of storage is completely reserved: *78 56 34 12*. An error will occur if a value is assigned to the variable on different architected computers via copying memory.

The x86 series computers are based on Little-Endian but Macintosh on Big-Endian. Keil C51 uses Big-Endian. The communication buffer is to copy the memory data directly, so this situation shall be taken into account.

Let us still take unsigned long x = 0x12345678 for example. Suppose the four bytes are transmitted to the codes compiled by Keil C51 via the communication buffer. At this point, the sequence of the bytes in the communication buffer is 78 56 34 12. If using the following code to assign y, the result of y will be *0x78563412*, this obviously falls out of our

[5] There are similar forcing methods in other high-level languages. For example, the packed key word can be used in Delphi. If you do not want to change the setting of the compiler, you can also insert additional member variable in the definition of the struct to achieve the appropriate alignment. For example, in the definition of the struct x, you can insert additional three bytes char padding[3] behind char c.

expectation.

```
…
unsigned long y;
memcpy(&y, pbInBuff, sizeof(unsigned long));
```

The solution to this is to convert the byte sequence. When the data being transmitted via the communication buffer contain multi-byte variables, such as short, long, float and other types, conversion must be conducted either in the computer software or EXF. To achieve better code compatibility, it is recommended to conduct conversion in EXF.

Some pre-defined macros can assist in the conversion of byte sequence. These macros can determine automatically whether there is a need for conversion depending on the compiler type without the necessity of considering what compiler to use. For example, *LE16_TO_CC* can convert 16 bit variable from Little-Endian into the byte sequence used in EL. Other macros accomplish the similar functions. Abbreviation BE stands for Big-Endian while CC stands for C Compiler used by EL.



As the double-precision floating point is not the standard data type supported by the compiler, but a struct (DOUBLE_T) defined by Senselock, this data type is irrelevant to the compiler. No matter what compiler is used, DOUBLE_T is always encoded in Little-Endian.

Lib Files of System Function

This section does not discuss details of system functions. Rather, it explains how to use the library files of the system function in compiling EL codes.

If Keil C51 is used, there are three modes of library files available: *ses51c.lib*, *ses51s.lib* and *ses51l.lib*, which correspond respectively to Compact Model, Small Model and Large Model of compilation modes. It is not recommended using two modes other than Large Model. To compile codes, copy the corresponding library files to the directory where the project is located and include them to the project.

Data Type

The description of part of data types supported by EL is given below:

	char	**i**nt**eger**	**s**hort	**l**ong	**f**loat	**d**ouble
Length (byte)	1	2	2	4	4	8

Compilation Suggestions

Given below are some general proposals in compiling internal codes:

- Use fewer global variables to avoid stack overflow error as a result of over-numbered local variables, especially arrays and structs.
- Use fewer variables as arrays, structs, etc. costing a large amount of memory while passing parameters. Instead, using pointers to avoid possible stack overflow.
- There should not be too many levels of function nesting so as to avoid possible stack overflow.
- Under the VM mode, try to use external memory to avoid memory overflow or stack

insufficiency.

- Pay attention on code alignment and CPU architecture (Big-Endian & Little-Endian) to avoid data communication errors.
- Try to use the system functions to process. For example, using *_mem_copy()* is greater than *memcpy()* in operation efficiency.
- Write comments in English as some compilers do not support Asian texts well.
- Try not to operate strings with functions like *sprintf()*, *sscanf()*, etc. to avoid possible stack overflow.

Accessing Device

In the software, the EL device is accessed via predefined API. There are three levels of accesses: General Level, Developer Level and User level. Except for the General Level, the rest two can only have corresponding access authorizations using valid Developer PIN and User PIN respectively. This section will exemplify the general process of accessing the EL device in the software.

All samples in this section are developed and tested in Visual C++ 6.0.

General Level Access

General access refers to access that does not require security authorization, which involves listing/searching, and connecting device, switching directory, sending control code, closing device, etc. Please refer to the sample below:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "sense4.h"

int main(int argc, char** argv)
{
	SENSE4_CONTEXT s4ctx = {0};  /* current device context */
	SENSE4_CONTEXT *ps4ctx = NULL;  /* for device context list */
	unsigned long ctx_size = 0;  /* size of device context list */
	unsigned long ret = 0;  /* return value */
	unsigned long len = 0;  /* for returned data length. */

	/*step 1:  Enumerate all the EL devices connected. */
	ret = S4Enum(NULL, &ctx_size);
	if (ret != S4_SUCCESS && ret != S4_INSUFICIENT_BUFFER)
	{
		printf("Enumerate EL failed!<error code = %08x>\n", ret);
		return 1;
	}

	if (ctx_size == 0)
	{
		printf("No EL found!\n");
		return 1;
	}

	ps4ctx = (SENSE4_CONTEXT *)malloc(ctx_size);
	ret = S4Enum(ps4ctx, &ctx_size);
	if (ret != S4_SUCCESS) /* Some error occurred. */
	{
		printf("Enumerate EL failed!<error code = %08x>\n", ret);
		free(ps4ctx);
		ps4ctx = NULL;
```

```
                return 1;
        }

        /* step 2:  Open one of the device. For example, the first one. */
        memcpy(&s4ctx, ps4ctx, sizeof(SENSE4_CONTEXT));
        /* Here we can get device's Global Serial Number via s4ctx.bID. */

        free(ps4ctx);   /* Or free it later if you want to access other
devices.*/
        ps4ctx = NULL;

        ret = S4Open(&s4ctx);
        if (ret != S4_SUCCESS)
        {
                printf("Open first EL failed!<error code = %08x>\n", ret);
                return 1;
        }

        /* step 3:  Now we can send some control code to device,
                for example, control the LED. This step is not necessary.
*/
        ret = S4Control(&s4ctx, S4_LED_UP, NULL, 0, NULL, 0, &len);
        if (ret != S4_SUCCESS)
        {
                printf("Light LED failed!<error code = %08x>\n", ret);
                S4Close(&s4ctx);
                return 1;
        }

        /* step 4:  We may change the current directory here. */
        ret = S4ChangeDir(&s4ctx, "\\");  /* Change to root dir. */
        if (ret != S4_SUCCESS)
        {
                printf("Change to root dir failed!<error code = %08x>\n",
ret);
                S4Close(&s4ctx);
                return 1;
        }

        /* step 5:  Do some more actions here... */

        /* step 6:  Close device. */
        S4Close(&s4ctx);


        return 0;
}
```

This sample program describes the general process of accessing EL in 6 steps:

Step	Operation
List device	Call *S4Enum()* function twice in all. At the first time, set the first parameter to NULL and the second parameter to *0*. Then the function will list all valid EL devices in the system and the requested size of device context (the quantity of devices can be calculated this way: *ctx_size*/*sizeof (*SENSE4_CONTEXT*)* Dynamically allocate the needed memory so as to save the context list of all devices or statically allocate a large enough block of memory), call the *S4Enum()* function again which will copy contexts of all devices to the list.

	The device context contains the serial number[6] which could be used to distinguish the device under the circumstance of multiple devices connected.
Open specified device	The device can be opened by passing in the context pointer of the specified device as a parameter to *S4Open()* function.
Send control code	After opening, you can control the device by sending the control code, such as modifying LED status, communication mode, getting hardware serial number, etc.
Change current directory	At the first time of accessing, the current directory is the root directory. To change the current directory to sub-directory requires calling *S4ChangeDir()* function or calling it to switch sub-directory to root directory.
Other operation types	Other security level related accesses such as PIN authentication, file writing, program calling, etc.
Close device	After all operations are completed, close EL devices.

Table 3: Process of General Access

Developer Level Access

Generally speaking, developer level access can be made via tools we offer. If you need to manage your own device or to improve the testing efficiency, you can complete operations via developer level API function.

Developer level access involves creating directory (root directory and sub-directory), deleting (clearing) directory, creating a file, overwriting a file and modifying Developer PIN.

To ease the explanation of codes, we first encapsulate the device connection of the sample code in previous section General Level Access into a function: *OpenDongle()*, which opens the device of specified *index* and returns the device context. The source codes are as follows:

```
unsigned long OpenDongle(SENSE4_CONTEXT *ctx, int index)
{
	SENSE4_CONTEXT *ps4ctx = NULL;
	unsigned long ctx_size = 0;
	unsigned long ret = 0;

	If (ctx == NULL)
	{
		return S4_INVALID_PARAMETER;
	}

	/* Enumerate all the EL devices connected. */
	ret = S4Enum(NULL, &ctx_size);
	if (ret != S4_SUCCESS && ret != S4_INSUFFICIENT_BUFFER)
	{
		return ret;
	}

	if (ctx_size == 0)
	{
		return S4_KEY_REMOVED;
	}
```

[6] Each device is assigned a global unique hardware serial number, but if a user-defined ATR file is created under the root directory, the serial number in the context will be the user-defined serial number. In this case, the real hardware serial number can be only acquired via *S4Control()*. For the details of "User Defined Serial Number", please refer to the description of *S4CreateDir()* function.

```
        ps4ctx = (SENSE4_CONTEXT *)malloc(ctx_size);

        ret = S4Enum(ps4ctx, &ctx_size);
        if (ret != S4_SUCCESS) // Some error occurred.
        {
                free(ps4ctx);
                ps4ctx = NULL;
                return ret;
        }

        if (index > (int)(ctx_size/sizeof(SENSE4_CONTEXT) - 1))
        {
                return S4_KEY_REMOVED;
        }

        memcpy(ctx, ps4ctx+index, sizeof(SENSE4_CONTEXT));
        free(ps4ctx);
        ps4ctx = NULL;

        ret = S4Open(ctx);

        return ret;
}
```

The sample codes given below demonstrate how to re-initialize an EL device, create and write an EXF and a data file, modify Developer PIN of the root directory. This is also a general process of initializing device.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "sense4.h"
#include "Psense4.h"

int main(int argc, char **argv)
{
        SENSE4_CONTEXT s4ctx = {0};
        unsigned char fid_exe[] = "d001";  // exe file ID
        unsigned long exe_size = 2048;  // create file size
        unsigned char fid_dat[] = "d002";  // data file ID
        unsigned long dat_size = 1024;  // create file size
        char exe_path[] = "c: \\s4demo\\demo1.hex";   // exf file path in
disk
        char dat_path[] = "c: \\s4demo\\data1.dat";  // data file path in
disk
        unsigned char default_dev_pin[] = "12345678123456781234567812345678";
        unsigned char old_dev_pin[] = "12345678123456781234567812345678";
        unsigned char new_dev_pin[] = "87654321876543218765432187654321";
        unsigned long len = 0;
        unsigned ret = 0;

        /* Open first EL if exists. */
        ret = OpenDongle(&s4ctx, 0);
        if (ret != S4_SUCCESS)
        {
                printf("Open EL failed! <error code = 0x%08x>\n", ret);
                return 1;
        }

        /* Check whether a root dir exists. */
        ret = S4ChangeDir(&s4ctx, "\\");
        if (ret != S4_FILE_NOT_FOUND && ret != S4_SUCCESS)
        {
                printf("Change   to   root   dir   failed!   <error   code   =
0x%08x>\n", ret);
                S4Close(&s4ctx);
                return 1;
        }
```

```
	/* If a root dir exists. */
	if (ret != S4_FILE_NOT_FOUND)
	{
		/* Verify Developer PIN to get full access privilege. */
		ret = S4VerifyPin(&s4ctx, old_dev_pin, 24, S4_DEV_PIN);
		if (ret != S4_SUCCESS)
		{
			printf("Verify dev PIN failed! <error code =
0x%08x>\n", ret);
			S4Close(&s4ctx);
			return 1;
		}

		/* Delete old root dir. */
		ret = S4EraseDir(&s4ctx, NULL);
		if (ret != S4_SUCCESS)
		{
			printf("Delete root dir failed! <error code =
0x%08x>\n", ret);
			S4Close(&s4ctx);
			return 1;
		}
	}

	/* Create new root dir. */
	ret = S4CreateDir(&s4ctx, "\\", 0, S4_CREATE_ROOT_DIR);
	if (ret !=  S4_SUCCESS)
	{
		printf("Create new root failed! <error code = 0x%08x>\n",
ret);
		S4Close(&s4ctx);
		return 1;
	}

	/* Verify Developer PIN to get full access privilege. */
	ret = S4VerifyPin(&s4ctx, default_dev_pin, 24, S4_DEV_PIN);
	if (ret != S4_SUCCESS)
	{
		printf("Verify dev PIN failed! <error code = 0x%08x>\n",
ret);
		S4Close(&s4ctx);
		return 1;
	}

	/* Write disk file to EL. */
	ret = PS4WriteFile(&s4ctx,
		fid_exe, exe_path, &exe_size,
		S4_CREATE_NEW, S4_HEX_FILE,
		&len);
	if (ret != S4_SUCCESS)
	{
		printf("Write exe file failed! <error code = 0x%08x>\n",
ret);
		S4Close(&s4ctx);
		return 1;
	}

	ret = PS4WriteFile(&s4ctx,
		fid_dat, dat_path, &dat_size,
		S4_CREATE_NEW, S4_DATA_FILE,
		&len);
	if (ret != S4_SUCCESS)
	{
		printf("Write data file failed! <error code = 0x%08x>\n",
ret);
		S4Close(&s4ctx);
		return 1;
	}
```

```
        /* Change Developer PIN. */
        ret = S4ChangePin(&s4ctx,
                          default_dev_pin, 24,
                          new_dev_pin, 24,
                          S4_DEV_PIN);
        if (ret != S4_SUCCESS)
        {
                printf("Change dev PIN failed! <error code = 0x%08x>\n",
ret);
                S4Close(&s4ctx);
                return 1;
        }

        /* Close EL. */
        S4Close(&s4ctx);

        /* everything done! */
        printf("Congratulations!\n");

        return 0;
}
```

The basic process of Developer Level Access is: open device->switch directory->verify Developer PIN->developer level operation->close device. As codes of developer level access may contain Developer PIN in form of plan text, it is essential make sure to keep confidentiality of the developer level access codes.

For the detailed description of API involved in the codes, please refer to Chapter "API Reference".



Under whatever condition, do not include developer level access codes in the software to be released. Please change the Developer PIN before releasing the dongle.

User Level Access

Except for the basic operations, there is actually only one access in user level: executing internal EXF.

In the sample codes as given below, we continue to use the *OpenDongle()* function as mentioned above.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "sense4.h"
#include "psense4.h"

int main(int argc, char **argv)
{
        SENSE4_CONTEXT s4ctx = {0};
        unsigned char fid_exe[] = "d001";  // exe file ID
        unsigned char user_pin[] = "12345678";
        unsigned long len = 0;
        unsigned ret = 0;
        unsigned char input[256] = {0};
        unsigned long input_len = 128;
        unsigned char output[256] = {0};
        unsigned long output_len = 251;

        /* Open first EL if exists. */
        ret = OpenDongle(&s4ctx, 0);
        if (ret != S4_SUCCESS)
```

```
        {
                printf("Open EL failed! <error code = 0x%08x>\n", ret);
                return 1;
        }

        /* Change to root dir if exists. */
        ret = S4ChangeDir(&s4ctx, "\\");
        if (ret != S4_FILE_NOT_FOUND && ret != S4_SUCCESS)
        {
                printf("Change to root dir failed! <error code =
0x%08x>\n", ret);
                S4Close(&s4ctx);
                return 1;
        }

        /* Verify User PIN to get invoking privilege. */
        ret = S4VerifyPin(&s4ctx, user_pin, 8, S4_USER_PIN);
        if (ret != S4_SUCCESS)
        {
                printf("Verify User PIN failed! <error code = 0x%08x>\n",
ret);
                S4Close(&s4ctx);
                return 1;
        }

        /* Invoke exf 0xd001. */
        ret = S4Execute(&s4ctx, “d001”,
                                    input, input_len, output, output_len,
&len);

        if (ret != S4_SUCCESS)
        {
                printf("Invoke 0xd001 failed! <error code = 0x%08x>\n",
ret);
                S4Close(&s4ctx);
                return 1;
        }

        /* Close EL. */
        S4Close(&s4ctx);

        /* everything done! */
        printf("Congratulations!\n");

        return 0;
}
```



During development, do mixing or encrypting communication data in order to get higher randomness to resist the malicious decryption analysis.

Testing Codes

From the perspective of testing internal codes, there are two ways:

- **Hardware Level Test**
  To download compiled codes directly to the device and then call and check the running result.

- **Software Emulation Test**
  With the simulator offered, code tests can be easily performed. The advantage is that, besides test, codes could be traced and debugged. The disadvantage is that some extra configuration is required.

From the perspective of invoking device, there are two ways:

- **Using the Development Test Tool**
- **Write High-level Language Testing Codes**

The process of software emulation test is up to the compiler. For detailed usage, see Chapter “Using Compiler”. For tool software, see Chapter “Using Tools”.

Using Compiler 5

This chapter will give a description of the usage and precautions for Keil 51. For the installation method of Keil C51, please refer to its relevant instruction.

The following table shows the features of Keil 51.

	Keil C51
Code Mode	VM Mode
Code bit	8 bits
Byte sequence	BE(Big-Endian)
Optimization Efficiency	High
Memory Structure	Internal RAM + External RAM
Notes	Commercial software. Special attention shall be given to the issue of byte sequence.

Table 4: Compiler Software

As standard C programming is required, whatever compiler is being used, it will only have some impact on development conventions, but not on the codes. By using the macros defined in *ses_v3.h* effectively, you can compose compiler-independent codes.



The compilation result is HEX file in text format which cannot be run directly in EL and needs to convert into binary codes. The *Hexbin.exe* (*SDK**Tools*) is designed to implement this task. For details, please refer to Section “Executable File Format”.

Using Keil C51

Keil C51 is a suite of excellent compilation software developed by Keil. Its IDE called µVision. You can download the evaluation version from the official website of Keil: http://www.keil.com. For limitation in the evaluation version, please refer to the instruction on the website.

C51 manage codes by using “project”, and a typical project usually includes project file **.uv2*, several C code files (including header files), SES header file *ses_v3.h* and SES library file *ses51?.lib*[7].

For conveniences of configuring C51 compilation environment, a patch (*Wizard.exe* in the directory of *SDK**IDE**keil*) is provided to automatically process functions as copying library file, debugger setup, etc.

The demo uses C51 v7.01 to illustrate how to use. The general using scenario will not be changed in the further version.

[7] ‘?’ indicates S, C or L, corresponding to three compilation modes: Small, Compact and Large. Large is highly recommended.

Project Creation and Management

The process of creating a project goes like following. (There are some slight differences in operation process if µVision in different version is being used. In this case, µVision 3 is chosen as a sample):

- **Create a Project**

5. Start the software. Click the menu bar **Project**->**New Project…** and save to the disk;
6. Select **Intel**->**8052AH** from the list box **Data Base** of popped out dialog box. For an existing project, click **Project**->**Select Device for Target 'Target1'**to open the dialog box.

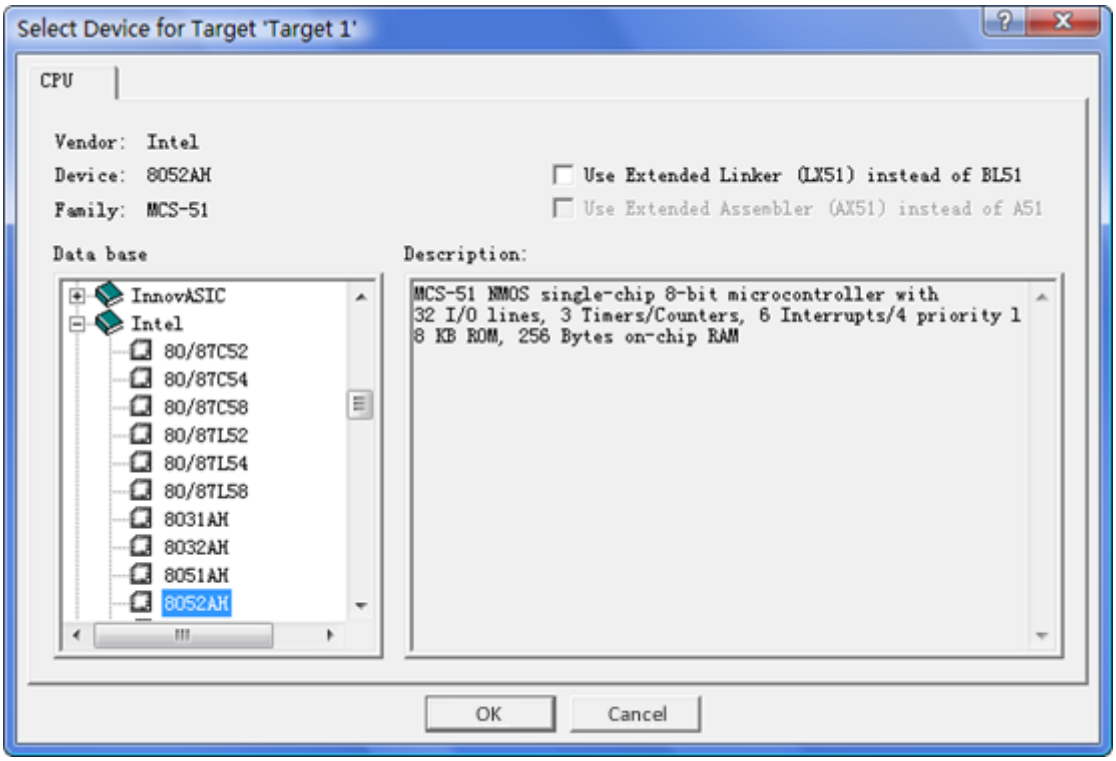


Figure 5: Select CPU Type

If the patch is installed already to configure the Keil development environment, a device database file needs to be chosen:

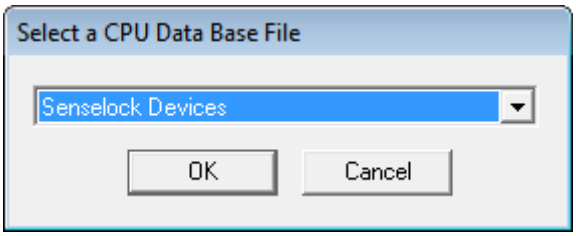


Figure 6: Select a Device Database File

7. Select **Senselock Devices** and, and Click **OK** to pop the dialog box of **Select Device for Target 'Target1'**. The field **Description** displays basic information of the hardware device and warns you to set Memory Model to Large.

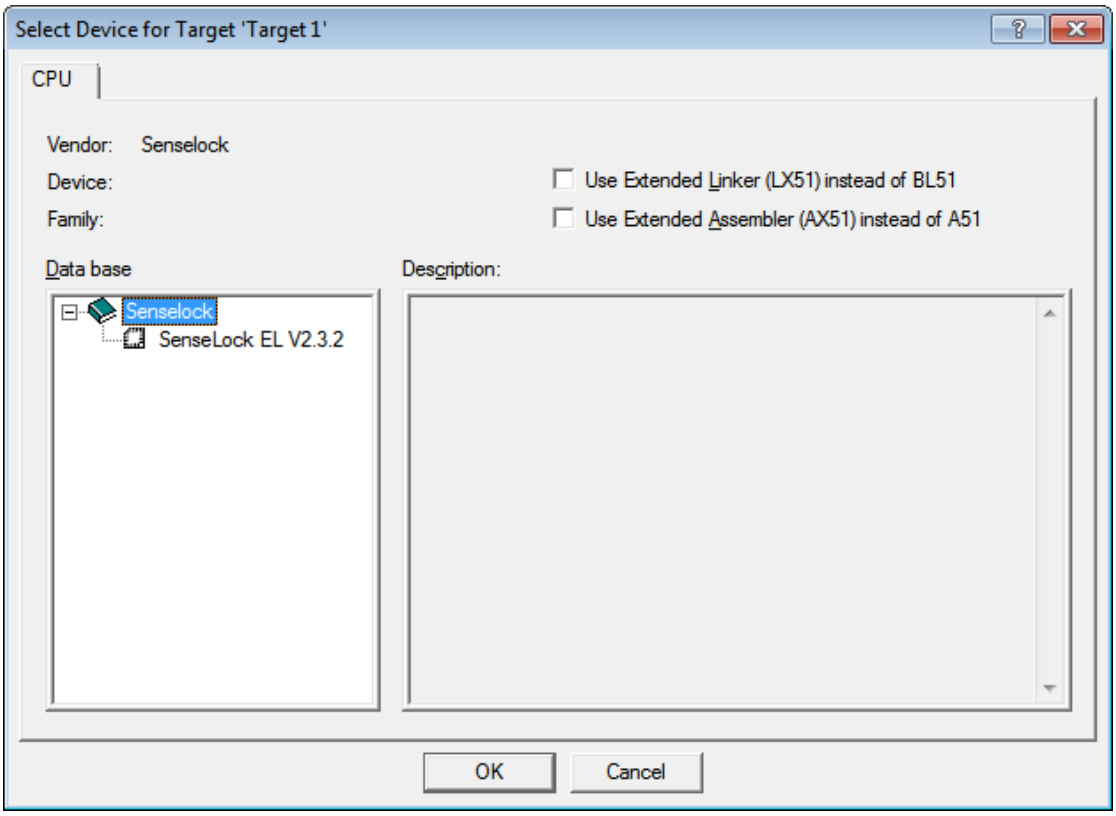


Figure 7: Select a Device

Select **Senselock->SenseLock EL V2.3.2** from the list box **Data base**, and Click **OK**.

- **Set Project Options**

8. Click P**roject**->**Options for Target 'Target 1'**, and go to the tab Target, select Large: variables in XDARA from the drop down box **Memory Model**:

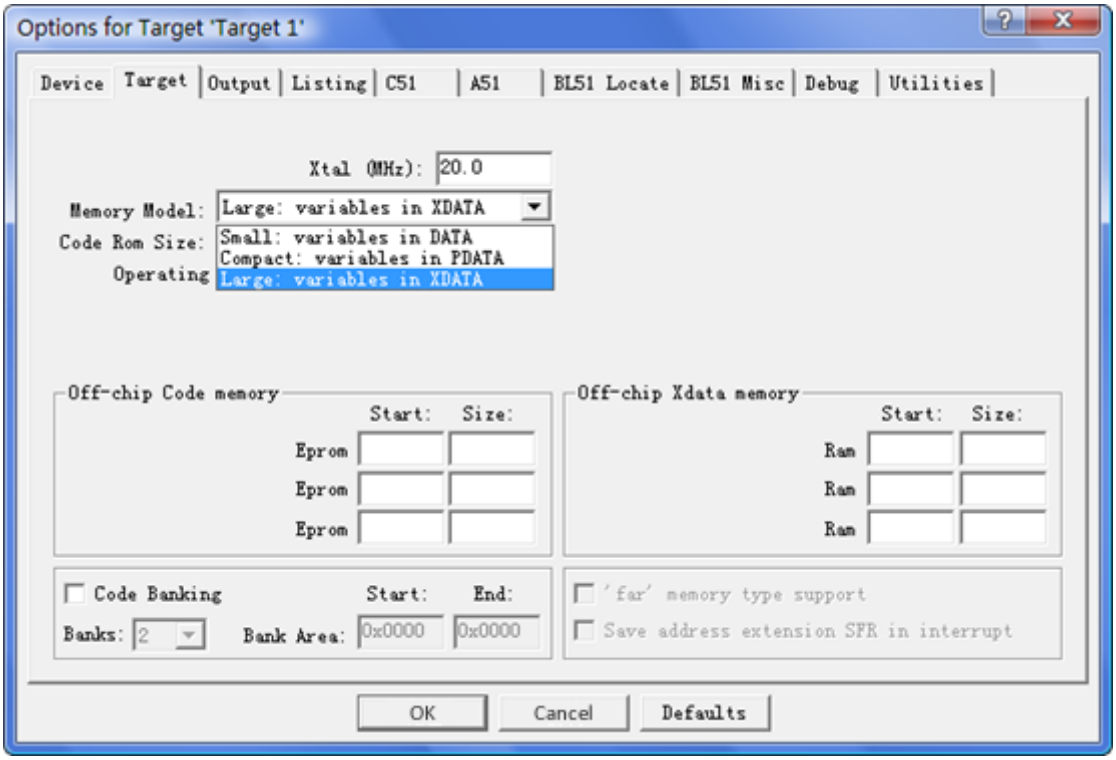


Figure 8: Select a Memory Model

9. Then go to the tab **Output**, tick the checkbox **Create HEX File** and only this way, the target file in hex format can be correctly generated. Close the project option dialog box:

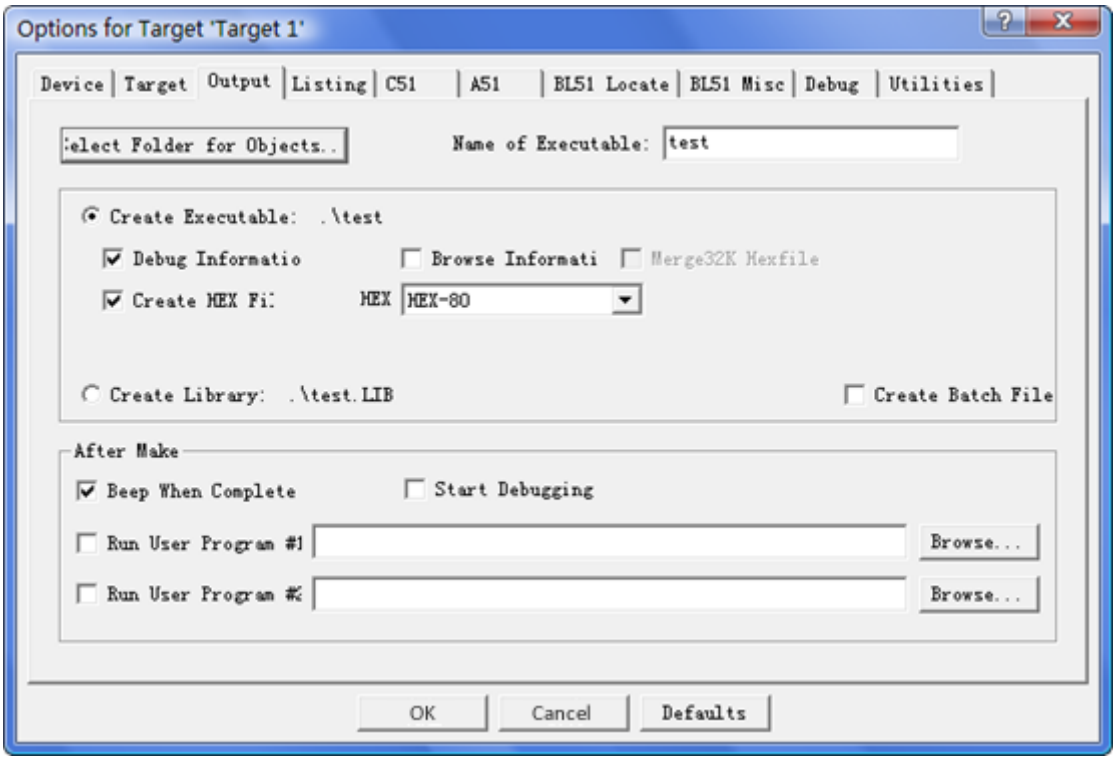


Figure 9: Output HEX File

- **Add the Header File**

10. Copy *ses_v3.h* to the current project directory or to the C51 installed directory (…*Keil**C51**INC);* missing this step will cause a failure of finding the header file *ses_v3.h*(*SDK**IDE**keil*\ *inc*\) in compiling process*;*

- **Add the SES Lib File**

11. Click **Project**->**Components, Environment, Books…** to add SES lib file;

    Depending on the selection of Memory Model, add the corresponding SES library file to the project. For instance, if **Memory Model** is set to Large (highly recommended), add *ses51L.lib* (*SDK**IDE**keil**lib*).

- **Add your C Source Codes**
- After the project is successfully created and configured, you can modify the options of the project at any time.

Code Compilation and Debugging Configuration

- **Compile Codes**

  Press F7 key, or Click the menu bar **Project**->**Build target** to compile. If the compilation is successful executed, a HEX file will be generated accordingly in the project directory; otherwise an error will occur.

- **Configure Simulator**

  Using the software simulator we offered in Keil C51, It can trace and debug codes that require modifying the configuration. You can either use the patch (*Wizard.exe, SDK**IDE**keil*) to complete modification automatically or manually as follows:

12. Copy *S4Simulator.dll* and *vfsView.exe* from (*SDK**IDE**keil**bin*\) to the directory (…*Keil**C51**BIN*\);

13. Open the configuration file (…*keil**Tools.ini*) and add the following to the section **[C51]**: **TDRV0=BIN\S4Simulator.dll("SenseLock EL Simulator")**. If TDRV0 is already being used, change the last digit 0 to the following 1, and so on.

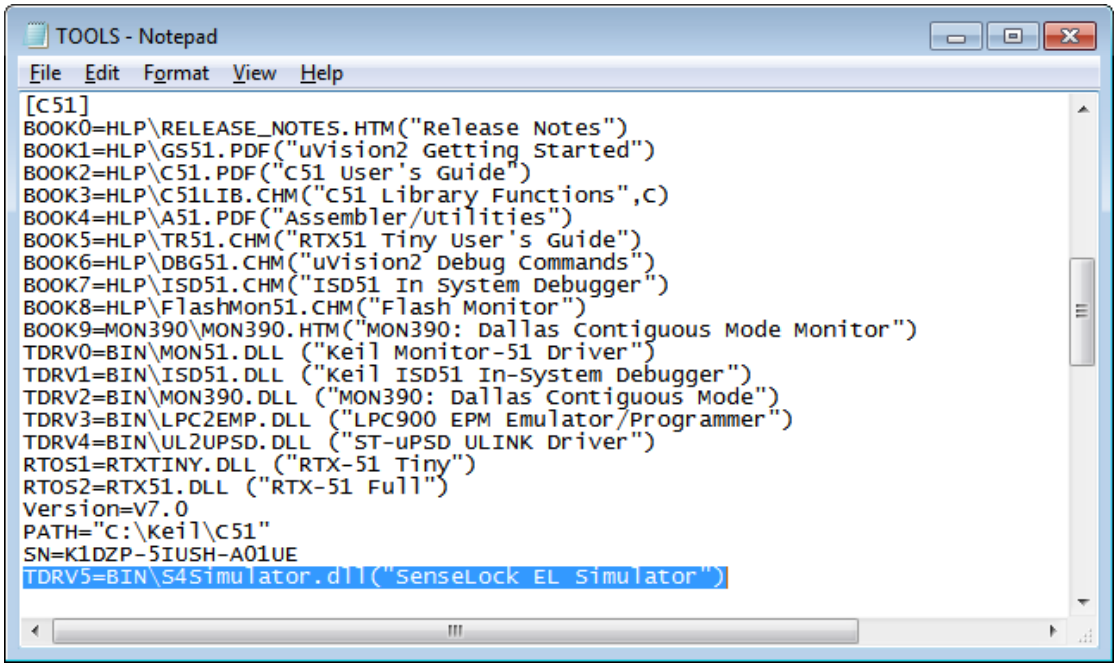


Figure 10: Keil C51 Simulator Config File

14. Start Keil µVision to set debug options, click **Project**->**Options for Target 'Target 1'** and go to the tab **Debug** to select **Senselock EL Simulator** from the drop down box **Use:** on the upper right:

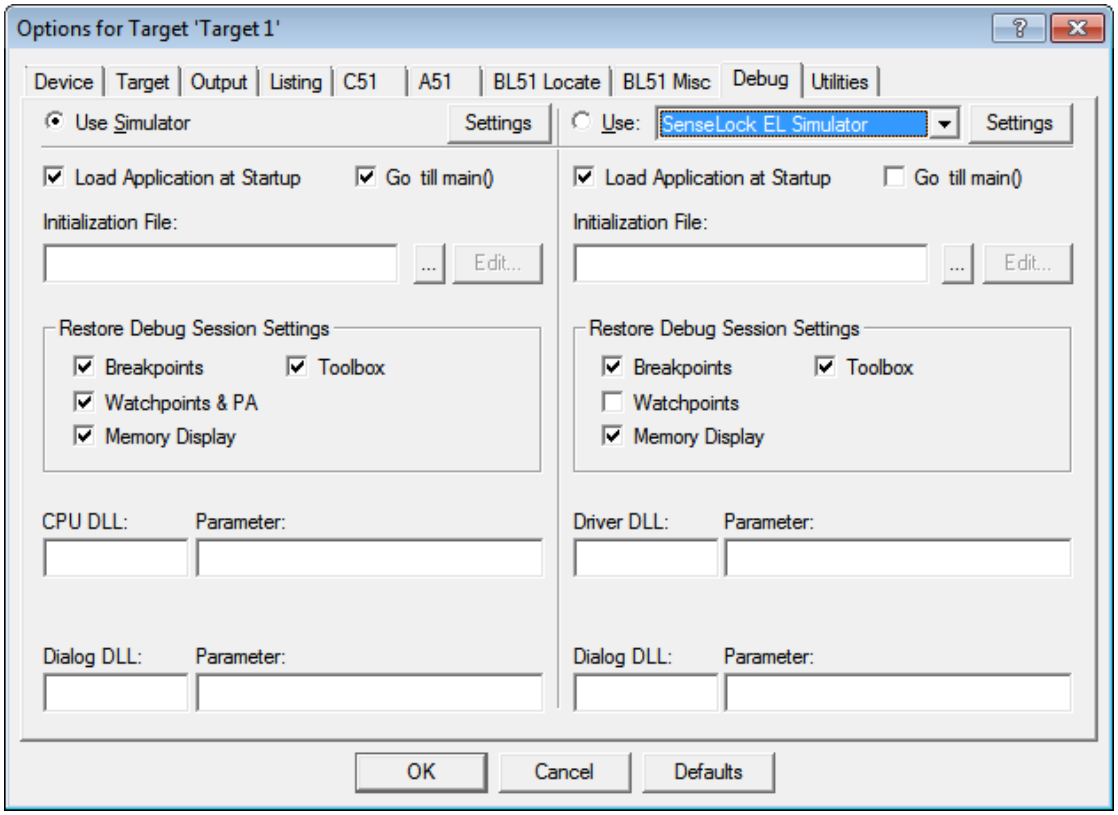


Figure 11: Debug Options

15. Click **OK** to exit the dialog box. Now you can debug codes in the project.

Debugging Codes

- **Initiate Debug Status**

16. Click the menu bar **Debug**->**Start/Stop Debug Session** or Press Ctrl+F5. If the simulator is properly installed and configured, an input window will be popped:

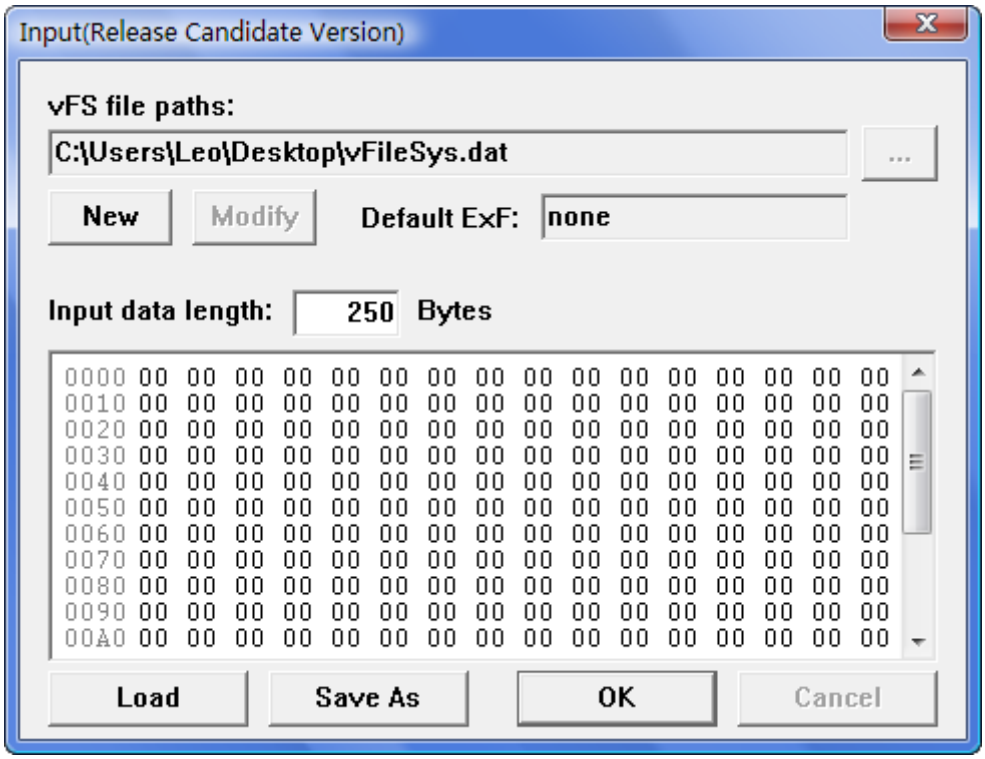


Figure 12: Data Input Window

There are two parts of data input window. The upper half manages the simulation file system while the lower half sets the data to be transmitted to the communication buffer of EL device.

- **Setup Simulate File System**

EL file system is simulated using the file (*vFileSys.dat*) stored in the current project directory.

17. Click the button **New** in the data input window to create a new simulated file system. It is not necessary to create a simulated file system for EXF that does not call the service of file system.

    Click the button **Modify** in the data input window to modify the simulated file system.

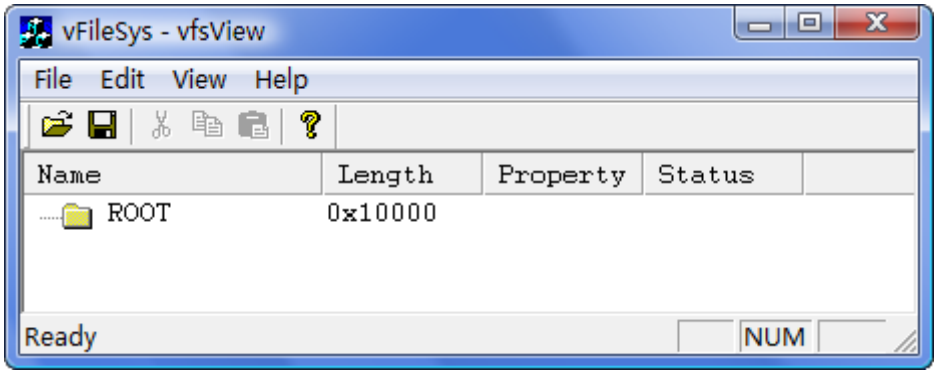


Figure 13: Simulated File System Tool

The simulated file system takes the directory named **ROOT** as a root node (namely, MF of EL) as a root node and displays the information by a tree structure. As designed for debugging codes, the simulated file system only supports one-level directory structure that means no sub-directory can be created.

- **Create a Simulated File**

  The simulated file system supports four file types: Executable file (EXF), Data file (DAT), RSA Public Key (136 bytes by default) and Private Key (330 bytes by default). The read-write attributes of internal readable/writable files should be set "R/W" referring to Section "Read-Write Attributes".

18. Right click the directory **ROOT** to get the shortcut menu, then Click **New File** to pop out a dialog box **Create New File**.

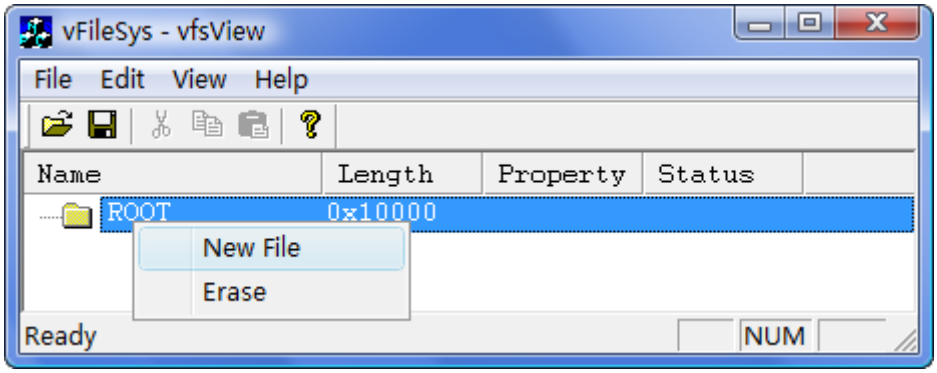


Figure 14: Create a New File

Click **OK** to create a new file in the current directory after inserting defined values.

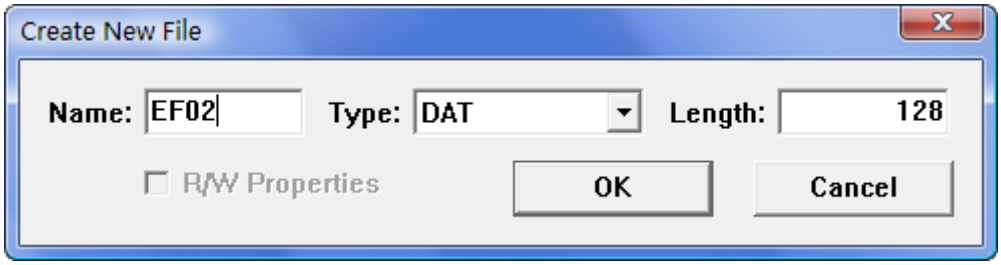


Figure 15: Define the File Property

- **Edit a Simulated File**

  Double click or Right click the file node and click the **Edit** of shortcut menu.

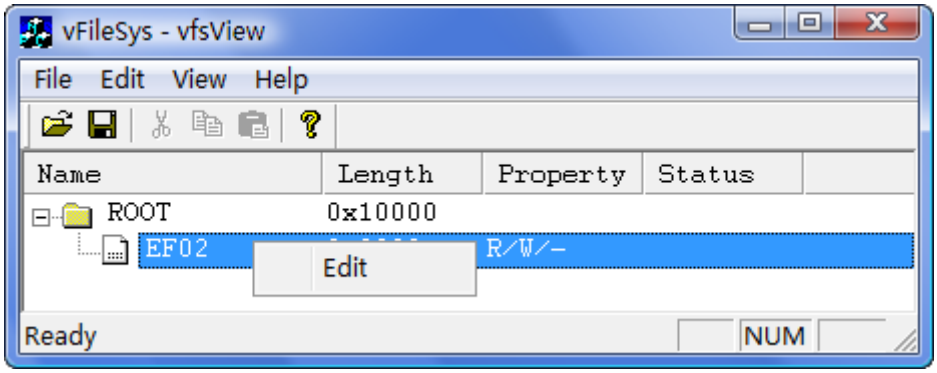


Figure 16: Edit a Simulated File

You can modify the hexadecimal data directly when editing a file or import data from a disk file or save the data to a disk file.

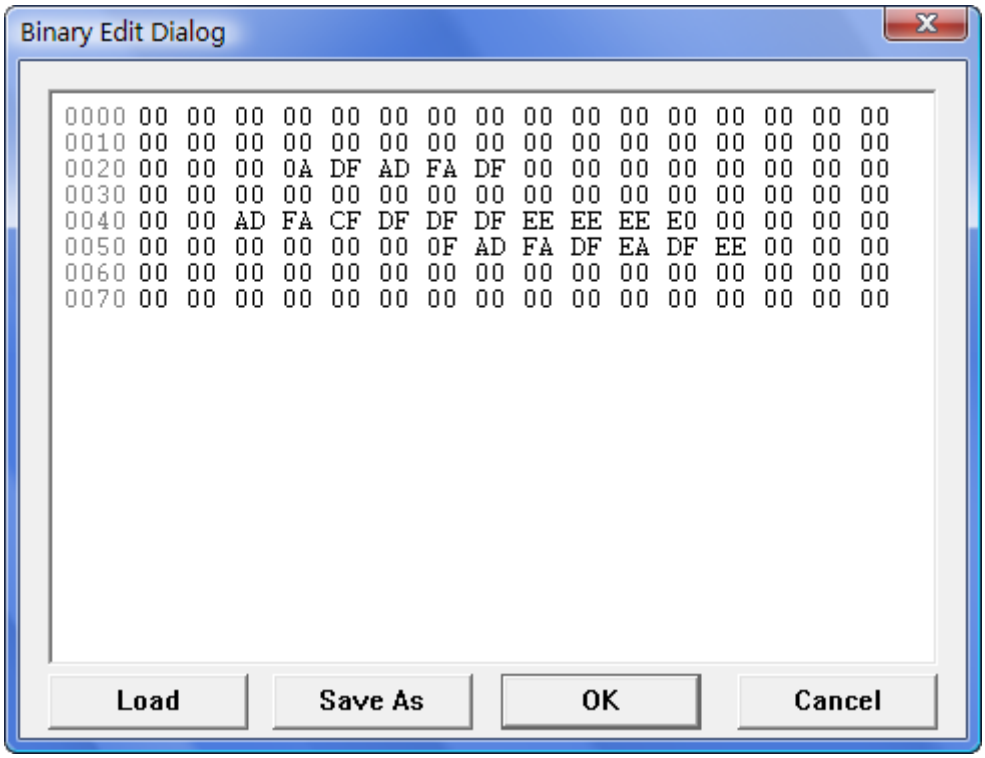


Figure 17: Edit Binary Value

- **Internal Communication**

Sometimes the EXF needs to operate on itself. For example, the SES function *_create()* can acquire the information of the current EXF. As the current EXF is not saved in the simulated file system but being debugged in the Keil compilation environment, this kind of self operation will be failed. The solution is to set the EXF as "default file" in the simulated file system (Right click the EXF to pop a shortcut menu and Click **Set to Default**). In this way, while the EXF needs to operate itself, it will take the default file as the current EXF to operate.

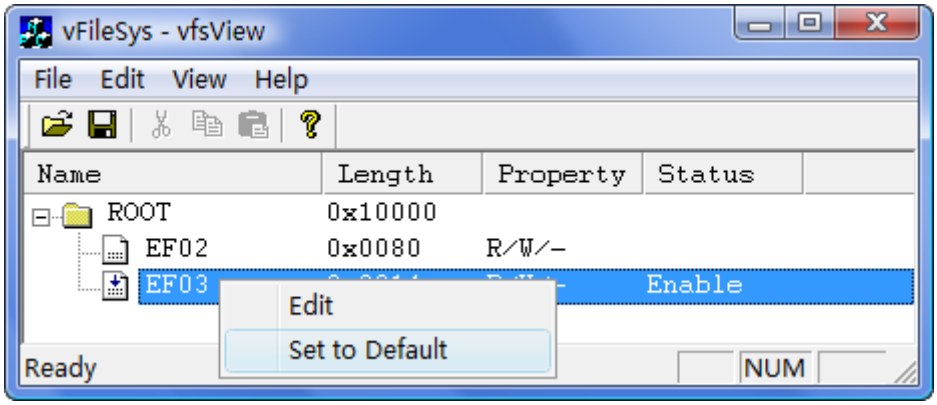


Figure 18: Set Default File

- **External Communication**

Like the real hardware device, the software simulator simulates the communication buffer so as to exchange data with the outside. The data in the input data edit field will be copied to the communication buffer inside the software simulator for being used by EXF. The insertion of hexadecimal data or formatted data is allowed directly.

19. Right click an address to input data in a variety of types:

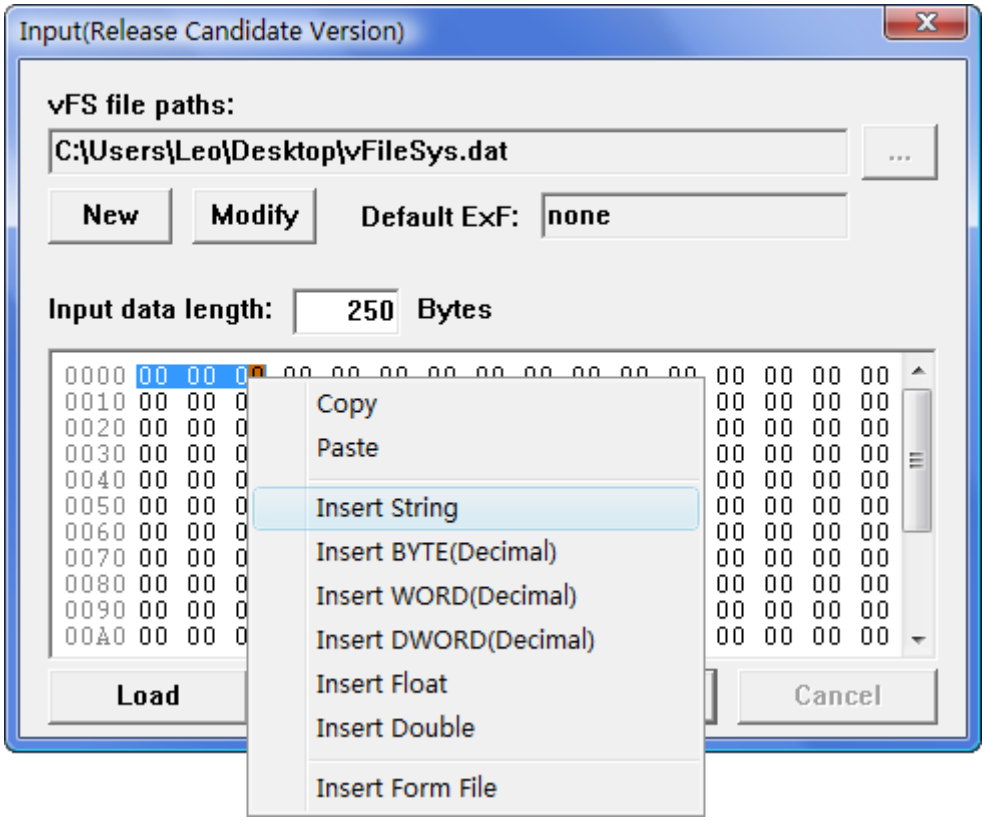


Figure 19: Data Input Demo

- **Get Into Debug Status**
  After setting the file system and the data to be transmitted to device, Click **OK** in the dialog box to enter the Keil debug status. For any detail of debugging, please refer to the instruction manual of Keil C51 or visit http: //www.keil.com.

- **Execution Error**
  If a dialog box prompting execution error pops out in debugging, besides checking error codes, it is essential to check the tab **Regs** of **Project Workspace** window (**View**->**Project Window**):

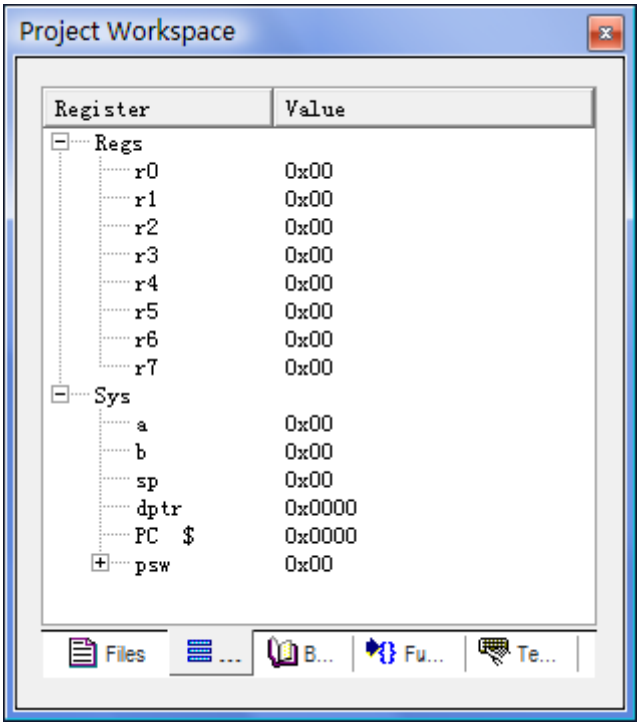


Figure 20: Project Window

The **Regs** displays the current CPU register list: **R0~R7** (Universal Registers), **R0~R1** (IRAM, Indirect Addressing Registers of Internal Memory), **dptr** (XRAM, Indirect Addressing Registers of External Memory), **sp** (Stack Pointer), **PC** (Program Pointer).

The following errors are likely to meet: **dptr** out of range (valid between *0x0000~0x08ff*), SP overflow (valid between *0x07~0xff*, increasing upward) or PC out of range (exceeding the valid range of codes).

- **Exit Debug Status**

20. Click the menu **Debug**->**Start/Stop Debug Session** or Press Ctrl+F5 to exit the debug status.

If the EL program exits the debugging status by executing the SES function *_exit()*, the execution result (the data returned from EL device) will be shown up in a dialog box that is the output data of last calling SES function *_set_response()*, as same as operating on the real device.

Executable File Format

The compilation result is a HEX file in text format. Before execution in EL, it is required to convert it into binary format, abbreviated as BIN file.

For the convenience of conversion, SDK offers tool software: *hexbin.exe*. The usage of this software is like this:

hexbin.exe [file.hex] [file.bin] i 1

The API and the Development Test Tool of the current version can process HEX files directly. For example, when the *S4WriteFile()* function is used to write the EXF into EL, you can input directly the HEX file and specify the file type as *S4_HEX_FILE* (VM executable file). In this case, API will conduct automatically the conversion from HEX to BIN.

Using Tools

6

To make it easy to test the device and develop application programs, the directory *SDK/Tools* contains tools as follows.

Tool	**Application**	File **N**ame	Hardware **V**ision
Development Test Tool	It carries out developer level management and user level device calling.	*DevTest*.exe	v2.0 or above
User Test Tool	It implements basic tests on devices, designed to retrieve device property and to check the running status.	*UserTest*.exe	v2.0 or above
HEX Converter	It converts HEX file into binary file.	*Hexbin*.exe	

Table 5: Introduction to Tools



The device was being initialized by the current version of tools cannot be operated by previous version of tools, and vice versa.

Development and Test Tool

Device Reconnection

When the tool is started, it will search automatically devices connected. If no device is found, the software will give a warning. Click **Reconnect** to display hardware information as follows:

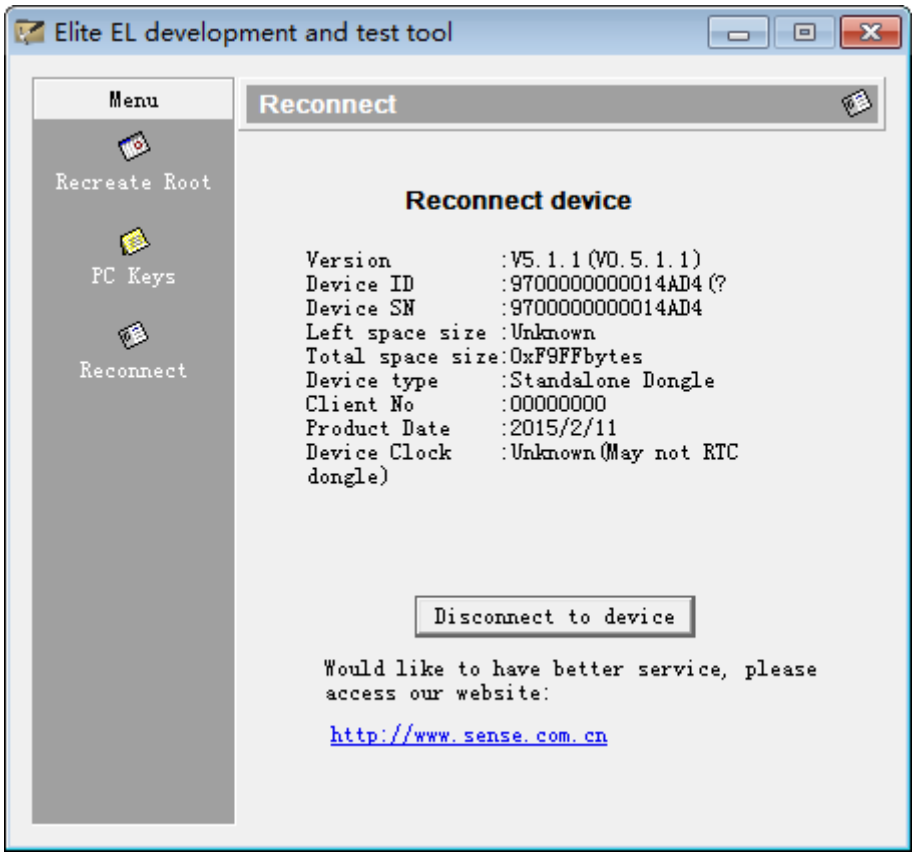


Figure 21: Display Device Property

The tool software accesses the device in “shared mode”, so other software can still access the device at the same time. Click **Disconnect** to switch off the connection between the tool software and the device.

Device Reset

The most fundamental directory management is to create or delete the root directory. In the software, click **Recreate Root** to perform root-directory-related operations: creating root directory, deleting root directory and create new root directory (by default).

As in the above figure, no root directory has been yet created in the device. A new root directory will be created, after input a Device ID and reset the device as follows.

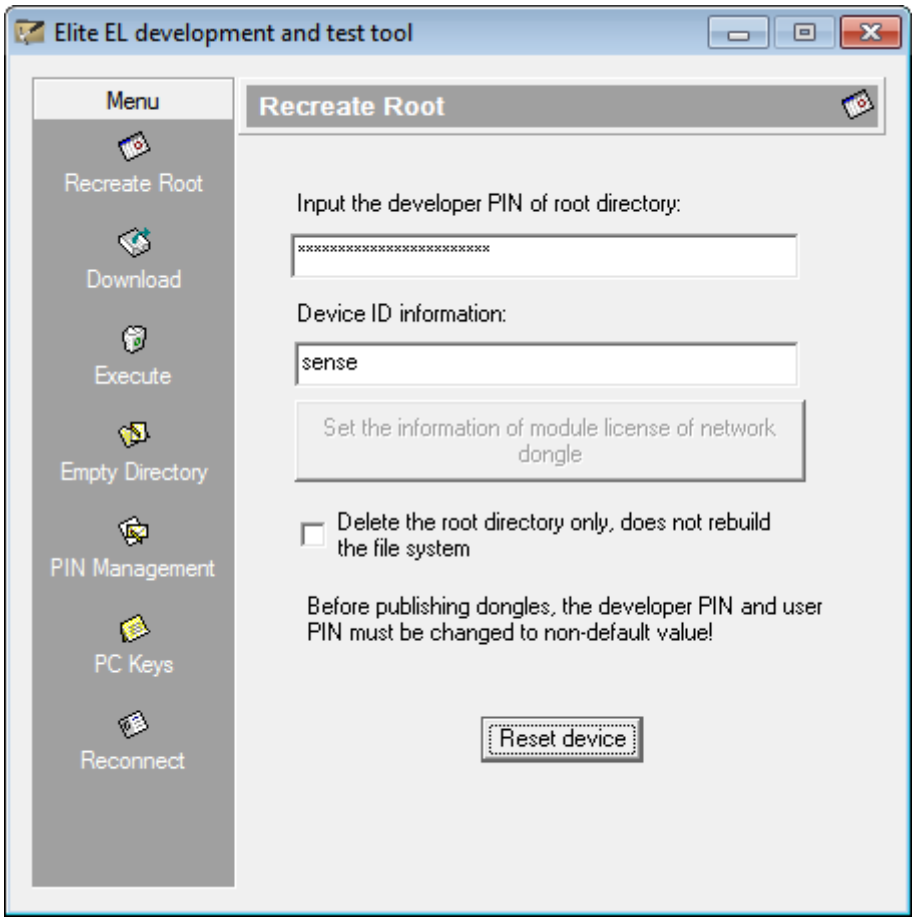


Figure 22: Reset Device with Device ID

For a device having the root directory already, if you need to recreate the root directory, it is required to input its valid Developer PIN. If you need to delete the root directory directly, it is required to tick the check box **Delete the root directory only, does not rebuild the file system**.

After a directory is created/deleted/recreated/cleared, it’s Developer PIN and User PIN will restore to default. For the detailed description of default PIN value and security, please refer to the Section “PIN”.

Only one level of sub-directory is supported in DevTest. To use two levels of sub-directory, you need to write programs yourself.

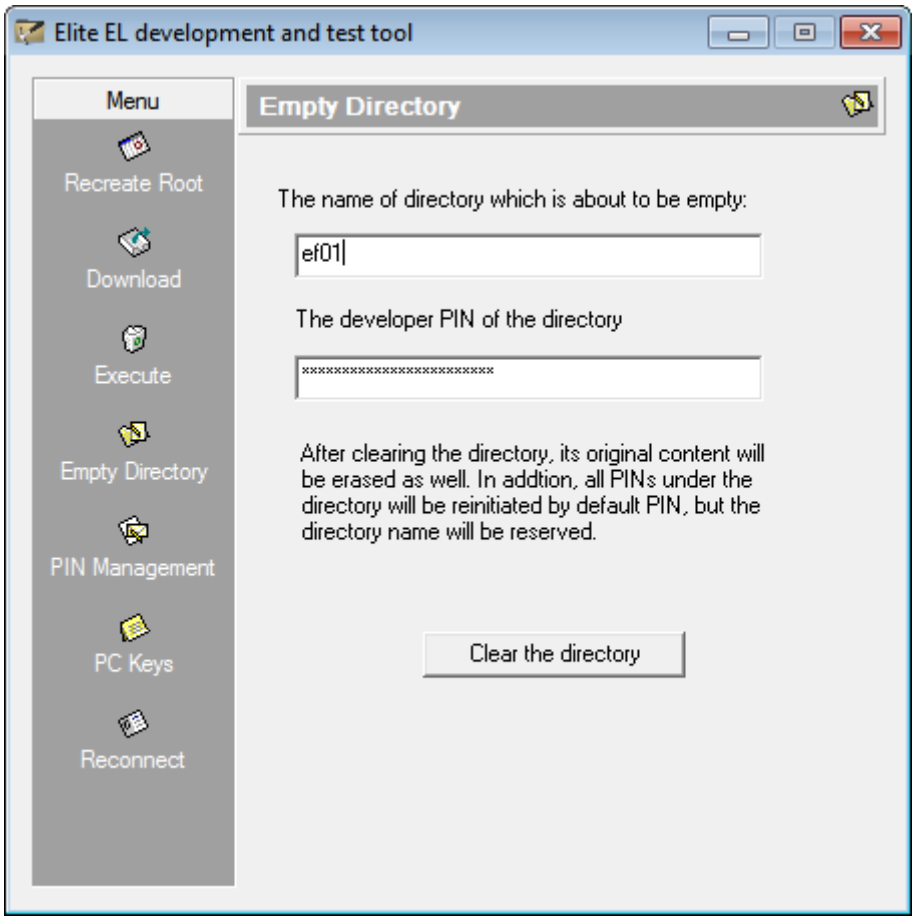


Figure 23: Clear Sub-directory Contents

PIN Management

PIN (also known as password) is the foundation of EL file system. For the detailed description of the PIN code, please refer to the Section “PIN”.

On the **PIN Management** of the DevTest, it is available modify the Developer PIN and User PIN as follows:

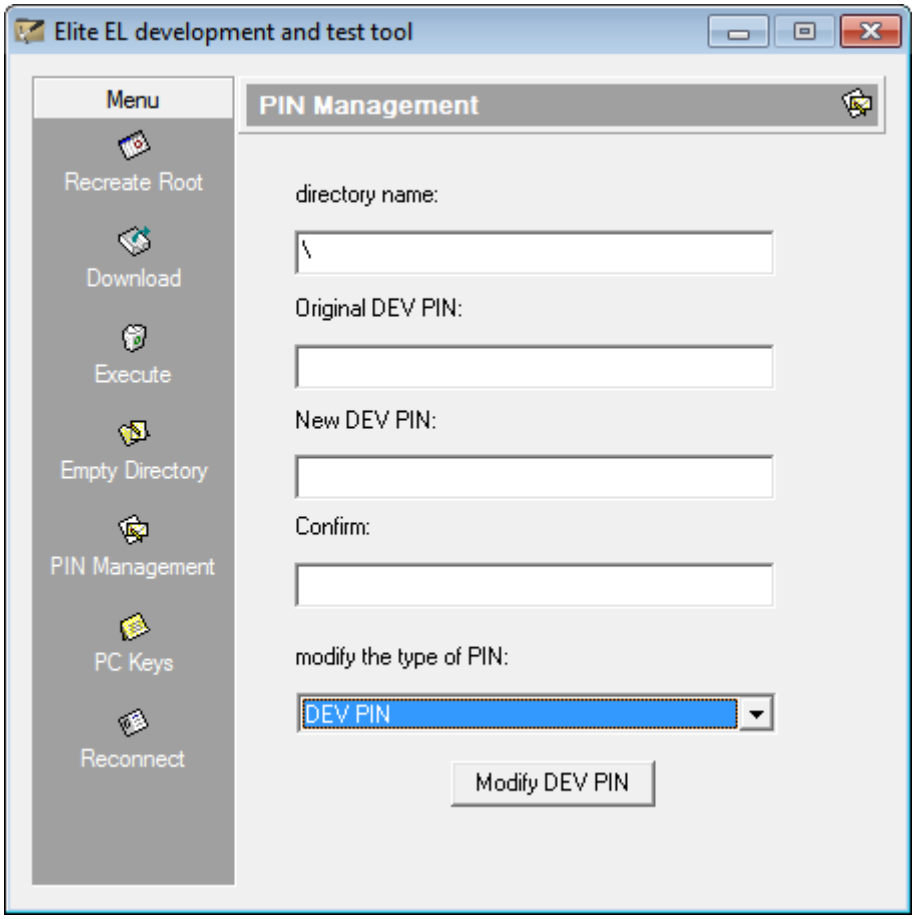


Figure 24: PIN Management

Input the **directory name** whose PIN code needs to be modified, select PIN modification type and input original PIN and new PIN according to text above the input field of PINs, Click the button to modify the PIN according to the selected PIN type.

Download Files

This function is to import disk files to the hardware device. The current downloadable file types are executable files, data files and RSA key files.

For the executable file, it is required that the disk file should be in INTEL HEX format or binary format. Plus, the tool will convert the HEX file automatically into BIN file and save it in the current directory.

The format required for the downloading of RSA files is S4_RSA_PUBLIC_KEY or S4_RSA_PRIVATE_KEY, which differs from the actual storage format in the hardware. The tool will convert automatically.

For detailed information, please refer to *S4WriteFile()*.

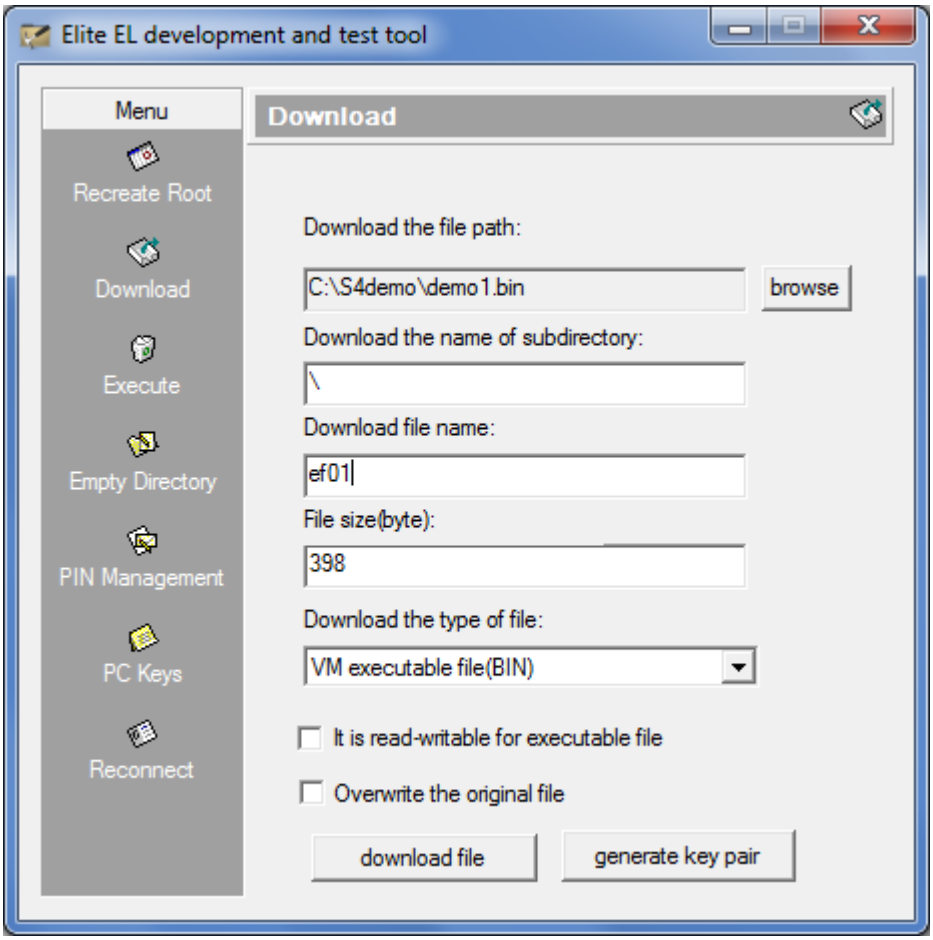


Figure 25: Download Files

It is necessary to choose the type of files to be downloaded and then click the button **Browse** select the target files. After the selection, the input box **File size** will be automatically updated to the actual storage size on the device. To create a new file, inputting a larger number will make the file usable repeatedly. Then, fill in the downloading directory and file name (the directory and file names should be a 2-byte hexadecimal number. In above figure, the file name is EF01). Click the button **Download File** to import the file to the hardware device.

For executable file, you could tick the checkbox **It is read-writable for executable file**. For the detail, please refer to the function *S4WriteFile()*.

If the file already exists, tick **Overwrite the original file** to update. The size of file to be downloaded must not be greater than the original.

If the signified child directory does not exist when downloading the file, a hint will remind you to build.

Empty Directory

Click **Empty Directory** on left panel, input the Directory ID and Developer PIN of directory to empty the directory and reinitiate with default PIN as follows:

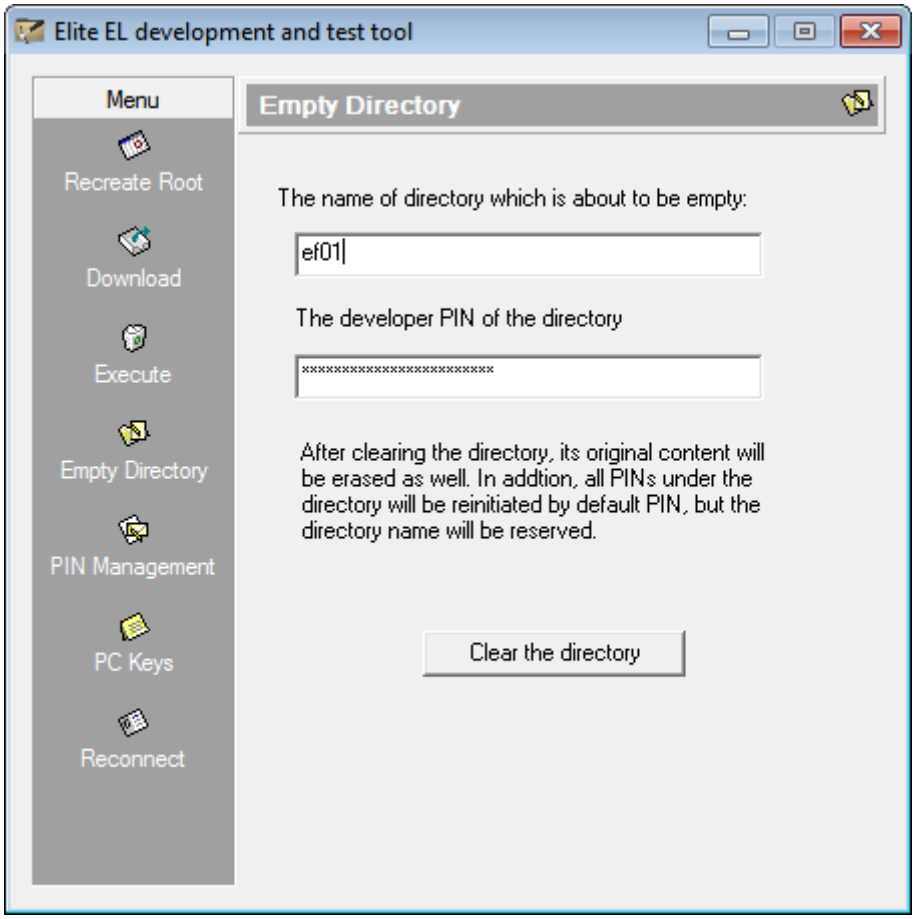


Figure 26: Clear Directory Content

Execute File

The function is to execute an exe file stored in the device.

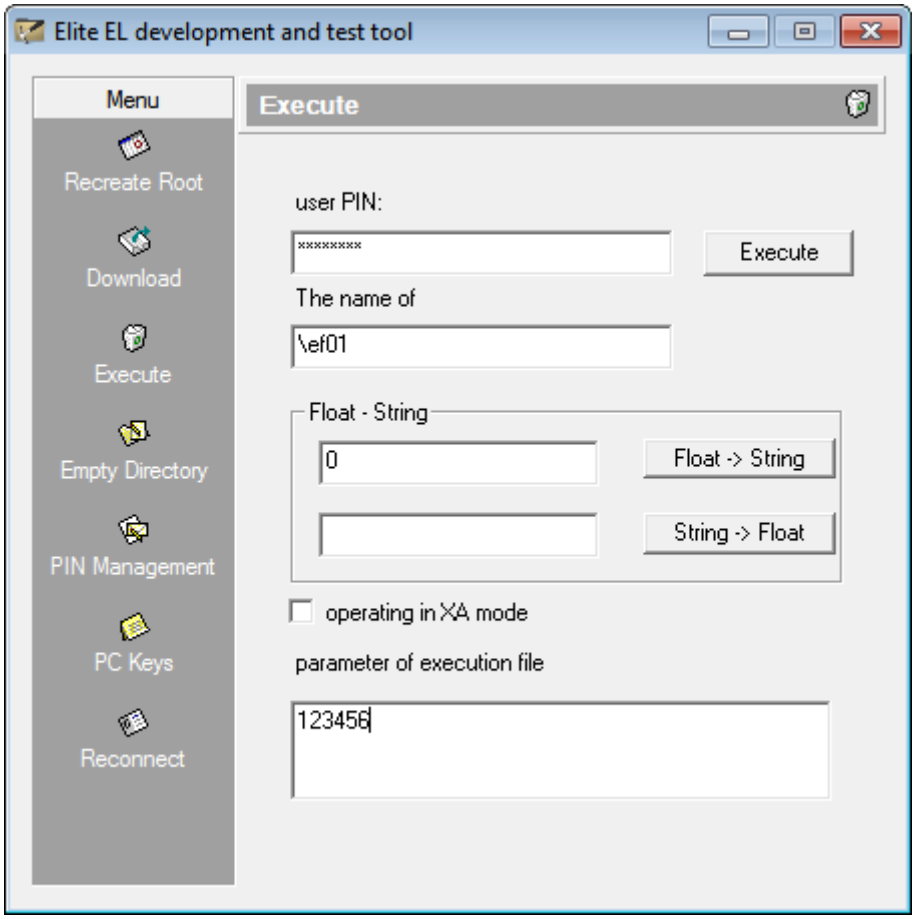


Figure 27: Execute File

While executing, User PIN is required. For the convenience of testing, “12345678” (Default User PIN) is preset. **The name of** must be filled by full path, for instance, “*000a*”. **Parameter of Execution File** must be in hexadecimal value.

Click the button **Execute** and the tool will execute the specified executable file in the hardware and return execution result.

PC Keys

It is designed to generate RSA key pair in format of S4_RSA_PUBLIC_KEY and S4_RSA_PRIVATE_KEY on the local computer and to import to the device or other usage.

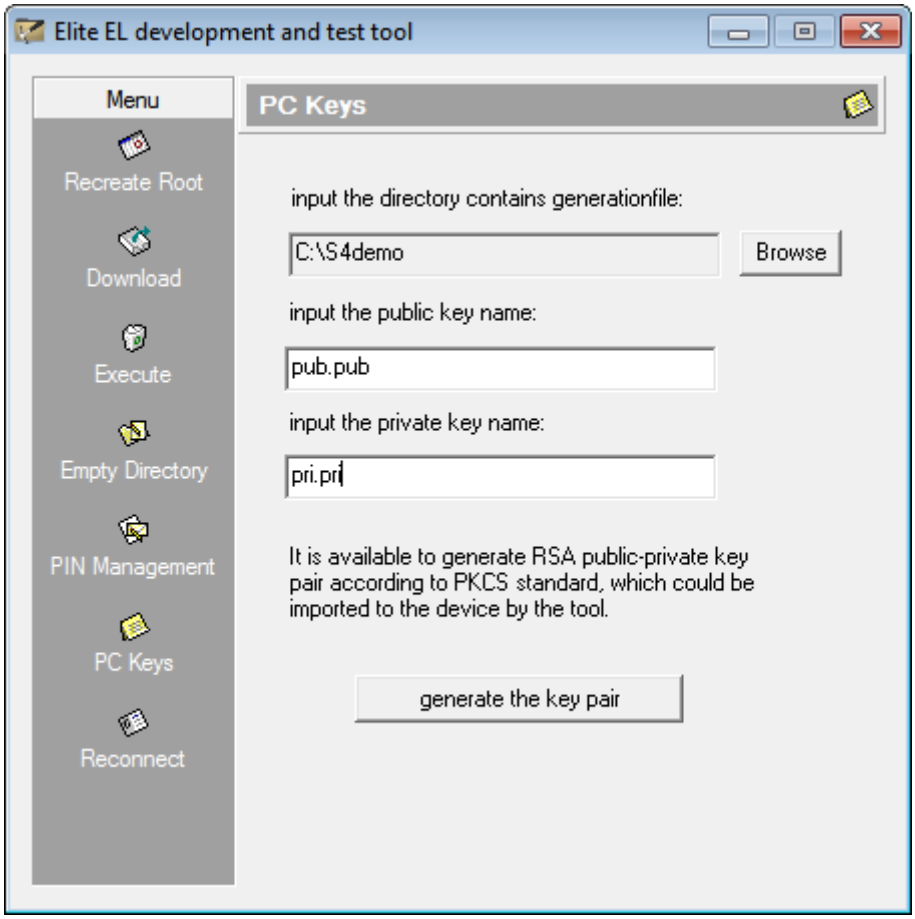


Figure 28: Download PC Keys

Choose the directory where the files are to be saved, fill in **Public Key file name** and **Private Key file name** and then click the button **Generate** to generate specified key pairs.

SES References 7

SES (System Extension Service) is a group of functions supported by hardware device. All the SES functions are declared in the header file: *ses_v3.h*. This header file also defines a group of macros together with some auxiliary functions to simplify the developing process. According to the different purpose, SES and macros can be divided to the following categories:



Some SES functions require specific hardware version. For more detail, please refer to the “Requirements” section of each SES function.

Category	Comments
Flow Control	Please refer to here
Input & Output	Please refer to here
File Operation	Used to operate internal file of EL, i.e. data file. For more detail, please refer to here
Mathematics	Support simple/double precision float calculation. Provide all the common mathematic functions defined in *math.h* of standard C language. For more detail, please refer to here
Cryptographic Algorithm	Provide SHA1, RSA, DES/TDES Cryptographic algorithms. For more detail, please refer to here
Memory Operation	Basic memory operation support. I.e.: dynamic memory allocation, memory copy etc. For more detail, please refer to here
Time Control	Timer and real time clock functions. For more detail, please refer to here
Macro and Auxiliary Functions	For more detail, please refer to here
Get Device Info	I.e. function used to get the GUID of the device. For more detail, please refer to here

Table 6: SES Categories

Flow Control

_exit

Exit EXF.

```
void _exit(void)
```

Parameters:

None

Return Values:

None

Remarks:

It is available to call anywhere to terminate EXF.

Requirements:

Hardware Version: v2.xx, v5.xx

Example:

Please refer to *_set_response().*

Input & Output

_set_response

Set the data transmitted to computer while exiting EXF.

```
BYTE  _set_response(
      BYTE   bLen,
      BYTE *pbBuff);
```

Parameters:

bLen [in] Length of returned data

pbBuff [in] Buffer address

Return Values:
SES_SUCCESS or error code

Remarks:

The length of returned data cannot exceed that of the communication buffer (250 bytes for standalone version).

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
#include “ses_v3.h”

char buff[32] = “This is a test buffer.”;

void main()
{
        _set_response(sizeof(buff), (BYTE *)buff);
        _exit();
}
```

File Operation

This group of functions is used to operate the internal file system of EL device.

_open

Select and open the specified file.

```
BYTE _open(
      WORD wFid,
      HANDLE *pHandle);
```

Parameters:

wFid [in] File ID.

pHandle [out] Handle to the file opened.

Return Values:
SES_SUCCESS or error code

Remarks:
EXF can open up to max 3 files concurrently. For operating more files, needless file must be closed using *_close()*.

Return value tells whether the file-open operation succeeds or not.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:
Please refer to the sample code of *_write()*.

_close

Close the file specified.

```
BYTE _close(HANDLE handle);
```

Parameters:
handle [in] Handle of the file to be closed

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:
Please refer to the sample code of *_write()*.

_read

Read data from an opened file. The read-write attributes of the file must be set appropriately for having reading privileges.

```
BYTE _read(
      HANDLE handle,
      WORD wOffset,
      BYTE bLength,
      BYTE* pbBuff);
```

Parameters:

handle	[in] Handle to the file opened.
wOffset	[in] Reading offset
bLength	[in] Length of the data to be read, 1~247bytes.
pbBuff	[out] Pointer to data buffer, used to store the data read.

Return Values:
SES_SUCCESS or error code

Remarks:
Data file, Public key file can be read by EXFs in the same directory freely.

If an EXF has been set “Unreadable/Unwritable”, then it cannot be read by other EXFs. Otherwise, it can be read by the other EXF residing in the same directory. It is highly recommended to set all the EXFs “UNREADABLE/UNWRITTABLE” to enhance security unless the remote update is being applied)

Private Key file can NEVER be read.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:
Please refer to the sample code of *_write()*.

_write

Write data to an opened file. The read-write attributes of the file must be set appropriately for having writing privileges.

```
BYTE _write(
      HANDLE handle,
      WORD   wOffset,
      BYTE   bLength,
      BYTE*  pbBuff);
```

Parameters:

handle	[in] Handle to the file opened.
wOffset	[in] Writing offset
bLength	[in] Length of the data to be written, 1~249bytes.
pbBuff	[in] Pointer to the data buffer, used to store data to be written in.

Return Values:
SES_SUCCESS or error code

Remarks:
Data file, Public key file can be written by EXFs in the same directory freely.

If an EXF has been set “Unreadable/Unwritable”, then it cannot be written by other EXFs. Otherwise, it can be written by the other EXF residing in the same directory. It is highly recommended to set all the EXFs “UNREADABLE/UNWRITTABLE” to enhance security unless the remote update is being applied)

Private Key file can be written by other EXF in the same directory.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
/* This demonstrates how to operate file in EL. */
#include “ses_v3.h”

unsigned char buff[128];

void main()
{
        HANDLE hFile1;
        HANDLE hFile2;
        BYTE ret = 0;

        /* Open data file 0xa001. */
        ret = _open(0xa001, &hFile1);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
```

```
        }

        /* Read 128 bytes from file 0xa001. */
        ret = _read(hFile1, 0, 128,buff);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }

        /* Open data file 0xa002. */
        ret = _open(0xa002, &hFile2);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }

        /* Write what we read to file 0xa002. */
        ret = _write(hFile2, 0, 128, buff);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }

        /* Close both files. */
        _close(hFile1);
        _close(hFile2);

        /* ret=0 indicates a successful operation.
                You can also send what we read to PC via
                _set_response(). */
        _set_response(1, &ret);

        /* Exit program. */
        _exit();

}
```

_create

Create a new file and inherit the read-write attributes from its parent (SES Caller).

```
BYTE _create(
      WORD    wFid,
      WORD    wSize,
      BYTE    bFileType,
      BYTE    bFlag,
      HANDLE * pHandle);
```

Parameters:

wFid	[in] ID of the file to be created.
wSize	[in] Size of the file.
bFileType	[in] File type, possible values:
bFlag	SES_FILE_TYPE_EXE (EXF)
	SES_FILE_TYPE_EXE_DATA (Data file)
	SES_FILE_TYPE_RSA_PUB (RSA Public Key file)
	SES_FILE_TYPE_RSA_SEC (RSA Private Key file)
pHandle	[in] Flag, possible values:
	CREATE_OPEN_ALWAYS
	Open the file if it already exists, create a new file otherwise.
	CREATE_FILE_NEW
	Create and open the file, return error if it already exists.
	CREATE_OPEN_EXISTING
	Open an existing file. Similar to *_open()*.
	[out] Handle to the file if it succeeds to open.

Return Values:
SES_SUCCESS or error code

Remarks:
This function is designed specifically for remote operations; files created by this SES function can only be accessible internally. In other words, even with the Developer PIN, it is still not available to write by using *S4WriteFile()* or *PS4WriteFile()*.

File created by this SES function will inherit the read-write attribute from its parent (SES Caller).The precondition to create a usable new file is to set its caller “READABLE/WRITTABLE”. Otherwise, the dead file will be created.

Executable (EXF) created by this SES function must be enabled using SES function *_enable_exe()* before being executed.

Executing this function does not write any data into the file but calling *_write()*.

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

Please refer to the sample code of *_enable_exe()*.

_enable_exe

Enable EXFs created by SES *_create()*.

```
BYTE _enable_exe(WORD wFid);
```

Parameters:
wFid [in] ID of the EXF to be enabled.

Return Values:
SES_SUCCESS or error code

Remarks:
It is effective only on EXFs created by SES *_create()*.

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

```
/* This demonstrates how to create a new exe file and enable it in EL. */
#include “ses_v3.h”

void main()
{
        BYTE ret = 0;
        HANDLE hFile;

        /* Create a new exe file with file ID 0xa003. */
        ret = _create(0xa003, 1024, SES_FILE_TYPE_EXE,
                                        CREATE_FILE_NEW, &hFile);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }

        /* SHOULD set file content here….*/

        /* Enable the newly created file. */
        ret = _enable_exe(0xa003);

        _set_response(1, &ret);
        _exit();
}
```

Mathematics

This category includes all the commonly used mathematic functions and is the largest group of functions of SES.

Single Precision

If single precision calculation result is sufficient, using SES functions is not absolutely necessary but using predefined functions in *math.h*.

For VM EXF, using single precision float SES functions can boost calculation speed greatly.

Double Precision

Applying double precision float calculation MUST use SES to achieve due to the fact that the compiler cannot support effective double type (keyword “double” maybe handled as “float”, depending on the compiler setting). It is represented using a struct (defined as DOUBLE_T) and thus there is no way to assign a value simply by using “=”.

There are three common methods to make an assignment:

- **Memory Copy**
  For instance, get the representation of byte array in memory before assignment and then copy this array to a struct DOUBLE_T;
- **Type Switch**
  Using SES to convert different format among single precision, integer, and double precision;
- **Auxiliary Functions**
  Call the auxiliary function which converts a string to a double precision float.

About the struct DOUBLE_T, the following attributes need to pay attention:

- This struct takes 8 bytes totally, and it has the same format as the double precision float real*8 stipulated in IEEE.
- DOUBLE_T uses Little-Endian, which is compiler-independent.
- Single and double precision float SES functions have similar naming scheme, all the single precision functions have the suffix of ‘f’.

_addf

Single precision float addition

```
float _addf(
      float x,
      float y);
```

Parameters:

X [in] augends

Y [in] addend

Return Values:
Sum of two single precision floats

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
/* After running, the result of sum will be 4.9.*/
float sum, a = 3.2, b = 1.7;
sum = _addf(a, b);
```

_add

Double precision float addition

```
BYTE _add(
      DOUBLE_T      *presult,
      DOUBLE_T      *px,
      DOUBLE_T      *py);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the augends.
py	[in] Pointer to the addend.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

```
DOUBLE_T sum, a, b;
BYTE ret = 0;
/* assign value to a and b here... */
ret = _add(&sum, &a, &b);
```

_subf

Single precision float subtraction

```
float _subf(
      float x,
      float y);
```

Parameters:

X	[in]minuend
Y	[in] subtrahend

Return Values:
Subtraction of two single precision floats

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_sub

Double precision float subtraction

```
BYTE _sub(
      DOUBLE_T      *presult,
      DOUBLE_T      *px,
      DOUBLE_T      *py);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the minuend.
py	[in] Pointer to the subtrahend.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_multf

Single precision float multiplication

```
float _subf(
        float x,
        float y);
```

Parameters:

X	[in]multiplicand
Y	[in] multiplier

Return Values:
Product of two single precision floats

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_mult

Double precision float multiplication

```
BYTE _mult(
      DOUBLE_T      *presult,
      DOUBLE_T      *px,
      DOUBLE_T      *py);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the multiplicand.
py	[in] Pointer to the multiplier.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_divf

Single precision float division

```
float _divf(
      float x,
      float y);
```

Parameters:

X	[in]dividend
Y	[in] devisor

Return Values:
Quotient of two single precision floats

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_div

Double precision float division

```
BYTE _div(
      DOUBLE_T      *presult,
      DOUBLE_T      *px,
      DOUBLE_T      *py);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the dividend.
py	[in] Pointer to the divisor.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_sinf

Sine of a single precision float

```
float _sinf(float x);
```

Parameters:
X [in]Radian

Return Values:
Sin of the radian

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_sin

Sine of a double precision float

```
BYTE _sin(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the radian.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_cosf

Cosine of a single precision float

```
float _cosf(float x);
```

Parameters:
X [in]Radian

Return Values:
Cosine of the radian.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_cos

Cosine of a double precision float

```
BYTE _cos(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the radian.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_tanf

Tangent of a single precision float

```
float _tanf(float x);
```

Parameters:
X [in]Radian

Return Values:
Tangent of the radian.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_tan

Tangent of a double precision float

```
BYTE _tan(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the radian.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_asinf

Asin of a single precision float

```
float _asinf(float x);
```

Parameters:
X [in]Sine value.

Return Values:
Asin of the float.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_asin

Asin of a double precision float

```
BYTE _asin(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the sine value.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_acosf

Acos of a single precision float

```
float _acosf(float x);
```

Parameters:
X [in]Cosine value.

Return Values:
Acos of the float.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_acos

Acos of a double precision float

```
BYTE _acos(
      DOUBLE_T      *presult,
      DOUBLE_T      *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the cosine value.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_atanf

Atangent of a single precision float

```
float _atanf(float x);
```

Parameters:
X [in]Tangent value.

Return Values:
Atangent of the float.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_atan

Atangent of a double precision float

```
BYTE _atan(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the tangent value.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_atan2f

Atangent of two single precision float's quotient

```
float _atan2f(
      float x,
      float y);
```

Parameters:

X	[in]dividend
Y	[in] divisor

Return Values:
Atangent of two floats' quotient.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_atan2

Atangent of two double precision float

```
BYTE _atan2(
      DOUBLE_T      *presult,
      DOUBLE_T      *px,
      DOUTLE_T      *py);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the tangent value.
py	[in] Pointer to the divisor.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_sinhf

Sinh of a single precision float

```
float _sinhf(float x);
```

Parameters:
X [in]Radian.

Return Values:
Sinh of the float.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_sinh

Sinh of a double precision float

```
BYTE _sinh(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the radian.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_coshf

Cosh of a single precision float

```
float _coshf(float x);
```

Parameters:
X [in]Radian.

Return Values:
Cosh of the float.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_cosh

Cosh of a double precision float

```
BYTE _cosh(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the radian.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_tanhf

Tanh of a single precision float

```
float _tanhf(float x);
```

Parameters:
X [in]Radian.

Return Values:
Tanh of the float.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_tanh

Tanh of a double precision float

```
BYTE _tanh(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the radian.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_ceilf

Calculates the ceiling of a single precision float

```
float _ceilf(float x);
```

Parameters:
X [in]Float parameter.

Return Values:
The smallest integer that is greater than or equal to x.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_ceil

Calculates the ceiling of a single precision float

```
BYTE _ceil(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the float parameter.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_floorf

Calculates the floor of a single precision float

```
float _floorf(float x);
```

Parameters:
X [in] Float parameter.

Return Values:
The greatest integer that is less than or equal to x.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_floor

Calculates the floor of a double precision float

```
BYTE _floor(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the float parameter.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_absf

Calculate the absolute value of a single precision float

```
float _absf(float x);
```

Parameters:
X [in] Float parameter.

Return Values:
Absolute value of x.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_abs

Calculate the absolute value of a double precision float

```
BYTE _abs(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the float parameter.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_fmodf

Calculates the single precision floating-point remainder

```
float _fmodf(
      float x,
      float y);
```

Parameters:

X	[in] Dividend.
Y	[in] Divisor.

Return Values:
The floating point remainder of x/y.

Remarks:
It calculates the floating-point remainder **f** of **x/y**, as **x=I*y+f**, where **i** is an integer, **f** has the same sign as **x**, and the absolute value of **f** is less than that of **y**.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_fmod

Calculates the double precision floating-point remainder

```
BYTE _fmod(
      DOUBLE_T      *presult,
      DOUBLE_T      *px,
      DOUBLE_T      *py);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the dividend.
py	[in] Pointer to the divisor.

Return Values:
SES_SUCCESS or error code

Remarks:
It calculates the floating-point remainder **f** of **x/y**, as **x=I*y+f**, where **i** is an integer, **f** has the same sign as **x**, and the absolute value of **f** is less than that of **y**.

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_expf

Calculates the exponential

```
float _expf(float x);
```

Parameters:
X [in] Exponent.

Return Values:
The exponential value of the floating-point parameter x, if it is succeeded.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_exp

Calculates the exponential

```
BYTE _exp(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the dividend.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_logf

Calculates logarithms of a single precision float

```
float _logf(float x);
```

Parameters:
X [in] Float parameter.

Return Values:
Logarithm of the single precision float x.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_log

Calculates logarithms of a double precision float

```
BYTE _log(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the float parameter.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_log10f

Returns the base 10 logarithm of a specified single precision float

```
float _log10f(float x);
```

Parameters:
X [in] Float parameter.

Return Values:
The logarithm with base 10 of a specified single precision float.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_log10

Returns the base 10 logarithm of a specified double precision float

```
BYTE _log10(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult [out] Pointer to the result.

px [in] Pointer to the float parameter.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_sqrtf

Calculates the square root of a single precision float

```
float _sqrtf(float x);
```

Parameters:
X [in] Float parameter.

Return Values:
The square root of the single precision float x.

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

_sqrt

Calculates the square root of a double precision float

```
BYTE _sqrt(
      DOUBLE_T       *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the float parameter.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_powf

Calculates x rose to the power of y where both x and y are of single precision float

```
float _powf(
      float  x,
      float  y);
```

Parameters:

X [in] Base.

Y [in] Exponent.

Return Values:

The value of x^y.

Remarks:

None

Requirements:

Hardware Version: v2.xx, v5.xx

Example:

_pow

Calculates x rose to the power of y where both x and y are of double precision float

```
BYTE _pow(
      DOUBLE_T      *presult,
      DOUBLE_T      *px,
      DOUBLE_T      *py);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the base.
py	[in] Pointer to the exponent.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:



The following functions have only double float version.

_modf

Divide a double precision float to integral part and fractional part.

```
BYTE _modf(
      DOUBLE_T      *presult,
      DOUBLE_T      *px,
      DOUBLE_T      *intptr);
```

Parameters:

presult	[out] Pointer to fractional part
px	[in] Pointer to the double to be divided.
intptr	[out] Pointer to the integral part.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_frexp

Divides a double precision float to the product of a fraction and power of 2

```
BYTE _frexp(
      DOUBLE_T       *presult,
      DOUBLE_T       *px,
      int            *expptr);
```

Parameters:

presult	[out] Pointer to fractional part.
px	[in] Pointer to the double to be divided.
expptr	[out] Pointer to the exponent.

Return Values:
SES_SUCCESS or error code

Remarks:
The equation is *px = *presult × $2^{(*expptr)}$, where 1 > *presult >= 0.5

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_ldexp

Calculate the power of a double precision float and 2

```
BYTE _ldexp(
      DOUBLE_T      *presult,
      DOUBLE_T      *px,
      int    exp);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the float parameter.
exp	[in] Exponent of 2.

Return Values:
SES_SUCCESS or error code

Remarks:
The calculation formula is like *presult = *px × $2^{(exp)}$

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_fdcmp

Compare two double precision floats

```
char _fdcmp(
      DOUBLE_T       *px,
      DOUBLE_T       *py);
```

Parameters:
px [in] Pointer to the first float parameter.

exp [in] Pointer to the second float parameter.

Return Values:
Returns three possible values:

1 stands for *px>*py

0 stands for *px=*py

-1 stands for *px<*py

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_dtof

Type conversion: from double precision to single precision float

```
BYTE _dtof(
      float          *presult,
      DOUBLE_T       *px);
```

Parameters:

presult	[out] Pointer to the single precision float.
px	[in] Pointer to the double precision float.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_ftod

Type conversion: from single precision to double precision float

```
BYTE _ftod(
      DOUBLE_T       *presult,
      float          *px);
```

Parameters:

presult [out] Pointer to the double precision float.

px [in] Pointer to the single precision float.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_dtol

Type conversion: from double precision float to 32 bits signed long integer.

```
BYTE _dtol(
      long          *presult,
      DOUBLE_T      *px);
```

Parameters:

presult	[out] Pointer to the result.
px	[in] Pointer to the double precision float.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_altod

Type conversion: from 32 bits signed integer (long) to double precision float

```
BYTE _altod(
      DOUBLE_T       *presult,
      long           *px);
```

Parameters:
presult [out] Pointer to the result.

px [in] Pointer to 32 bits signed integer.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

_lltod

Type conversion: from 32 bits unsigned long integer to double precision float.

```
BYTE _lltod(
      DOUBLE_T              *presult,
      DWORD   *px);
```

Parameters:
presult [out] Pointer to the result.

px [in] Pointer to 32-bit unsigned integer.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.30 or above, v5.xx

Example:

Cryptographic Algorithm

EL provides four categories of fundamental cryptographic algorithms: Asymmetric cipher algorithm (RSA), Symmetric cipher algorithm (DES/TDES), hash algorithm (SHA1) and real random number generator. These cryptographic algorithms can not only be used in software copyrights protection, but more importantly, can help developers to expand their software to network. For example, they can be used together to implement a capable Remote Update platform.

To use these cryptographic algorithm functions properly, some basic understanding of cryptographic algorithm and its application is required. This manual is not professional material for cryptography study and only describes the algorithms and protocols briefly.

The cryptographic algorithm has corresponding software version implementation. For example, you can use software-version functions to encrypt data, and then use corresponding SES of internal hardware to decrypt. For more detail about software version cryptographic algorithms, please refer to the “Appendix A: Cryptographic API”.



Some of the cryptographic algorithm SES functions are reserved only for compatibility. For the convenience of technology support, it’s NOT recommended to use. They are marked explicitly in this guide.

_tdes_enc

TDES Encryption, ECB mode

```
BYTE _tdes_enc(
      BYTE    *pbKey,
      BYTE    bLen,
      BYTE    *pbData);
```

Parameters:

pbKey	[in] 16 octets DES key
bLen	[in] Length of the data to be encrypted 1~248.
pbData	[in, out] Plaintext to be encrypted when inputting, ciphertext when outputting.

Return Values:
SES_SUCCESS or error code

Remarks:
It accomplishes triple-DES encryption in ECB mode dividing the plaintext into 8-octet blocks, and independently generating ciphertext for each block with the same length. If you would like to use different encryption mode i.e. CBC, please encapsulate codes based on it.

Keep the length of plaintext be a multiple of 8, or add extra to meet the request. Otherwise, the function will fill extra plaintext automatically, and lead to decrypted result containing non-user defined plaintext.

The length of ciphertext is always a multiple of 8, and maybe longer than that of the plaintext (if the latter is not a multiple of 8). For the plaintext and the ciphertext use the same buffer, the former will be overwritten by the latter. If the plaintext is required to use after calling this SES, please backup it to other memory block. In the other hand, you need sufficient buffer to hold the output ciphertext to avoid possible memory leak since the ciphertext maybe longer than the plaintext.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:
Please refer to the sample code of *_tdes_dec()*.

_tdes_dec

TDES decryption in ECB mode, paired with *_tdes_enc()*

```
BYTE _tdes_dec(
        BYTE    *pbKey,
        BYTE    bLen,
        BYTE    *pbData);
```

Parameters:

pbKey	[in] 16 octets DES key
bLen	[in] Length of the data to be decrypted, 8~248, must be a multiple of 8.
pbData	[in, out] Ciphertext when input, plaintext when output.

Return Values:
SES_SUCCESS or error code

Remarks:
It accomplishes triple-DES decryption in ECB mode dividing the ciphertext into 8-octet blocks, and independently generating plaintext for each block with the same length. If you would like to use different decryption mode i.e. CBC, please encapsulate codes based on it.

The length of ciphertext generated by triple-DES encryption is always a multiple of 8, so this function demands the length of inputting ciphertext to be a multiple of 8.

For the plaintext and the ciphertext use the same buffer, the latter will be overwritten by the former. If the ciphertext is required to use after calling this SES, please backup to other memory block.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
/*      This demonstrate how to use Triple-DES algorithm to
        encrypt some text and decrypt it later.
*/

#include “ses_v3.h”
#include “string.h”

char plaintext[24] = “This is some test data.”;
BYTE ciphertext[24] = “”;
BYTE buff[24] = “”;
BYTE deskey[16] = {0x23,0x5a,0xb8,0x91,0xfc,0xe2,0x6c,0x9a,

0x3a,0x98,0x34,0x8e,0x1d,0xaa,0x97,0xe1};

void main()
{
        BYTE ret = 0;
```

```
        /* Encrypt…*/
        memcpy(ciphertext, plaintext, 24);
        ret = _tdes_enc(deskey, 24, ciphertext);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
        }

        /* Decrypt… */
        memcpy(buff, ciphertext, 24);
        ret = _tdes_dec(deskey, 24, buff);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
        }
        /* Now the content of buff should be the same as
        plaintext. */

        _set_response(1,&ret);
        _exit();

}
```

_des_enc

DES encryption in ECB mode

```
BYTE _des_enc(
      BYTE    *pbKey,
      BYTE    bLen,
      BYTE    *pbData);
```

Parameters:

pbKey	[in] 8 octets DES key.
bLen	[in] Length of the data to be encrypted, 1~248.
pbData	[in, out] Plaintext to be encrypted when input, ciphertext when output.

Return Values:
SES_SUCCESS or error code

Remarks:
It accomplishes DES encryption in ECB mode dividing the plaintext into 8-octet blocks, and independently generating ciphertext for each block with the same length. If you would like to use different encryption mode i.e. CBC, please encapsulate codes based on it.

This function uses 8-octet key, which has 56 effective bits and is considered to compromise the security. Please use *_tdes_enc()* which has 112 effective bits key instead. As the calculation is implemented by the DES/TDES accelerator in hardware, there will be no difference in performance between DES and TDES.

Keep the length of plaintext be a multiple of 8, or add extra to meet the request. Otherwise, the function will fill extra plaintext automatically, and lead to decrypted result containing non-user defined plaintext.

The length of ciphertext is always a multiple of 8, and it maybe longer than that of the plaintext (if the latter is not a multiple of 8). Because the plaintext and the ciphertext use the same buffer, the former will be overwritten by the latter. So if you want to use the plaintext after calling this SES, please backup it to other memory block. In the other hand, you need sufficient buffer to hold the output ciphertext to avoid possible memory leak since the ciphertext maybe longer than the plaintext.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:
Please refer to the sample code of*_des_dec()*.

_des_dec

DES decryption in ECB mode, paired with *_des_enc()*

```
BYTE _des_dec(
        BYTE    *pbKey,
        BYTE    bLen,
        BYTE    *pbData);
```

Parameters:

pbKey	[in] 8 octets DES key.
bLen	[in] Length of the data to be decrypted, 8~248, must be a multiple of 8.
pbData	[in, out] Ciphertext when input, plaintext when output.

Return Values:
SES_SUCCESS or error code

Remarks:
It accomplishes DES decryption in ECB mode dividing the ciphertext into 8-octet blocks, and independently generating plaintext for each block with the same length. If you would like to use different decryption mode i.e. CBC, please encapsulate codes based on it.

This function uses 8-octet key, which has 56 effective bits and is considered to compromise the security. Please use *_tdes_dnc()* which has 112 effective bits key instead. As the calculation is implemented by the DES/TDES accelerator in hardware, there will be no difference in performance between DES and TDES.

The length of ciphertext generated by DES encryption is always a multiple of 8, so this function demands the length of inputting ciphertext to be a multiple of 8.

For the plaintext and the ciphertext use the same buffer, the latter will be overwritten by the former. If the ciphertext is required to use after calling this SES, please backup to other memory block.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
/*      This demonstrate how to use DES algorithm to encrypt
        some text and decrypt it later.
        This sample is almost the same as that using DES.
*/
#include “ses_v3.h”
#include “string.h”

char plaintext[24] = “This is some test data.”
BYTE ciphertext[24] = “”;
BYTE buff[24] = “”;
BYTE deskey[8] = {0x23,0x5a,0xb8,0x91,0xfc,0xe2,0x6c,0x9a};
```

```
void main()
{
	BYTE ret = 0;

	/* Encrypt…*/
	memcpy(ciphertext, plaintext, 24);
	ret = _des_enc(deskey, 24, ciphertext);
	if (ret != SES_SUCCESS)
	{
		_set_response(1, &ret);
	}

	/* Decrypt… */
	memcpy(buff, ciphertext, 24);
	ret = _des_dec(deskey, 24, buff);
	if (ret != SES_SUCCESS)
	{
		_set_response(1, &ret);
	}
	/* Now the content of buff should be the same as
	plaintext. */

	_set_response(1,&ret);
	_exit();

}
```

_sha1_init

SHA1 initialization

```
BYTE _sha1_init(HASH_CONTEXT *pContext);
```

Parameters:
pContext [in] HASH context.

Return Values:
SES_SUCCESS or error code

Remarks:
It initializes HASH context. A typical HASH procedure includes:

Initialize context using *_sha1_init()*;

Add data by calling *_sha1_update()* several times;

Get hash result using *sha1_ final()*,purge the hash context at the same time;

Function group *_sha1_xxx()* needs to maintain a buffer in the system, which will be used for the completion of RSA signature verification. For detail, please refer to *_rsa_veri().*

Multi HASH concurrent processes are not allowed. Namely, a HASH process can run only after the previous one has completed. For instance, the following procedure is not legal and results incorrectly:

```
…
HASH_CONTEXT hctx1, hctx2;
BYTE message1[] = “test1”;
BYTE message2[] = “test2”;
BYTE len1 = 5;
BYTE len2 = 5;
BYTE digest1[20], digest2[20];

_sha1_init(&hctx1);
_sha1_init(&hctx2);

_sha1_update(&hctx1, message1, len1);
_sha1_update(&hctx2, message2, len2);

_sha1_final(&hctx1, digest1);
_sha1_final(&hctx2, digest2);
…
```

The correct procedure is:

```
…
HASH_CONTEXT hctx1, hctx2;
BYTE message1 = “test1”;
BYTE message2 = “test2”;
BYTE len1 = 5;
BYTE len2 = 5;
BYTE digest1[20], digest2[20];

_sha1_init(&hctx1);
```

```
        _sha1_update(&hctx1, message1, len1);
        _sha1_final(&hctx1, digest1);

        _sha1_init(&hctx2);
        _sha1_update(&hctx2, message2, len2);
        _sha1_final(&hctx2, digest2);
…
```

Requirements:
Hardware Version: v2.xx, v5.xx

Example:
Please refer to the sample code of *_sha1_final()*.

_sha1_update

Add SHA1 data by using a calling sequence. For detail, please refer to the remarks of *_sha1_init()*.

```
BYTE _sha1_update(
      HASH_CONTEXT *pContext,
      BYTE         *pbData,
      BYTE          bLen);
```

Parameters:

pContext	[in] HASH Context.
pbData	[in] Data to be added this time.
bLen	[in] Length of the data to be added this time.

Return Values:
SES_SUCCESS or error code

Remarks:
Use this function to add data to be hashed. A big data block can be divided to several smaller ones and added to hash by calling *_sha1_update()* continuously.

SHA1 of software version has no limitation for the input data, however, hardware device cannot tolerate over 8191 bytes input data due to its recourse limitation.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:
Please refer to the sample code of *_sha1_final()*.

_sha1_final

Get the result of SHA1 HASH and then purge the context. SHA1 is achieved by a group of functions. For more detail, please refer to the “Remark” section of *_sha1_init()*.

```
BYTE _sha1_final(
      HASH_CONTEXT *pContext,
      BYTE         *pbResult);
```

Parameters:

pContext	[in] HASH Context.
pbResult	[out] 20 bytes hash result returned.

Return Values:
SES_SUCCESS or error code

Remarks:
After calling this SES, the system will also copy the result to a specific HASH buffer so as to be used in signature verification process. For relations between HASH buffer and signature verification, please refer to the “Remarks” section of *_rsa_veri()*.

pbResult must have sufficient storage for the HASH result.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
/* This demonstrates how to get digest of a message. */

#include “ses_v3.h”

HASH_CONTEXT hctx;
BYTE message1[] = “This is the first part of test message.”;
BYTE message2[] = “This is the secont part of test message.”;
BYTE digest[20];

void main()
{
        BYTE ret = 0;

        ret = _sha1_init(&hctx);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }

        ret = _sha1_update(&hctx, message1, sizeof(message1));
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }
```

```
	ret = _sha1_update(&hctx, message2, sizeof(message2));
	if (ret != SES_SUCCESS)
	{
		_set_response(1, &ret);
		_exit();
	}

	ret = _sha1_final(&hctx, digest);
	if (ret != SES_SUCCESS)
	{
		_set_response(1, &ret);
		_exit();
	}

	/* Now we get the digest of message1 and message2. */

	_set_response(1, &ret);
	_exit();
}
```

_rsa_enc

RSA Encryption

```
BYTE _rsa_enc(
      BYTE    bMode,
      WORD    wFid,
      BYTE    bLen,
      BYTE    *pbData);
```

Parameters:

bMode	[in] Calculation Mode: RSA_CALC_NORMAL Direct calculation without encoding. RSA_CALC_PKCS Encryption according to PKCS#1 standard.
wFid	[in] Public Key file ID
bLen	[in] Length of the data to be encrypted, different for two modes: RSA_CALC_NORMAL (128 bytes) RSA_CALC_PKCS (1~117 bytes)
pbData	[in, out] Plaintext when input, 128 bytes ciphertext when output.

Return Values:
SES_SUCCESS or error code

Remarks:
It completes the RSA Public Key encryption, and the length of the Public Key is 1024 bits. Different with DES encryption, you have to specify an effective RSA Public Key file instead of giving the key value directly.

The encrypted result is always 128 bytes in length; you have to allocate sufficient storage for *pbData*.

When using this function for data encryption, setting *bMode* to be RSA_CALC_PKCS is highly recommended. Using RSA_CALC_NORMAL, set the first byte of the 128 bytes data to be 0, otherwise, it may lead to an error due to the input data greater than the modulus. In fact, when inputting data less than 128 bytes under RSA_CALC_NORMAL mode, the function will fill extra automatically (adding 0s in the forehead) which is not recommended at all. RSA_CALC_NORMAL is commonly used in constructing other algorithm protocols, for instance, digital signature.

There are three methods of generating RSA key pair files in hardware device:

Call API *S4WriteFile()or PS4WriteFile()* to generate a pair of RSA key files in device;

Call API *S4WriteFile()or PS4WriteFile()* to import a pre-generated RSA key pair into device;

Use the DevTest.exe to generate or import a pair of RSA keys to device;

RSA keys in memory are stored using TLV (Tag + Length + Value) format.

The following illustrates the Public Key format, 136 bytes in total:

```
typedef struct {
	struct {
		char tag;
		unsigned char length;
		unsigned char value[0x80];
	}n;
	struct {
		char tag;
		unsigned char length;
		unsigned char value[0x04];
	}e;
} COS_RSA_PUBLIC_KEY;
```

Private Key can be stored in two formats. The following is in quintuple format which takes 330 bytes, is faster, and recommended to use:

```
typedef struct {
	struct {
		char tag;
		unsigned char length;
		unsigned char value[0x40];
	}p;
	struct {
		char tag;
		unsigned char length;
		unsigned char value[0x40];
	}q;
	struct {
		char tag;
		unsigned char length;
		unsigned char value[0x40];
	}dp;
	struct {
		char tag;
		unsigned char length;
		unsigned char value[0x40];
	}dq;
	struct {
		char tag;
		unsigned char length;
		unsigned char value[0x40];
	}qinv;
} COS_RSA_PRIVATE_KEY;
```

The second one is in a simplified format, but quite slow in performance, not so commonly used and not recommended. It takes 260 bytes totally:

```
typedef struct {
	struct {
		char tag;
		unsigned char length;
		unsigned char value[0x80];
```

```
        }n;
        struct {
                char tag;
                unsigned char length;
                unsigned char value[0x80];
        }d;
} COS_RSA_PRIVATE_KEY_2;
```

Requirements:
Hardware Version: v2.xx, v5.xx

Example:
Please refer to the sample code of *_rsa_dec()*.

_rsa_dec

RSA decryption

```
BYTE _rsa_dec(
      BYTE    bMode,
      WORD    wFid,
      BYTE    bLen,
      BYTE    *pbData);
```

Parameters:

bMode	[in] Calculation mode: RSA_CALC_NORMAL Calculation directly without encoding RSA_CALC_PKCS Decryption according to PKCS#1 standard
wFid	[in] ID of Private Key file.
bLen	[in] Length of the data to be decrypted must be 128 bytes.
pbData	[in, out] Ciphertext when input, plaintext when output.

Return Values:
SES_SUCCESS or error code

Remarks:
It completes the RSA Private Key decryption, and the length of the Private Key is 1024 bits. Different with DES decryption, you have to specify an effective RSA Private Key file instead of giving the key value directly.

To decrypt in RSA_CALC_NORMAL mode, the result is always 128 bytes long; in RSA_CALC_PKCS, the result is “data length” + “plaintext”. Namely, first byte of *pbData* is the length of plaintext, and the following bytes are the decrypted plaintext.

Encryption and decryption needs to be done in the same mode, otherwise there will be encoding error.

For the generation and format of RSA keys, please refer to the remarks section of *_rsa_enc()*.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
/*      This demonstrates how to use RSA algorithm to encrypt
        some text and decrypt it later.
        The sample code supposes a pair of valid RSA key files
        have already been created.
*/

#include “ses_v3.h”
#include “string.h”

char plaintext[24] = “This is some test data.”;
BYTE ciphertext[128] = “”;
```

```
BYTE buff[128] = “”;
WORD pubkey_fid = 0xc001;
WORD prikey_fid = 0xc002;

void main()
{
        BYTE ret = 0;

        /* Encrypt… */
        memcpy(ciphertext, plaintext, 24);
        ret = _rsa_enc(RSA_CALC_PKCS, pubkey_fid, 24, ciphertext);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }

        /* Decrypt…
                Notice that the ciphertext is 128 bytes long. */
        memcpy(buff, ciphertext, 128);
        ret = _rsa_dec(RSA_CALC_PKCS, prikey_fid, 128, buff);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }

        /*      Now buff[0] should be length of plaintext, that is, 24.
                And buff[1..25] should be the same as plaintext. */

        _set_response(1, &ret);
        _exit();

}
```

_rsa_sign

RSA digital signature

```
BYTE _rsa_sign(
      BYTE   bMode,
      WORD   wFid,
      BYTE   bLen,
      BYTE   *pbData);
```

Parameters:

bMode [in] Signing mode:

RSA_CALC_NORMAL
Encrypt the HASH value directly using Private Key;

RSA_CALC_HASH
Do the HASH operation first for the inputted data and then encrypt the result using Private Key;

RSA_CALC_PKCS
Sign the HASH value according to PKCS#1 standard;

RSA_CALC_HASH|RSA_CALC_PKCS
Do the HASH operation first and then sign the HASH result according to PKCS#1 standard;

wFid [in] ID of Private Key file.

bLen [in] The data to be signed, having different requirement for different modes:

RSA_CALC_NORMAL (20 bytes)

RSA_CALC_HASH (1~128 bytes)

RSA_CALC_PKCS (20 bytes)

RSA_CALC_HASH|RSA_CALC_PKCS (1~128 bytes)

pbData [in, out] Plaintext or HASH value when input, 128 octets signature when output.

Return Values:
SES_SUCCESS or error code

Remarks:
It completes RSA Private Key signing operation, and the key length is 1024 bits while the signing result is 128 bytes. In the four modes above, RSA_CALC_PKCS is a standard digital signing scheme. If you want the signature has a good compatibility with other signing system, you must use this mode. If inputted plaintext is relatively short, you can use RSA_CALC_HASH|RSA_CALC_PKCS to do the hashing and signing together. The rest two are not recommended.

This SES supports only one HASH algorithm – SHA1. To use other HASH algorithm, use *_rsa_dec()* instead. For the generation and format of RSA keys, please refer to the “Remarks” section of *_rsa_enc()*.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:
Please refer to the sample code of *_rsa_veri()*.

_rsa_veri

It versifies RSA digital signature according to the HASH buffer and inputted signature. For detail, please refer to “Remarks” section of this table.

```
BYTE _rsa_veri(
      BYTE   bMode,
      WORD   wFid,
      BYTE   bLen,
      BYTE   *pbData);
```

Parameters:

bMode	[in] Calculation mode: RSA_CALC_NORMAL Decrypt the signature directly and compare with HASH buffer to verify; RSA_CALC_PKCS According to the Hash buffer and signature, do the standard PKCS#1 verification;
wFid	[in] ID of Public Key file.
bLen	[in] Length of digital signature must be 128.
pbData	[in] The signature.

Return Values:
SES_SUCCESS or error code

Remarks:
It completes RSA signature verification, and the key length is 1024 bits.

Different from *_rsa_sign()*, this verification function does not provide hashing calculation for the plaintext, you have to call corresponding hashing SES to do HASH first and then call this. Besides, this SES takes only HASH value from system's HASH buffer and cannot take directly inputted HASH value, and that means you cannot get HASH value of plaintext no matter what (i.e. do the hashing in software). If the plaintext is quite big, In this case, it is better to use *_rsa_enc()* to implement a verification for the HASH value in your own EXF .

This SES supports only one HASH algorithm – SHA1. To use other HASH algorithm, use *_rsa_enc()* instead.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
/* This demonstrates how to make a signature of a short
        message and then verify it later.
        The sample code supposes a pair of valid RSA key files
        have already been created.
*/

#include “ses_v3.h”
#include “string.h”

char message[24] = “This is some test data.”;
BYTE signature[128] = “”;
BYTE digest[20];
WORD pubkey_fid = 0xc001;
WORD prikey_fid = 0xc002;
HASH_CONTEXT hctx;

void main()
{
        BYTE ret = 0;

        /* Make signature… */
        memcpy(signature, message, 24);
```

```
        ret = _rsa_sign(RSA_CALC_HASH|RSA_CALC_PKCS, prikey_fid,
                                                        24, signature);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }

        /* Verify… */
        /* Hash the message first… */
        _sha1_init(&hctx);
        _sha1_update(&hctx, message, 24);
        _sha1_final(&hctx, digest);

        /* Start to verify… */
        ret = _rsa_veri(RSA_CALC_PKCS, pubkey_fid, 128, signature);
        if (ret != SES_SUCCESS)
        {
                _set_response(1, &ret);
                _exit();
        }

        /*      ret = SES_SUCCESS means the signature is valid. */

        _set_response(1, &ret);
        _exit();

}
```

_rand

Generates real random number by hardware

```
BYTE _rand(
        BYTE    bLen,
        BYTE    *pbData);
```

Parameters:

bLen	[in] Length of the random number to be generated, 1~255.
pbData	[out] Random number generated.

Return Values:
SES_SUCCESS or error code

Remarks:
Random number produced by this SES is hardware generated real one and needs no initialization.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
/* This demonstrates how to generate 128 bytes of random
        data and returned it to PC.
*/

#include “ses_v3.h”

/* Customized IO package encoding structure. */
typedef struct {
   /* Flag may be used for error-code or function code. */
        BYTE flag;
        /* Data length in buff. */
        BYTE len;
        /* Data transmitted in or out. */
        BYTE buff[1];
} IO_PACKAGE;

unsigned char tmp[128+2];
IO_PACKAGE *out = (IO_PACKAGE *)tmp;

void main()
{
        BYTE ret = 0;

        /* Generate random data… */
        ret = _rand(128, out->buff);
        if (ret != SES_SUCCESS)
        {
                out->flag = ret;
                out->len = 0;
```

```
                _set_response(2+out->len, (BYTE *)out);
                _exit();

        }

        /* Return random data... */
        out->flag = ret;
        out->len = 128;
        _set_response(2+out->len, (BYTE *)out);
        _exit();
}
```

Memory Operation

Functions of this category provide basic memory operations, i.e. memory copy, memory move etc. They are offered for the same purpose as single precision float function: to enhance the efficiency of code execution and decrease the size of objective executable. If the efficiency of execution and the size of objective executable do not matter, functions defined in *string.h*, for example, *memcpy()*, will be also available to use.

_mem_copy

Copy data between memory buffers, similar as *memcpy()*in standard C

```
BYTE _mem_copy(
      void   *pDest,
      void   *pSrc,
      BYTE   bLen);
```

Parameters:

pDest	[in] Destination address.
pSrc	[in] Source address.
bLen	[in] Length of the data to be copied.

Return Values:
SES_SUCCESS or error code

Remarks:
Different with *memcpy* in standard C, *_mem_copy* returns SES error code instead of a pointer to destination address.

Requirements:
Hardware Version: v2.xx, v5.xx

_mem_move

Move data from one buffer to another, allows overlapping of destination buffer and source buffer, similar as *memmove()* in standard C

```
BYTE _mem_move(
      void   *pDest,
      void   *pSrc,
      BYTE   bLen);
```

Parameters:

pDest	[in] Destination address.
pSrc	[in] Source address.
bLen	[in] Length of the data to be moved.

Return Values:
SES_SUCCESS or error code

Remarks:
Different with *memmove* in standard C, *_mem_move* returns SES error code instead of a pointer to destination address.

Requirements:
Hardware Version: v2.xx, v5.xx

_mem_set

Fill the buffer with the value specified, similar as *memset()* in standard C

```
BYTE _mem_set(
      void   *pDest,
      BYTE   c,
      BYTE   bLen);
```

Parameters:

pDest	[in] Destination address.
c	[in] Value to be filled in.
bLen	[in] Length of the buffer to be filled.

Return Values:
SES_SUCCESS or error code

Remarks:
Different with *memset()* in standard C, *_mem_set* returns SES error code instead of a pointer to destination address.

Requirements:
Hardware Version: v2.xx, v5.xx

_mempool_init

Initialize the starting address and size of memory pool, used for dynamic memory management. It must be called before any one of *_malloc()*, *_calloc() or _realloc()*

```
BYTE _mempool_init(
      void   *pStart,
      WORD   wSize);
```

Parameters:

pStart	[in] Starting address of memory pool must be a pointer to XRAM.
wSize	[in] Size of the memory pool.

Return Values:
SES_SUCCESS or error code

Remarks:
The internal resources in EL cannot be used as freely as in PC. Use static memory instead of dynamic means unless you are very familiar with memory structure and usage.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
_mempool_init((void xdata*)0x0400, 1024);
```

_malloc

Allocate a memory block in the memory pool set by *_mempool_init()*

```
void* _malloc(WORD wSize);
```

Parameters:
wSize [in] Size of the memory to be allocated.

Return Values:
Return pointer to the starting address of the memory block allocated if succeed, NULL otherwise.

Remarks:
Must call *_mempool_init()* first to set an available memory pool.

The internal resources in EL cannot be used as freely as in computer. Use static memory instead of dynamic means unless you are very familiar with memory structure and usage.

Requirements:
Hardware Version: v2.xx, v5.xx

_calloc

Allocate an array in the memory pool set by *_mempool_init()* with elements initialized to 0

```
void* _calloc(
      WORD   wNobj,
      WORD   wSize);
```

Parameters:

wNobj	[in] Number of the elements.
wSize	[in] Length in bytes of each element.

Return Values:
Return pointer to the starting address of the memory block allocated if succeed, NULL otherwise.

Remarks:
Must call *_mempool_init()* first to set an available memory pool.

The internal resources in EL cannot be used as freely as in computer. Use static memory instead of dynamic means unless you are very familiar with memory structure and usage.

Requirements:
Hardware Version: v2.xx, v5.xx

_realloc

Reallocate memory blocks in the memory pool set by *_mempool_init()*

```
void* _realloc(
      void   *pointer,
      WORD   wSize);
```

Parameters:

pointer	[in] Pointer to previously allocated memory block.
wSize	[in] New size in bytes.

Return Values:
Return pointer to the starting address of the memory block allocated if succeed, NULL otherwise.

Remarks:
Must call *_mempool_init()* first to set an available memory pool.

The internal resources in EL cannot be used as freely as in PC. Use static memory instead of dynamic means unless you are very familiar with memory structure and usage.

Requirements:
Hardware Version: v2.xx, v5.xx

_free

Free allocated memory block

```
BYTE _free(void *pointer);
```

Parameters:
pointer [in] Pointer to the memory block to be freed.

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

_invert

Invert the content of memory buffer

```
BYTE _invert(
      void   *pvdata,
      BYTE   bLen);
```

Parameters:

pvdata [in] Pointer to the memory buffer.

bLen [in] Length of the data to be inverted.

Return Values:
SES_SUCCESS or error code

Remarks:
This SES can invert the higher and lower part of a specified memory buffer, used for the conversion between data of Big-Endian and Little-Endian.

For example, data in a memory block are: *0x12, 0x34, 0x56, and 0x78*. After calling *_invert()*, they become to be: *0x78 0x56 0x34 0x12*.

Requirements:
Hardware Version: v2.xx, v5.xx

_mem_cmp

Compare the data stored in two memory buffer

```
char _mem_cmp(
      void   *pdest,
      void   *psrc,
      BYTE   length);
```

Parameters:

pdest [in] First buffer.

psrc [in] Second buffer.

bLen [in] Number of characters.

Return Values:
Return Values indicates the relationship between the buffers:

=-1 *pdest* less than *psrc*

= 0 *pdest* identical to *psrc*

= 1 *pdest* greater than *psrc*

Remarks:
The *_mem_cmp* function compares the first *bLen* bytes of *pdest* and *psrc* and returns a value indicating their relationship.

Requirements:
Hardware Version: v2.3

Time Control

Two time functions are provided in EL hardware: Timer and Real Time Clock. The former is a standard function for all EL devices, and the latter is supported only for specific version of EL with independent-powered clock.

Timer is substantively a 64 bits counter of CPU. You can use the current reading of the counter and the frequency of CPU to calculate the real elapsed time. Timer can be used only when EL is working properly. On the other hand, Real Time Clock is powered by independent battery; it can work finely even when EL is unplugged from computer.

_set_timer

Set timer's initial value and working mode

```
BYTE _set_timer(
      BYTE    bMode,
      DWORD   *pdwCount);
```

Parameters:

bMode [in] Timer mode:

0, Non-cycle mode, the timer stops when the counter number overflows

1, Cycle mode, the timer restarts when the counter number overflows

2, Cycle reset mode, the timer restarts from initial value after the counter number overflows

pdwCount [in] Pointer to the initial value.

Return Values:
SES_SUCCESS or error code

Remarks:
The timer uses counting-up mechanism.

Requirements:
Hardware Version: v2.xx, v5.xx

_start_timer

Start the timer according to the specified working mode and initial value

```
BYTE _start_timer();
```

Parameters:
None

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

_stop_timer

Stop the timer

```
BYTE _stop_timer();
```

Parameters:
None

Return Values:
SES_SUCCESS or error code

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

_get_timer

Get current reading of the timer

```
BYTE _get_timer(DWORD *pdwCount);
```

Parameters:

pdwCount	[out] Address of the variable for storing current reading of the timer.

Return Values:
SES_SUCCESS or error code

Remarks:
The timer uses counting up mechanism, and the counter increases itself by one for 64 CPU clock cycles. The CPU frequency, of before and including v2.3.2 EL is 16MHZ; of after v2.3.2 EL is 18MHZ; of after and including v2.3.6 EL is 24MHZ. The time span represented by one counter unit is:

1*64/16000000 = 4μs（before and including v2.3.2）

1*64/18000000 ≈ 3.6μs（after v2.3.4）

1*64/24000000 ≈ 2.7μs（after and including 2.3.6）

The counter number is stored in a DWORD variable, so the total time can be represented by the counter before its overflow is:

0xffffffff*4/1000000/3600 ≈ 4.77 hours（before and including v2.3.2）

0xffffffff*3.6/1000000/3600 ≈ 4.29 hours（after v2.3.4）

0xffffffff*2.7/1000000/3600 ≈ 3.21 hours（after and including 2.3.6）

It is available to use DevTest, or User Test tool to get the device version number, or go to the field dwVersion of the struct SENSE4_CONTEXT to get. For instance, if the hardware version is v2.4.4, the dwVersion is *0x00020304*. Note: the SENSE4 CONTEXT must be enumerated successfully. About the SENSE4 CONTEXT, please refer to the note of *S4Enum*.

Requirements:
Hardware Version: v2.xx, v5.xx

Example:

```
/* This demonstrate how to use a Timer to determine the running time of a function. */
#include “ses_v3.h”

DWORD tick_count = 0;
float time_used = 0;

void foo()
{
        /* Operations here…*/
}
```

```
void main()
{
	BYTE ret = 0;

	tick_count = 0;
	ret = _set_timer(1, &tick_count);
	if (ret != SES_SUCCESS)
	{
		_set_response(1, &ret);
		_exit();
	}

	ret = _start_timer();
	if (ret != SES_SUCCESS)
	{
		_set_response(1, &ret);
		_exit();
	}

	foo();

	ret = _get_timer(&tick_count);
	if (ret != SES_SUCCESS)
	{
		_set_response(1, &ret);
		_exit();
	}

	/* Get time used in millisecond. */
	time_used = (tick_count – 0) * 4/1000;

	_stop_timer();

	_set_response(1, &ret);
	_exit();

}
```

_time

Get current time, returning the number of seconds since midnight (00: 00: 00), January 1st, 1970, coordinated universal time. Similar as *time()* in standard C

```
BYTE _time(time_t *ptime);
```

Parameters:
Ptime [out] Address of the variable for holding the current time.

Return Values:
SES_SUCCESS or error code

Remarks:
The self-powered clock returns time of GMT format. If there is time difference problem, one may need to adjust it according to local time zone.

If there is no sufficient battery power, even after you may replace it with a new battery, this function will still return error. Initialization after battery replacement can only be done by EL manufacturer.

Different with *time()* in standard C, *_time()*returns SES error code instead of a time value. So please make sure the effectiveness of input pointer parameter (for returning data).

Requirements:
Hardware Version: v2.4 with real-time clock (RTC)

Example:
Please refer to the sample code *of_gmtime()*.

_mktime

Convert the local time (time structure) to a calendar value (*time_t*)

```
BYTE _mktime(
      time_t         *ptime,
      RTC_TIME_T     *ptm);
```

Parameters:

ptime [out] Pointer to the converted *time_t*.

ptm [in] Pointer to *a* RTC_TIME_T struct

Return Values:
SES_SUCCESS or error code

Remarks:
RTC_TIME_T struct definition:

```
typedef struct _RTC_TIME_T {
BYTE    second;  /* second, 0~59 */
BYTE    minute;  /* minute, 0~59. */
BYTE    hour;    /* hour, 0~24. */
BYTE    day;     /* day of month, 1~31. */
BYTE    week;    /* day of the week, Sunday is 0, Monday is 1, and so on. */
BYTE    month;   /* month, 0~11, January is 0 */
WORD    year;    /* year (100- 138, Year 2000 is 100. */
} RTC_TIME_T;
```

When converting from RTC_TIME_T to *time_t*, member variable *week* is ignored.

Different with *mktime()* in standard C, *_mktime()* does not return *time_t* but SES error code instead. Parameter *ptime* then is used to return the value of *time_t*.

Requirements:
Hardware Version: v2.4 with real-time clock (RTC)

Example:
Please refer to the sample code of *_gmtime()*.

_gmtime

Convert a *time_t* value to RTC_TIME_T struct

```
BYTE _gmtime(
      time_t         *ptime,
      RTC_TIME_T     *ptm);
```

Parameters:

ptime	[in] Pointer to *time_t* variable.
ptm	[out] Pointer to RTC_TIME_T struct, please refers to the Remarks section of this table.

Return Values:
SES_SUCCESS or error code

Remarks:
This function is for the convenience of handling time in EL internally. It request that the year represented by *time_t* is not earlier than A.D.2000, or it will lead to error.

Different with *gmtime()* in standard C, *_gmtime()* returns SES error code. Parameter *ptm* is used to return the RTC_TIME_T struct.

Requirements:
Hardware Version: v2.4 with real-time clock (RTC)

Example:

```
/* This demonstrate how to set an expiration date on specific function */

#include “ses_v3.h”

#define MY_SUCCESS 0x00
#define MY_ERROR_SES 0x01
#define MY_ERROR_EXPIRED 0x02

RTC_TIME_T exp = {0,0,0,20,0,11,105}; /* 2005.12.20 */
time_t et;

BYTE foo()
{
        BYTE ret = 0;
        time_t t = 0;

        /* Get current time. */
        ret = _time(&t);
        if (ret != SES_SUCCESS)
        {
                return MY_ERROR_SES;
        }

        /* Convert from RTC_TIME_T structure. */
        ret = mktime(&et, &exp);
        if (ret != SES_SUCCESS)
```

```
	{
		return MY_ERROR_SES;
	}

	/* Compare to expiration date. */
	if (t > et)
	{
		/* if you want to display current time, you can convert
		it to RTC_TIME_T structure. For example:
		RTC_TIME_T cur;
		…
		ret = _gmtime(&t, &cur);
		_set_response(sizeof(RTC_TIME_T), (BYTE *)&cur);
		_exit();
		*/
		return MY_ERROR_EXPIRED;
	}

	/* do something here. */

	return MY_SUCCESS;
}

void main()
{
	foo();

	_exit();
}
```

Macro and Auxiliary Function

For convenience of programming, EL provides some additional macros and auxiliary functions which are implemented by coding instead of being offered by EL operating system.

_swap_u16

This Macro is used to invert a 2 bytes memory

```
_swap_u16(__x__);
```

Parameters:
__x__ [in] Address of a 2 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro is kept for compatibility purpose only, try to use *invert()* instead. The latter is defined as follows:

```
#define _swap_u16(__x__) _invert(__x__, 2)
```

Requirements:
Hardware Version: v2.xx, v5.xx

_swap_u32

This Macro is used to invert a 4 bytes memory

```
_swap_u32(__x__);
```

Parameters:
__x__ [in] Address of a 4 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro is kept for compatibility purpose only, try to use *invert()* instead. The latter is defined as follows:

```
#define _swap_u32(__x__) _invert(__x__, 4)
```

Requirements:
Hardware Version: v2.xx, v5.xx

LE16_TO_CC

This macro converts a Little-Endian 2 bytes variable (i.e. *short*) to a byte sequence supported by current compiler

```
LE16_TO_CC(__x__);
```

Parameters:
__x__ [in] Address of a 2 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro can convert byte sequence automatically according to current compiler type. You can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other compilers use Little-Endian, this macro is equal to *_swap_u16()* when used in Keil C51.

Requirements:
Hardware Version: v2.xx, v5.xx

LE32_TO_CC

This macro converts a Little-Endian 4 bytes variable (i.e. *long*) to a byte sequence supported by current compiler

```
LE32_TO_CC(__x__);
```

Parameters:
__x__ [in] Address of a 4 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro can convert byte sequence automatically according to current compiler type. You can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while the other compilers use Little-Endian, this macro is equal to *_swap_u32()* when used in Keil C51.

Requirements:
Hardware Version: v2.xx, v5.xx

CC_TO_LE16

This macro, contrary to LE16_TO_CC, converts a 2 bytes variable to Little-Endian according to compiler type

```
CC_TO_LE16(__x__);
```

Parameters:
__x__ [in] Address of a 2 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro can convert byte sequence automatically according to current compiler type. You can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while other compilers use Little-Endian, this macro is equal to *_swap_u16()* when used in Keil C51 and does nothing with other compilers.

Requirements:
Hardware Version: v2.xx, v5.xx

CC_TO_LE32

This macro, contrary to LE32_TO_CC, converts a 4 bytes variable to Little-Endian according to compiler type

```
CC_TO_LE32(__x__);
```

Parameters:
__x__ [in] Address of a 4 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro can convert byte sequence automatically according to current compiler type. You can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while other compilers use Little-Endian, this macro is equal to *_swap_u32()* when used in Keil C51 and does nothing with other compilers.

Requirements:
Hardware Version: v2.xx, v5.xx

BE16_TO_CC

This macro converts a Big-Endian 2 bytes variable (i.e. *short*) to a byte sequence supported by current compiler

```
BE16_TO_CC(__x__);
```

Parameters:
__x__ [in] Address of a 2 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro can convert byte sequence automatically according to current compiler type. You can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while other compilers use Little-Endian, this macro does nothing when used in Keil C51 and equal to *_swap_u16()* with other compilers.

Requirements:
Hardware Version: v2.xx, v5.xx

BE32_TO_CC

This macro converts a Big-Endian 4 bytes variable (i.e. *long*) to a byte sequence supported by current compiler

```
BE32_TO_CC(__x__);
```

Parameters:
__x__ [in] Address of a 4 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro can convert byte sequence automatically according to current compiler type. You can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while other compilers use Little-Endian, this macro does nothing when used in Keil C51, and equal to *_swap_u32()* with other compilers.

Requirements:
Hardware Version: v2.xx, v5.xx

CC_TO_BE16

This macro, contrary to *BE16_TO_CC*, converts a 2 bytes variable to Little-Endian according to compiler type.

```
CC_TO_BE16(__x__);
```

Parameters:
__x__ [in] Address of a 2 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro can convert byte sequence automatically according to current compiler type. You can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while other compilers use Little-Endian, this macro does nothing when used in Keil C51, and it is equal to *_swap_u16()* when used in other compilers.

Requirements:
Hardware Version: v2.xx, v5.xx

CC_TO_BE32

This macro, contrary to *BE32_TO_CC*, converts a 4 bytes variable to Little-Endian according to compiler type

```
CC_TO_BE32(__x__);
```

Parameters:
__x__ [in] Address of a 4 bytes variable.

Return Values:
SES_SUCCESS or error code

Remarks:
This macro can convert byte sequence automatically according to current compiler type. You can write compiler-independent codes using this macro.

Because Keil C51 uses Big-Endian, while other compilers use Little-Endian, this macro does nothing when used in Keil C51, and it is equal to *_swap_u32()* when used in other compilers.

Requirements:
Hardware Version: v2.xx, v5.xx

_atod

Convert a string to double precision float

```
BYTE _atod(
      DOUBLE_T       *presult,
      char           *pstr);
```

Parameters:

presult	[out] Pointer to the double precision float struct.
Pstr	[in] String representation of a float, for example “3.1415926”.

Return Values:
SES_SUCCESS or error code

Remarks:
This function is provided with source code. Its header file and source code are *atod.h* and *atod.c,* please search those files under the same directory as *ses_v3.h*

Using this function will consume much system resource. If there is no too strict requirement for the initial value, try to use other system functions to assign a *double* float, for instance, *_ftod()*, or to initialize memory storage directly.

This function used to be named *_strtod()*, and is changed to be *_atod()* due to naming conflict in new version compiler.

Requirements:
Hardware Version: v2.4

DEFINE_AT

Define a variable at a specified address according to compiler type.

```
DEFINE_AT(TYPE, NAME, ADDRESS, MEMORY)
```

Parameters:

TYPE	[in] Type of the variable to be defined.
NAME	[in] Variable name.
ADDRESS	[in] Address of the variable specified.
MEMORY	[in] memory zone: RAM_EXT (xdata) RAM_INT_DE (data) RAM_INT_ID (idata) ROM (code memory)

Return Values:
None

Remarks:
None

Requirements:
Hardware Version: v2.xx, v5.xx

Get Device Info

A SES which can read basic hardware information is added since v2.3.2. It can read the GUSN (Global Unique Serial Number) and Developer ID (also called: Customer ID).

_get_gbdata

Read GUSN or Developer ID

```
BYTE _get_gbdata(
      BYTE bFlag,
      BYTE* pbData,
      BYTE bLen);
```

Parameters:

bFlag	[in] Flag: GLOBAL_SERIAL_NUMBER get the 8 octets GUSN GLOBAL_CLIENT_NUMBER get the 4 octets Developer ID
pbData	[out] Buffer for holding data read.
bLen	[in] size of the buffer *pbData*.

Return Values:
SES_SUCCESS or error code

Remarks:
Parameters *bLen* is used to represent the size of *pbData*. It is not necessary for its value to be big enough for holding all the data. For example, if you specify *bLen* =5 when reading hardware GUSN (8 octets), this function gets only first 5 bytes of GUSN and returns no error.

Requirements:
Hardware Version: v2.3.2

Compiler-dependent: Keil C51 only

Error Code List of SES

Error **C**ode	Value	**Definition**
SES_SUCCESS	0x00	Succeed
SES_PARA	0x02	Invalid parameter
SES_EEPROM	0x03	Error writing EEPROM
SES_RAM	0x04	Memory violation
SES_XCOS	0x05	Error from XCOS
SES_FILEID	0x11	Incorrect file ID, file not found
SES_FILE_ACCESS	0x12	Error accessing file
SES_FILE_SELECT	0x13	Error selecting file
SES_HANDLE	0x14	Invalid file handle
SES_RANGE	0x15	File out of range
SES_FILE_SPACE	0x16	Insufficient file storage
SES_FILE_EXISTING	0x17	File already exists
SES_KEYID	0x21	Error Key file ID
SES_KEY_ACCESS	0x22	Access to key file failed
SES_SHA1	0x23	SHA1 calculation error
SES_RAND	0x24	Error getting random number
SES_RSA	0x25	RSA calculation error
SES_RSAVERIFY	0x26	RSA verification failed, incorrect signature
SES_INVALID_POINTER	0x30	Invalid memory pointer
SES_INVALID_SIZE	0x31	Invalid memory size
SES_REAL_TIME_POWER	0x40	Error reading real time

Boundary Limitation of Double Precision Float

Under some marginal condition, double precision float calculation may have bias.

For functions marked with # in following table, the absolute value for inputting parameter must not be less than $1x10^{-154}$, (exclude 0), or it will exceed the boundary of float representable by EL. When the result exceeds the representable boundary, Return Value (presult) will NOT indicate being out of range.

Function Protocol	Boundary Condition Clarification
BYTE _mult(DOUBLE_T* presult, DOUBLE_T* px, DOUBLE_T* py)	#
BYTE _sin(DOUBLE_T* presult, DOUBLE_T* px)	# When the value of *px is close to multiple of π /2, there will be relatively big bias in result. When it is bigger than 1×10^{5}, the calculation precision will decrease to 9 effective digits due to accumulated bias.
BYTE _cos(DOUBLE_T* presult, DOUBLE_T* px)	# When the value of *px is close to multiple of π /2, there will be relatively big bias in result. When it is bigger than 1×10^{5}, the calculation precision will decrease to 9 effective digits due to accumulated bias.
BYTE _tan(DOUBLE_T* presult, DOUBLE_T* px)	# When the value of *px is close to multiple of π /2, there will be relatively big bias in result. When it is bigger than 1×10^{5}, the calculation precision will decrease to 9 effective digits due to accumulated bias.
BYTE _asin(DOUBLE_T* presult, DOUBLE_T* px)	# When the value of *px exceeds the domain of ±1 , Return Values indicates NO error.
BYTE _acos(DOUBLE_T* presult, DOUBLE_T* px)	# When the value of *px exceeds the domain of ±1 , Return Values indicates NO error.
BYTE _atan(DOUBLE_T* presult, DOUBLE_T* px)	#
BYTE _sinh(DOUBLE_T* presult, DOUBLE_T* px)	When the absolute value of *px is less than 1×10^{-8} (exclude 0), the calculation precision will decrease to 9 effective digits. The value of *px can NOT be greater than 709.
BYTE _cosh(DOUBLE_T* presult, DOUBLE_T* px)	When the absolute value of *px is less than 1×10^{-8} (exclude 0), the calculation precision will decrease to 9 effective digits. The value of *px can NOT be greater than 709.
BYTE _tanh(DOUBLE_T* presult, DOUBLE_T* px)	When the absolute value of *px is less than 1×10^{-8} (exclude 0), the calculation precision will decrease to 9 effective digits. The value of *px can NOT be greater than 709.
BYTE _atan2(DOUBLE_T* presult,	# The absolute value of *px/*py must between $1\times10^{-154}$and

DOUBLE_T* px, DOUBLE_T* py)	0(exclude 0)
BYTE _fmod(DOUBLE_T* presult, DOUBLE_T* px, DOUBLE_T* py)	There will be relatively big bias when the value of *px/*py is larger than 10^6
BYTE _exp(DOUBLE_T* presult, DOUBLE_T* px)	The value of *px cannot greater than 709
BYTE _pow(DOUBLE_T* presult, DOUBLE_T* px, DOUBLE_T* py)	Even when the scale of result is less than the representable boundary of double, it will NOT be set to zero.
BYTE _modf(DOUBLE_T* presult, DOUBLE_T* px, DOUBLE_T* intptr)	The absolute value of *px must between 1×10^{18} and 1×10^{-23}. (include 0)
BYTE _ldexp(DOUBLE_T* presult, DOUBLE_T* px, int exp)	Even when the scale of result is less than the representable boundary of double, it will NOT be set to zero.
BYTE _dtof(float* presult, DOUBLE_T* px)	Even when *px is greater than the representable boundary of double, return value indicates NO overflow. Even when *px is less than the representable boundary of double, it will NOT be set to zero.
BYTE _dtol(long* presult, DOUBLE_T* px)	Even when *px is greater than the representable boundary of long, return value indicates NO overflow. Even when *px is less than the representable boundary of long, it will NOT be set to zero.

API Reference 8

EL provides two types of API functions, for local and network operations respectively. Local APIs are used for local version, and for the local operations (initialization and development) made by network version; Network APIs are used for remote operation of network version.

Local API is sense4.dll, Network API is sense4user.dll, and both of them are offered with static lib as well.

The following table illustrates the relationship between API and the version of device.

	Desktop version EL	**Network version** EL
Local API	Applicable	Applicable, but limited to local operations such as initialization etc only.
Network API	Not applicable	Applicable, used for accessing EL device remotely in software.

According to the programming language used in your software, you can choose corresponding API library. SDK provides APIs of various language versions, i.e. C, Delphi, Java etc. To those languages not supported in SDK, you can use dll (dynamic link library) instead.

For better result in software protection, it's recommended to use static library of these programming languages. For better compatibility and more convenience in software upgrading, dll (dynamic link library) is recommended.

This chapter will further focus on the detail of API functions and when to use them. If you hope to understand the general procedure of accessing EL by API, please refer to the Chapter "Developing Codes".



Some APIs are used only in the stage of developing dongle applications, while some other APIs are used for accessing the dongle in the end-user environment. NEVER use API functions marked as "developer only" on the end-user side as shown in the following table.

API Function	Remarks	Privileges	**API Type**
S4Enum	List all the devices currently connected	Developer, User	Local & Network
S4Open	Connect the device with specified index	Developer, User	Local & Network
S4OpenEx	Connect the device specified using specific mode	Developer, User	Local
S4Close	Close connection to the device specified	Developer, User	Local & Network
S4Control	Send control codes to the device, for example, the lighting control of LED	Developer, User	Local & Network[8]
S4CreateDir	Create a new directory under current directory, it's recommended to use S4CreateDirEx instead.	Developer only	Local

[8] Local and Network API supports different Control Code, please check out note of *S4Control()*.

S4CreateDirEx	Create a new directory under current directory	Developer only	Local
S4ChangeDir	Change current working directory	Developer, User	Local & Network
S4EraseDir	Erase all the files and sub directories under the directory specified.	Developer only	Local
S4VerifyPin	Verify PIN code	Developer, User	Local & Network
S4ChangePin	Change PIN code	Developer only[9]	Local
S4WriteFile	File creation, update; RSA key pair generation, setup authorization	Developer only	Local
S4Execute	Execute VM EXF	User only	Local & Network
S4ExecuteEx	Execute VM EXF	User only	Local
PS4WriteFile	Similar to S4WriteFile,can operate disk file directly	Developer only	Local
S4Startup	Revoked	—	—
S4Clearup	Revoked	—	—

Table 7: List of API with Using Condition

[9] If it is to change User PIN, you can also use this function on the end-user side.

API List

This section will make detailed introduction to all APIs. To each API, we will illustrate its function for local version only.

S4Enum

Local: List all the available EL devices currently connected to the local computer, and store the device information obtained to *S4CtxList.*

```
DWORD WINAPI S4Enum(
      SENSE4_CONTEXT *pS4CtxList,
      DWORD *pdwSize);
```

Parameters:

pS4CtxList	[out] Pointer to SENSE4_CONTEXT struct array.
pdwSize	[in, out] The size of SENSE4_CONTEXT struct array (in byte) when input, length of the data actually returned when output. If *pS4CtxList*=NULL or the size of the provided struct array is insufficient, the function will return error and save the size of the storage actually needed to the variable specified by *pdwSize*.

Return values:
S4_SUCCESS or error code

Remarks:
Before calling any other functions, you must call this function to obtain device information. Information for each device is stored in a single struct SENSE4_CONTEXT which is defined as follows:

```
typedef struct {
       DWORD          dwIndex;
       DWORD          dwVersion;
       S4HANDLE       hLock;
       BYTE           reserve[12];
       BYTE           bAtr[MAX_ATR_LEN];
       BYTE           bID[MAX_ID_LEN];
       DWORD          dwAtrLen;
} SENSE4_CONTEXT, *P SENSE4_CONTEXT;
```

Definition of each member variable of this struct shown as follows below:

Variable	**Definition**	
dwIndex	Index for current device, begin with 0.	
dwVersion	EL hardware version number, currently support:	
	SENSE4_CARD_TYPE_V2_00	version 2.0
	SENSE4_CARD_TYPE_V2_01	version 2.1
	SENSE4_CARD_TYPE_V2_02	version 2.2
	SENSE4_CARD_TYPE_V2_30	version 2.3
	SENSE4_CARD_TYPE_V2_40	version 2.4
hLock	Device Handle.	
reserve	System reserved.	

bAtr	ATR information, system reserved.
bID	Device ID. If there is an ATR file in the hardware, then *bID* will hold a self defined ID (content of the ATR file), otherwise, it will hold the Global Unique Serial Number (GUSN) of the hardware. For detail information about ATR file, please refer to the Remark section of *S4CreateDirEx()*. For network API, *bID* will hold the ID of the master key on network server upon function return.
dwAtrLen	Length of ATR, system reserved.

If you don't know previously the size of SENSE4_CONTEXT struct array needed, you can set the input parameter *pS4CtxList* to be NULL, then the parameter *pdwSize* will return the memory storage actually needed.

If the system failed to connect EL, parameter *pdwSize* will return 0, and S4_SUCCESS.

The quantity of EL connected in current system can be calculated using the value returned by parameter *pdwSize* as follows:

```
Num = *pdwSize/sizeof(SENSE4_CONTEXT)
```

Requirement:
Hardware Version: v2.xx, v5.xx
Sample:
Please refer to the Section "General Level Access".

S4Open

Local: Connect to the specified EL device. Connection must be established before any access can be made

```
DWORD WINAPI S4Open(SENSE4_CONTEXT *pS4Ctx);
```

Parameters:

pS4Ctx	[in, out] Pointer to the SENSE4_CONTEXT struct of the EL device to be opened.

Return values:
S4_SUCCESS or error code

Remarks:
Use struct SENSE4_CONTEXT returned from *S4Enum()* as input parameter to establish a connection to this device.

To local API, this function connects to the device using share mode, namely, device connected supports multiple-process access;

If there are multiple EL devices in local system, it is very important to assure that correct device is opened. Two important reasons: One is that if connected to the wrong device, the software will mistakenly recognize the device is illegal and thus result in the software execution error; the other one is that operating on a device of other software will change the status of that dongle and thus influence the normal execution of that software. To avoid these problems and open correct dongle, one can take the following two commonly used approaches:

Method	Detail
Create and manage customized Device ID	This is the most effective method. When creating the root directory of a dongle, you can create a self-defined ATR file, content of which contains self-defined 8 octets Device ID. When running *S4Enum()*to list devices, IDs of all the dongles will be stored in the *bID* member variables of SENSE4_CONTEXT struct array, and thus you can distinguish different devices according to different IDs. For the details regarding how to create ATR file, please refer to the Remarks section of *S4CreateDirEx()*.
Use Developer ID （Customer ID）	EL offers the service of customizing Developer ID, which means EL can allocate different Developer ID for different software vendors. This ID has no relation with GUSN or self-defined device ID and it can be acquired by *S4Control()*. Software can distinguish different devices according to different Developer ID. For details regarding the information and usage of Developer ID, please refer to the Remarks section of *S4Control()*.

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:
Please refer to the Section “General Level Access”.

S4OpenEx

Local: Connect to the specified EL device using specified mode. Connection must be established before any access can be made. This function is the upgrade version of *S4Open()*.

```
DWORD WINAPI S4OpenEx(
        SENSE4_CONTEXT *pS4Ctx,
        S4OPENINFO *pS4OpenInfo);
```

Parameters:

pS4Ctx	[in, out] Pointer to the SENSE4_CONTEXT struct of the EL device to be opened. The content of the struct is returned by *S4Enum()*.
pS4OpenInfo	[in] Pointer to the S4OPENINFO struct, describing the way how the device is to be opened.

Return values:
S4_SUCCESS or error code

Remarks:
The major difference between this function and *S4Open()*is: *S4Open()*connects the EL device using share mode, namely, after a process has already connected to the EL device, other processes can still connect and make access to that device; *S4OpenEx()*,however, can specify the connection mode. When the device is opened using exclusive mode, other process is not able to connect to the same device concurrently.

S4OPENINFO struct is defined as follows:

```
typedef struct _S4OPENINFO {
        DWORD dwS4OpenInfoSize;
        DWORD dwShareMode;
} S4OPENINFO;
```

Explanation for each member variable of this struct follows below:

Variable	Definition
dwS4OpenInfoSize	Must be *sizeof (*S4OPENINFO*)*.
dwShareMode	Device connection mode, two modes are currently supported: S4_EXCLUSIZE_MODE S4_SHARE_MODE

For more, please refer to the remarks section of *S4Open()*.

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:

```
/* This demonstrates how to open EL in exclusive mode.*/

        ...
        SENSE4_CONTEXT s4ctx = {0};
        S4OPENINFO oinf = {0};
        DWORD ret = 0;
```

```
/* Enumerate device here… */

/* Open in exclusive mode… */
oinf.dwS4OpenInfoSize = sizeof(S4OPENINFO );
oinf.dwShareMode = S4_EXCLUSIZE_MODE;
ret = S4OpenEx(&s4ctx, &oinf);
…
```

S4Close

Local: Close connection to the specified device

```
DWORD WINAPI S4Close(SENSE4_CONTEXT *pS4Ctx);
```

Parameters:

pS4Ctx	[in, out] Pointer to SENSE4_CONTEXT struct, pointing to an opened device.

Return values:
S4_SUCCESS if the function succeeds or corresponding error code otherwise.

Remarks:
This function closes the connection to the device opened by *S4Open()*or *S4OpenEx()*.

Please notice that this function does NOT clear the security status of EL device, it only closes the connection to that device. If you want to clear the security status of the EL device utterly, please use *S4Control()* to send S4_RESET_DEVICE control code to the device before calling this function.

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:
Please refer to the Section “General Level Access”.

S4Control

Local: Sending control code to EL device.

```
DWORD WINAPI S4Control(
      SENSE4_CONTEXT *pS4Ctx,
      DWORD dwCtlCode,
      VOID *pInBuff,
      DWORD dwInBuffLen,
      VOID *pOutBuff,
      DWORD dwOutBuffLen,
      DWORD *pdwBytesReturned);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
dwCtlCode	[in] Control code. For currently available values.
pInBuff	[in] Pointer to the input buffer, storing additional data for the control code.
dwInBuffLen	[in] Length of the data in input buffer *pInBuff*.
pOutBuff	[out] Pointer to the output buffer, storing the returned information from device after sending the control code.
dwOutBuffLen	[in] Size of output buffer.
pdwBytesReturned	[out] Address for a DWORD variable, saving length of the data actually returned to *pOutBuff*, cannot be NULL.

Return values:
S4_SUCCESS or error code

Remarks:
Explanation for control codes currently supported by EL:

The following control parameters are only supportive in local API.

Parameter	Attributes
S4_LED_UP	Purpose: LED on.
	Input: None
	Output: None
S4_LED_DOWN	Purpose: LED off.
	Input: None
	Output: None
S4_LED_WINK	Purpose: LED blinks.
	Input: 1 byte integer, standing for the times of LED winking per second.
	Output: None
S4_GET_DEVICE_TYPE	Purpose: To get device type.
	Input: None

	Output: 1 byte data, indicating device type: local version, network version master, or network version slave
S4_GET_SERIAL_NUMBER	Purpose: To get Global Unique Serial Number (GUSN).
	Input: None
	Output: 8 bytes GUSN.
S4_GET_VM_TYPE	Purpose: To get VM type. This control code has been revoked.
	Input: None
	Output: 1 byte VM type
S4_GET_DEVICE_USABLE_SPACE	Purpose: To get usable space of the device.
	Input: None
	Output: 2 bytes WORD integer, standing for the size of usable device space.
S4_SET_DEVICE_ID	Purpose: Set ATR information, namely self-defined ID, developer-level authorization is required.
	Input: 8 bytes self-defined ID.
	Output: None
S4_RESET_DEVICE	Purpose: Reset device thus to clear current security status.
	Input: None
	Output: None
S4_DF_AVAILABLE_SPACE	Purpose: To get available space of current directory.
	Input: None
	Output: 2 bytes WORD integer, standing for the size of unused space under current directory.
S4_EF_INFO	Purpose: To get the attribute information of one file under current directory.
	Input: None
	Output: EFINFO struct, content of which will be illustrated later in this table.
S4_SET_USB_MODE	Purpose: Set current device to run on standard USB mode, supported only for EL 2.3 or above.
	Input: None
	Output: None
S4_SET_HID_MODE	Purpose: Set current device to run on standard HID mode, supported only for EL 2.3 or above. This mode makes it unnecessary to install driver under Windows98 or above, but has lower capability and efficiency than standard USB mode. So it is not recommended.
	Input: None
	Output: None
S4_GET_CUSTOMER_NAME	Purpose: To get Developer ID (Customer ID) of current device, supported only for EL 2.1 or above.
	Input: None
	Output: 4 bytes Developer ID.
S4_GET_MANUFACTURE_DATE	Purpose: To get manufactured date of current device, supported only for EL 2.1 or above.
	Input: None
	Output: S4_MANUFACTURE_DATE struct, content of which will be illustrated later in this table.
S4_GET_CURRENT_TIME	Purpose: To get the current time of the hardware clock, supported only for EL 2.3 or above devices with realtime clock.
	Input: None
	Output: a tm struct, which has the same definition as tm struct

	in C standard running library. For detail, please refer to the definition below or the exemplifications in MSDN.
S4_SET_NET_CONFIG	Purpose: After invoking *S4CreateDir*, *S4CreateDirEx* to create a network dongle and storing module authorization in the device memory, it is available to use *S4Control*. By using this parameter, modify the authorization or mode of module in network dongle.
	Input: Point to the handle of S4NETCONFIG struct.
	Output: None

EFINFO struct is defined as follows:

```
typedef struct{
	WORD EfID;
	BYTE EfType;
	WORD EfSize;
} EFINFO, *PEFINFO;
```

Member variables:

Variable	Definition
EfID	File ID.
EfType	File type
EfSize	File size.

tm struct is defined as follows:

```
struct tm {
	int tm_sec;	/* seconds after the minute - [0,59] */
	int tm_min;	/* minutes after the hour - [0,59] */
	int tm_hour;	/* hours since midnight - [0,23] */
	int tm_mday;	/* day of the month - [1,31] */
	int tm_mon;	/* months since January - [0,11] */
	int tm_year;	/* years since 1900 */
	int tm_wday;	/* days since Sunday - [0,6] */
	int tm_yday;	/* days since January 1 - [0,365] */
	int tm_isdst;	/* daylight savings time flag */
};
```

S4_MANUFACTURE_DATE struct is defined below:

```
typedef struct {
	WORD	wYear;
	BYTE	byMonth;
	BYTE	byDay;
}S4_MANUFACTURE_DATE;
```

Variable	Definition
wYear	Year.
byMonth	Month 1~12.
byDay	Day, 1~31.

S4NETCONFIG struct is defined as follows:

```
typedef struct S4NETCONFIG {
DWORD	dwLicenseMode;
DWORD	dwModuleCount;
S4MODULEINFO  ModuleInfo[64];
```

```
} S4NETCONFIG;
```

Variable	Definition
dwLicenseMode	Assign the mode of module authorization. Two modes: Process mode, in obtaining any modules authorization, it will take authorization of this module; IP mode, then any authorizations obtained by same module on the same machine only take as one authorization by sharing.
dwModuleCount	Assigned module quantity. Max. 64.
ModuleInfo	Info in detail of assigned module. Please refer to S4MODULEINFO structural definition.

S4MODULEINFO struct is defined as follows:

```
typedef struct S4MODULEINFO {
 WORD        wModuleID;
 WORD        wLicenseCount;
} S4MODULEINFO;
```

Variable	Definition
wModuleID	Module ID (0-65535)
wLicenseCount	Total amount of authorization of assigned module (0-255)

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:
Please refer to the Section “General Level Access”.

S4CreateDir

Local: Create a new directory. If creation succeeds, the newly created directory will be selected to be “current directory”. Using *S4CreateDirEx()*instead of this function is recommended.

```
DWORD WINAPI S4CreateDir(
      SENSE4_CONTEXT *pS4Ctx,
      LPCSTR lpszDirID,
      DWORD dwDirSize,
      DWORD dwFlags);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
lpszDirID	[in] Directory ID. In EL API, ID of file or directory is passed in using a string. For example, string “1234”means that the ID of directory or file is *0x1234*. Please notice that the inputted ID must contain 4 chars and be in hexadecimal format. For example, to the directory ID *0x0012*, you must input “0012” instead of “12”. For network version, the parameter can be only set as “\”, to create root directory only but others. For system reserved IDs, please refer to the Remarks section of this table.
dwDirSize	[in] Size of the directory to be created, cannot exceed the usable space of current directory. When creating directory, the file system itself will take certain amount of space: for local version, it will take 197 bytes. For example, if one specifies 1024 bytes as the total space when creating the directory, the actual usable space after creation will be: local version(1024-197)=827. Additionally, directory’s file header will also take certain amount of its parent directory’s space (42 bytes).For example, if one creates a sub directory of 1024 bytes under root directory; it actually takes 1024+42=1066 bytes space of the root directory totally.
dwFlags	[in] Flag options for creating a directory. S4_CREATE_ROOT_DIR Create root directory. When this flag is set, directory ID must be ”\”,*dwDirSize* must be 0,or there will be parameter error. S4_CREATE_SUB_DIR Create sub directory, effective only for local version. S4_CREATE_MODULE Create a sub module, effective only for network version. Sub module of network version is like a sub directory, it

can ONLY be created directly under the root directory

After the sub module is created, its license number will be set to be 10 automatically. You can use *S4WriteFile()*to modify the license number.

Return values:
S4_SUCCESS or error code

Remarks:
This function does NOT create ATR file automatically. For details regarding ATR file, please refer to the Remarks section of *S4CreateDirEx()*.

After a directory is created, its Developer PIN and User PIN will be set to be default value, namely, Developer PIN: ”123456781234567812345678”, User PIN: ”12345678”, you MUST modify the Developer PIN to be a secret value.

After using this function to create root directory in network version, the total authorization of the device is 10, authorization mode will be Process mode, 3 modules by default, 5 authorizations for each module (ID: 0, 1, 2)

For local version, max 3 level file structures can be created (root directory being taken as level one, like *0001**0002*).

Some IDs are reserved by system, these include: *0x0000*, *0x3f00* (reserved as ID of root directory).

The ID of any directory’s parent directory, sibling directory and subdirectory cannot be same.

For network version, the function is void invoked and return S4_SUCCESS.

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:
Please refer to the Section “Developer Level Access”.

S4CreateDirEx

Local: Create a new directory. If creation succeeds, the newly created directory will be selected to be “current directory”.

```
DWORD WINAPI S4CreateDirEx(
      SENSE4_CONTEXT *pS4Ctx,
      LPCSTR lpszDirID,
      DWORD dwDirSize,
      DWORD dwFlags,
      S4CREATEDIRINFO *pCreateDirInfo);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
lpszDirID	[in] Directory ID. In EL API, ID of file or directory is passed in using a string. For example, string “1234”means that the ID of directory or file is *0x1234*. Please notice that inputted ID must contain 4 chars and be in hexadecimal format. For example, to the directory ID *0x0012*, you must input “0012” instead of “12”. For network version, the parameter can be only set as “\”, to create root directory only but others. For system reserved IDs, please refer to the Remarks section of this table.
dwDirSize	[in] Size of the directory to be created, cannot exceed the remaining space of current directory. When creating directory, the file system itself will take certain amount of space: for local version, it will take 197 bytes. For example, if one specifies 1024 bytes as the total space when creating the directory, the actual usable space after creation will be: local version(1024-197)=827. Additionally, directory’s file header will also take certain amount of its parent directory’s space (42 bytes).For example, if one creates a sub directory of 1024 bytes under root directory; it actually takes 1024+42=1066 bytes space of the root directory totally.
dwFlags	[in] Flag options for creating a directory. S4_CREATE_ROOT_DIR Create root directory. When this flag is set, directory ID must be ”\”,*dwDirSize* must be 0,or there will be parameter error. S4_CREATE_SUB_DIR Create sub directory, effective only for local version.

	S4_CREATE_MODULE Create a sub module, effective only for network version. Sub module of network version is like a sub directory, it can ONLY be created directly under the root directory After the sub module is created, its license number will be set to be 10 automatically. You can use *S4WriteFile()*to modify the license number.
pCreateDirInfo	[in] This parameter is a pointer to the S4CREATEDIRINFO struct. You must fill in this struct if *dwFlags* is set to be S4_CREATE_ROOT_DIR .

Return values:
S4_SUCCESS or error code

Remarks:
Using this API to create root directory will automatically create an ATR file which is used to store self-defined ID (the content of this ID is from the member variable *szAtr[8]* of struct *pCreateDirInfo*).

ATR file is of special type, and its content has only 8 bytes. If this file exists, then when using *S4Enum()*to list devices, *bID* member variable of SENSE4_CONTEXT struct will be the 8 octets content of ATR file. If you want the real 8 octets GUSN, you must use *S4Control()*to send a S4_GET_SERIAL_NUMBER control code. Using ATR file can well distinguish devices from different developers. For detail, please refer to the Remarks section of *S4Enum()*.

After a directory is created, its Developer PIN and User PIN will be set to be default value, namely, Developer PIN: ”123456781234567812345678”, User PIN: ”12345678”, you MUST modify the Developer PIN to be a secret value.

Some IDs are reserved by system, these include: *0x0000*, *0x3f00* (reserved as ID of root directory).

The ID of any directory’s parent directory, sibling directory and subdirectory cannot be same.

S4CREATEDIRINFO struct is defined as follows:

```
typedef struct _S4CREATEDIRINFO {
       DWORD dwS4CreateDirInfoSize;
       BYTE  szAtr[8];
 S4NETCONFIG NetConfig;
} S4CREATEDIRINFO;
```

Variable	Definition
dwS4CreateDirInfoSize	Size of the struct, must be: *sizeof(*S4CREATEDIRINFO*)*.
szAtr	8 octets content of ATR file, namely self-defined ID.
NetConfig	S4NETCONFIG struct, please refer to the note of S4Control about S4NETCONFIG. It should be regardless to network version.

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:

```
/* This demonstrates how to create root dir with developer
```

```
specific identifier. */

…
SENSE4_CONTEXT s4ctx = {0};
S4CREATEDIRINFO rinf = {0};
DWORD ret = 0;

/* Open and verify Developer PIN here. Delete original root
dir if exists… */

/* Create new root dir… */
rinf.dwS4CreateDirInfoSize = sizeof(S4CREATEDIRINFO);
memcpy(rinf.szAtr, “Sense-EL”, 8);
ret = S4CreateDirEx(&s4ctx, “\\”, 0,

S4_CREATE_ROOT_DIR,&rinf);
…
```

S4ChangeDir

Local: Select a new working directory.

```
DWORD WINAPI S4ChangeDir(
      SENSE4_CONTEXT *pS4Ctx,
      LPCSTR lpszPath);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
lpszPath	[in] New working directory path. There are two ways for path selecting: Absolute path, based on the root directory, for example "*0001**0012*", "\", "*0011*" etc; Relative path, based on current directory, for example "*0001**0012*", "*00e4*". For local operations of network version, as only root directory available, this function is only way to switch the root directory.

Return values:
S4_SUCCESS or error code

Remarks:
When change to another directory of upper level or the same level, the security status of the original directory will be lost. Re-authentication is required if you want to re-select the original directory.

The ID of any directory's parent directory, sibling directory and subdirectory cannot be same.

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:
Please refer to the Section "General Level Access".

S4EraseDir

Local: Erase all the content of current directory, and reset its security attributes.

```
DWORD WINAPI S4EraseDir(
      SENSE4_CONTEXT *pS4Ctx,
      LPCSTR lpszPath);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
lpszPath	[in] Reserved, must be NULL.

Return values:
S4_SUCCESS or error code

Remarks:
If current directory is a sub directory, this function will erase all the files and children directories of current directory (current directory itself will NOT be deleted). The Developer PIN and User PIN of current directory will also be restored to default values.

If current directory is a root directory, this function will delete the root directory directly, and that will make the dongle to get back to empty status.

The ID of any directory's parent directory, sibling directory and subdirectory cannot be same.

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:
Please refer to the Section "Developer Level Access".

S4VerifyPin

Local: Verify PINs (Developer or User) of current directory so as to get corresponding access privileges.

```
DWORD WINAPI S4VerifyPin (
      SENSE4_CONTEXT *pS4Ctx,
      BYTE *pbPin,
      DWORD dwPinLen,
      DWORD dwPinType);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
pbPin	[in] Pointer to the PIN array.
dwPinLen	[in] Length of the PIN, Developer PIN is 24 octets, and User PIN is 8 octets.
dwPinType	[in] PIN type to be verified: S4_DEV_PIN Verify Developer PIN S4_USER_PIN Verify User PIN

Return values:
S4_SUCCESS or error code

Remarks:
EL may be locked after several times of failed retries in verifying developer level PIN. If the hardware is locked, no one can unlock it.

To avoid accidentally being locked, the Developer PIN of EL takes measures of "blocking strategy": If there are errors in the former 16 bytes of the Developer PIN, it won't result in being locked. If the former 16 bytes of the Developer PIN is correct, and there are errors in the latter 8 bytes, several attempts will make EL to be locked. In the latter case, function's return value will be: *0x63CX*, in which X is between 0~F (means the remaining times of retrials allowed before the EL's being locked) For example, if the function returns 0x63CA, it means that 10 attempts can be made before being locked.

No matter how many failed attempts have been made to verify User PIN, it NEVER locks the dongle.

All the PINs are effective only for current directory, for details, please refer to the Section "PIN".

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:

Please refer to the Section “Developer level access” and Section “User level access”.

S4ChangePin

Local: Change PINs (Developer or User) of current directory.

```
DWORD WINAPI S4ChangePin (
      SENSE4_CONTEXT *pS4Ctx,
      BYTE *pbOldPin,
      DWORD dwOldPinLen,
      BYTE *pbNewPin,
      DWORD dwNewPinLen,
      DWORD dwPinType);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
pbOldPin	[in] Pointer to the old PIN
dwOldPinLen	[in] Length of the old PIN, Developer PIN is 24 octets, and User PIN is 8 octets;
pbNewPin	[in] Pointer to the new PIN.
dwNewPinLen	[in] Length of the new PIN, Developer PIN is 24 octets, and User PIN is 8 octets;
dwPinType	[in] Type of the PIN to be modified: S4_DEV_PIN change developer level PIN S4_USER_PIN change user level PIN

Return values:
S4_SUCCESS or error code

Remarks:
This function will automatically verify the old PIN so as to get corresponding security authorization, so all the rules regarding PIN locking in the Remarks section of S4VerifyPin() apply to this function as well.

After PIN changed, corresponding security authorization will be obtained automatically; PIN changing can be done only at the same level, namely, Developer PIN cannot be used to modify User PIN.

All the PINs are effective only on current directory. For details, please refer to the Section “PIN”.

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:
Please refer to the Section “Developer Level Access”.

S4WriteFile

Local: Create, update file or modify license number etc. under current directory.

```
DWORD WINAPI S4WriteFile (
      SENSE4_CONTEXT *pS4Ctx,
      LPCSTR lpszFileID,
      DWORD dwOffset,
      LPVOID lpBuffer,
      DWORD dwBufferSize,
      DWORD dwFileSize,
      DWORD *pdwBytesWritten,
      DWORD dwFlags,
      BYTE bFileType);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
lpszFileID	[in] File ID, there are special stipulation when *dwFlags* is any of the following values: S4_KEY_GEN_RSA_FILE Need to input IDs of two files, 8 bytes in total, and the former 4 bytes is the ID of a public file and the latter 4 bytes is the ID of corresponding private file. S4_SET_LICENCES Must be NULL.
dwOffset	[in] Offset for file writing, when *dwFlags* is any of the following values, it must be 0: S4_KEY_GEN_RSA_FILE S4_SET_LICENCES S4_RSA_PUBLIC_FILE S4_RSA_PRIVATE_FILE
lpBuffer	[in] Data content to be written in. Data format has relation with file type. Please refer to explanation for the file types in the Remarks section of this table. When *dwFlags* is any of the following values, it must be NULL; S4_KEY_GEN_RSA_FILE S4_SET_LICENCES When *dwFlags* is any of the following values, you need to input keys with required format. For details, please refer to the Remarks section of this table: S4_RSA_PUBLIC_FILE

S4_RSA_PRIVATE_KEY

dwBufferSize [in] Length of the data to be written into the file. There are special stipulation when *dwFlags* is any of the following values:

S4_KEY_GEN_RSA_FILE
dwBufferSize = 0

S4_SET_LICENCES
dwBufferSize = 0

S4_RSA_PUBLIC_FILE
dwBufferSize = sizeof(S4_RSA_PUBLIC_KEY)

S4_RSA_PRIVATE_FILE
dwBufferSize = sizeof(S4_RSA_PRIVATE_KEY)

dwFileSize [in] Size of the file to be created or license number, depending on the value of *dwFlags*:

S4_CREATE_NEW
dwFileSize is the size of the file to be created. Please notice that, to public file, it must be 136; to private file, it must be 330.

S4_SET_LICENCES
dwFileSize is the license number to be set, must be between 1 and 255.

Else
Must be 0.

pdwByteWritten [out] Length of data actually written(in byte)

dwFlags [in] Flag options, please see Remarks section for details.

bFileType [in] File type, please see Remarks section for details.

Return values:
S4_SUCCESS or error code

Remarks:
Available values for flag option(*dwFlags*)are listed below:

Variable	Definition
S4_CREATE_NEW	Create new file. By default, attribute of the newly created executable file will be set to “internally read/writable”.
S4_FILE_EXECUTE_ONLY	Combined with flag *S4_CREATE_NEW* to create an “internally unreadable/unwritable” executable file, this flag is effective only on EXF.
S4_CREATE_PEDDING_FILE	For EL v 2.3 or higher, this flag is ignored.
S4_UPDATE_FILE	Update an existing file.
S4_KEY_GEN_RSA_FILE	Generate a RSA key pair, can be used together with *S4_CREATE_NEW*
S4_SET_LICENCES	Set license number for network version dongles.

* “Internally readable/writable” and “internally unreadable/unwritable” mean

whether the objective file can be operated by SES *_read()*and *_write()* or not. For detail, please refer to the Section Read-write Attributes".

Available values for file type(*bFileType*)are listed below:

Variable	Definition
S4_EXE_FILE	VM executable, binary objective file generated by Keil C51. Data process: Write the data in *lpbuffer* into the objective file using binary format.
S4_DATA_FILE	Data file Data process: Write the data in *lpbuffer* into the objective file using binary format.
S4_RSA_PUBLIC_FILE	RSA Public Key file*. Data process: Write the data in *lpbuffer* into the objective file after conversion, see the comments below this table
S4_RSA_PRIVATE_FILE	RSA private file. Data process: Write the data in *lpbuffer* into the objective file after conversion, see the comments below this table.
0	If *dwFlags* is *S4_KEY_GEN_RSA_FILE* or *S4_SET_LICENCES*,*bFileType* must be 0

* When writing Public Key or Private Key file, API does NOT write the inputted data to the file using binary format directly, instead, it converts the inputted data content first. The inputted data must be of Public Key S4_RSA_PUBLIC_KEY format and of Private Key S4_RSA_PRIVATE_KEY format as defined in EL API's header file; To files of other type, the function write directly the inputted data in binary format.

A file cannot be erased directly once it has been created. If you want to delete a file and free the storage space it takes completely, you must delete or empty the directory under which the file resides.

Some IDs are reserved by system, these include: *0x0000, 0x0015, 0x0016,0x0018,0x001e,0x3f00,0x3f01,0x3f02,0x3f03,0x3f04*

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:

```
/* This demonstrates how to create a 1K bytes data file
        and write       some information into it. */

        ……
        SENSE4_CONTEXT s4ctx = {0};
        unsigned char buff[] = "This is a test message!";
        DWORD dwRet = 0;
        DWORD dwBytesWritten = 0;

        /* Open and verify Developer PIN here… */
```

```
        /* Now create a new data file and write it. */
        dwRet = S4WriteFile(
                        &s4ctx,
                        “d001”,         /* New file ID. */
                        0,                              /* Write offset. */
                        buff,                   /* Data to write. */
                        sizeof(buff),   /* Number of bytes to write. */
                        1024,                   /*  New  file  size,  1K  bytes
here. */
                        &dwBytesWritten,
                        S4_CREATE_NEW,/*  Specify  to  create  a  new  file.
*/
                        S4_DATA_FILE); /* File type. */

        if (dwRet != S4_SUCCESS)
        {
                /* Handle error here… */
        }
        ……
```

PS4WriteFile

Local: Create or update a file under current directory. This API encapsulate *S4WriteFile()*and operate directly the disk file which simplifies greatly the process of initializing EL file system. All the work done by this API can also be achieved by *S4WriteFile()*.

This function is not a fundamental EL API, so only C version and Win32 dynamic link library of it are provided. It's declared in the file *psense4.h*.

```
DWORD WINAPI PS4WriteFile (
      SENSE4_CONTEXT *pS4Ctx,
      LPCSTR lpszFileID,
      LPCSTR lpszPCFilePath,
      DWORD *dwFileSize,
      DWORD dwFlags,
      DWORD dwFileType,
      DWORD *pdwBytesWritten);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
lpszFileID	[in] File ID.
lpszPCFilePath	[in] The path of disk file; and the content of the file is the data to be written into the dongle. There are different requirements for the format of disk file according to different objective file types, please refer to the explanation for file types in Remarks section of this table. If you want just to create a new file without writing in any content, you can set this parameter to be NULL.
dwFileSize	[in, out] Size of the file to be created. If you want to determine automatically the size according to the type and content of disk file, you can set this parameter to be 0; If you want to update an existing file, this parameter MUST be set to 0.
dwFlags	[in] Flag options, please see Remarks section for details.
bFileType	[in] File type, please see Remarks section for details.
pdwByteWritten	[out] Length of data actually written(in byte)

Return values:
S4_SUCCESS or error code

Remarks:
Available values for flag option(*dwFlags*)are listed below:

Variable	Definition
S4_CREATE_NEW	Create new file. By default, attribute of the newly created executable file will be set to "internally read/writable".
S4_FILE_EXECUTE_ONLY	Combined with flag S4_CREATE_NEW to

	create an “internally unreadable/unwritable” executable file, this flag is effective only to executable files.
S4_CREATE_PEDDING_FILE	For EL v 2.3 or higher, this flag is ignored.
S4_UPDATE_FILE	Update an existing file.

* “Internally readable/writable” and “internally unreadable/unwritable” mean whether the objective file can be operated by SES *_read()*and *_write()* or not. For detail, please refer to the Section Read-write Attributes”.

Available values for file type(*bFileType*) are listed below:

Variable	**Definition**
S4_EXE_FILE	VM executable, binary objective file generated by Keil C51. Data process: Write the data in disk file into the objective file using binary format.
S4_HEX_FILE	VM executable, binary objective file generated by Keil C51. The difference between this value and S4_EXE_FILE is that when this file type is specified, the disk file inputted will be of HEX format and API will internally convert it to binary format automatically. Data process: Convert the data in disk file and then write them into the objective file.
S4_DATA_FILE	Data file Data process: Write the data in disk file into the objective file using binary format.
S4_RSA_PUBLIC_FILE	RSA Public Key file, the data contained in the disk file must be of Public Key *S4_RSA_PUBLIC_KEY* format as defined in EL API’s header file; Data process: Convert the data in disk file and then write them into the objective file.
S4_RSA_PRIVATE_FILE	RSA private file, the data contained in the disk file must be of Private Key *S4_RSA_PRIVATE_KEY* format as defined in EL API’s header file; Data process: Convert the data in disk file and then write them into the objective file.

A file cannot be erased directly once it has been created. If you want to delete a file and free the storage space it takes completely, you must delete or empty the directory under which the file resides.

Some IDs are reserved by system, these include: *0x0000, 0x0015, 0x0016,0x0018,0x001e,0x3f00,0x3f01,0x3f02,0x3f03,0x3f04*

Requirement:
Hardware Version: v2.xx, v5.xx

S4Execute

Local: Execute VM EXF specified under current directory.

```
DWORD WINAPI S4Execute (
      SENSE4_CONTEXT *pS4Ctx,
      LPSCTR lpszFileID,
      LPVOID lpInBuffer,
      DWORD dwInBufferSize,
      LPVOID lpOutBuffer,
      DWORD dwOutBufferSize,
      DWORD *dwBytesReturned);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
lpszFileID	[in] ID of the EXF to be executed.
lpInBuffer	[in] Input buffer, storing the data to be transmitted in, namely the content of communication buffer when input
dwInBufferSize	[in] Length of the inputted data
lpOutBuffer	[out] Output buffer, storing the data to be transmitted out, namely the content of communication buffer when output
dwOutBufferSize	[in] Size of *lpOutBuffer*, when it is less than the length of the data returned from inside Elite EL, this function will return error indicating insufficient output buffer space.
dwBytesReturned	[out] Length of the data actually returned from Elite EL, namely the length of the data in communication buffer.

Return values:
S4_SUCCESS or error code

Remarks:
You must verify User PIN before invoking this API.

For local operations, the input buffer is max 250 bytes; for network operations, input buffer is max 237 bytes.

Requirement:
Hardware Version: v2.xx, v5.xx

Sample:
Please refer to the Section “User level access”.

S4ExecuteEx

Local: Execute VM EXF specified under current directory.

```
DWORD WINAPI S4ExecuteEx (
      SENSE4_CONTEXT *pS4Ctx,
      LPSCTR lpszFileID,
      DWORD dwFlag,
      LPVOID lpInBuffer,
      DWORD dwInBufferSize,
      LPVOID lpOutBuffer,
      DWORD dwOutBufferSize,
      DWORD *dwBytesReturned);
```

Parameters:

pS4Ctx	[in] Pointer to SENSE4_CONTEXT struct, pointing to an opened EL device.
lpszFileID	[in] ID of the EXF to be executed.
dwFlag	[in] Type of the executable: S4_VM_EXE VM executable file
lpInBuffer	[in] Input buffer, storing the data to be transmitted in, namely the content of communication buffer when input
dwInBufferSize	[in] Length of the inputted data
lpOutBuffer	[out] Output buffer, storing the data to be transmitted out, namely the content of communication buffer when output
dwOutBufferSize	[in] Size of *lpOutBuffer*, when it is less than the length of the data returned from inside Elite EL, this function will return error indicating insufficient output buffer space.
dwBytesReturned	[out] Length of the data returned from EL, namely the length of the data in communication buffer.

Return values:
S4_SUCCESS or error code

Remarks:
You must verify User PIN before invoking this API.

This API is almost identical with *S4Execute()*.

While executing VM EXF file, input buffer is max 250 bytes.

Requirement:
Hardware Version: v2.xx, v5.xx

S4Startup

S4Cleanup

These two functions has already been reserved, please do not use.

Error Code List of API

Error Code	Value	Definition
S4_SUCCESS	0x00000000	Operation succeeded.
S4_UNPOWERED	0x00000001	The device has been powered off
S4_INVALID_PARAMETER	0x00000002	Invalid parameter.
S4_COMM_ERROR	0x00000003	Communication error, i.e. data transmission timeout.
S4_PROTOCOL_ERROR	0x00000004	Wrong communication protocol.
S4_DEVICE_BUSY	0x00000005	Device busy.
S4_KEY_REMOVED	0x00000006	Device removed or not connected.
S4_INSUFFICIENT_BUFFER	0x00000011	Insufficient buffer.
S4_NO_LIST	0x00000012	No device is found.
S4_GENERAL_ERROR	0x00000013	Commonly indicates not enough memory
S4_UNSUPPORTED	0x00000014	Function not supported or file system has not been created.
S4_DEVICE_TYPE_MISMATCH	0x00000020	Device type mismatch.
S4_FILE_SIZE_CROSS_7FFF	0x00000021	(Applied only to version 2.2 or earlier).
S4_CURRENT_DF_ISNOT_MF	0x00000201	The net module to be created isn't child directory of the root directory, available for network version only.
S4_INVAILABLE_MODULE_DF	0x00000202	The current directory is not a module, available only for network version.
S4_FILE_SIZE_TOO_LARGE	0x00000203	The file size is larger than *0x7FFF*.
S4_DF_SIZE	0x00000204	The size of the specified directory is not enough.
S4_DIRECTORY_EXIST	0x00006901	Directory already exists; max level of directory is exceeded; the directory is built completely; Repetitive creating root directory; unset root directory
S4_FILE_TYPE_MISMATCH	0x00006981	The device type does not match
S4_INSUFFICIENT_SECU_STATE	0x00006982	Security status not satisfied, corresponding PIN must be verified.
S4_PIN_BLOCK	0x00006983	Device locked.
S4_APPLICATION_TEMP_BLOCK	0x00006985	Application temporarily locked.
S4_FILE_EXIST	0x00006a80	File already exists.
S4_DEVICE_UNSUPPORTED	0x00006a81	Unsupported by the device.
S4_FILE_NOT_FOUND	0x00006a82	File not found.
S4_INSUFFICIENT_SPACE	0x00006a84	Insufficient file space.
S4_OFFSET_BEYOND	0x00006B00	File offset exceeds the boundary.
S4_CRYPTO_KEY_NOT_FOUND	0x00009403	Cryptographic key not found.
S4_APPLICATION_PERM_BLOCK	0x00009303	The directory has been locked.
S4_DATA_BUFFER_LENGTH_ERROR	0x00006700	Invalid data length
S4_CODE_RANGE	0x00010000	Out of code range, generally result from stack overflow.
S4_CODE_RESERVED_INST	0x00020000	Illegal instruction.
S4_CODE_RAM_RANGE	0x00040000	Illegal pointer to internal RAM.
S4_CODE_BIT_RANGE	0x00080000	Illegal bit variable.
S4_CODE_SFR_RANGE	0x00100000	Illegal SFR.

S4_CODE_XRAM_RANGE	0x00200000	Illegal pointer to external RAM.
S4WF_INVALID_S4CONTEXT	0x00000101	Invalid S4Context parameter (PS4WriteFile).
S4WF_INVALID_FILE_ID	0x00000102	Invalid File ID (PS4WriteFile).
S4WF_INVALID_PC_FILE	0x00000103	Invalid PC files (PS4WriteFile).
S4WF_INVALID_FLAGS	0x00000104	Invalid dwFlags parameter (PS4WriteFile).
S4WF_INVALID_FILE_SIZE	0x00000105	Invalid dwFileSize parameter (PS4WriteFile).
S4WF_INVALID_FILE_TYPE	0x00000106	Invalid dwFileType parameter (PS4WriteFile).
S4_MODULE_NOT_FOUND	0x00000301	Failed to find assigned module
S4_LICENSE_EXIST	0x00000302	License granted already
S4_USER_NOT_FOUND	0x00000303	No users found in network version
S4_LICENSE_INVALID	0x00000304	Non-effective license
S4_TIMEOUT	0x00000305	Overdue operating on network version
S4_NETWORK_ERROR	0x00000306	Network error
S4_LICENSE_NOT_FOUND	0x00000307	No available license
S4_EXECUTE_ERROR	0x00000308	Execution error
S4_TOTALLICENSE_BEYOND	0x00000309	Exceeding total licenses
S4_MODULELICENSE_BEYOND	0x00000310	Exceeding total licenses of module
S4_DEVICE_INVALID	0x00000311	Device is not available
S4_USERPIN_ERROR	0x00000312	Verification error on User PIN
S4_MODULE_ZERO	0x00000313	Module number is 0
S4_DEVICETYPE_ERROR	0x00000314	Device type error
S4_DEVICE_START_FAILED	0x00000315	Device initiating error
S4_DEVICE_STOP_FAILED	0x00000316	Failed to stop device
S4_NET_TIMEOUT	0x00000324	Overdue communication on network version
S4_ERROR_UNKNOWN	0xffffffff	Unknown error.

Appendix A: Cryptographic API 9

EL hardware provides DES/TDES, RSA, SHA1 algorithms. In actual applications, the calculation procedure for some cryptographic algorithms must be done on computer end. For instance, you would like to encrypt part of a file to ciphertext before releasing software, and decrypt it using EL when running the software. In this case, you can use software algorithms to complete the encryption process.

Algorithms in EL hardware are all of standard ones, so you can easily get various implementations of them. That's one of the key features a secure algorithm must possess: it must not depend on the algorithm itself, instead, it must depend on the confidentiality of the secret key. That's also what distinguishes EL from other ordinary dongles which often provide so-called "black box algorithm" and it security is completely questionable.

SDK provides the software implementations of all the algorithms mentioned above to make it convenient for corresponding development. You can use these algorithms' interface directly or use other implementations you obtained. They will do the same job.

Besides, to simplify the generation, conversion and other operations of secret keys, SDK also provides some key-operating function interfaces.

Corresponding libraries and header files are placed under the directory (*SDK**support**crypto api*). Only C-version API is offered currently.

Key Operating Functions

X_GenerateRsaKeys

Generate a RSA key pair on computer and convert it to the Public/Private Key format supported by EL hardware

```
int WINAPI X_GenerateRsaKeys(
      COS_RSA_PUBLIC_KEY *pCosPubKey,
      COS_RSA_PRIVATE_KEY *pCosPriKey);
```

Parameters:

pCosPubKey	[out] Pointer to COS_RSA_PUBLIC_KEY struct which defines the format of RSA Public Key inside EL.
pCosPriKey	[out] Pointer to COS_RSA_PRIVATE_KEY struct which defines the format of RSA Private Key inside EL.

Return values:
RE_SUCCESS or error code

Remarks:
If you want to perform RSA calculation internally in EL hardware, a RSA key pair in corresponding format must be provided. You can use API *S4_WriteFile()* to generate a key pair inside the hardware or use this API(*X_GenerateRsaKeys()*) to generate a key pair and then import them using binary mode into Public/Private Key files inside the hardware. Please note that the RSA Private Key generated inside the hardware can NEVER be exported, and thus cannot backup either.

Another usage of the key pair generated by this API is: you can save them to two files in hard disk and when your want to debug your code, you can import these files to virtual device.

Requirement:
Hardware Version: v2.xx, v5.xx

R_GenerateRsaKeys

Generate a RSA key pair of PKCS#1 format on PC

```
int WINAPI R_GenerateRsaKeys(
      R_RSA_PUBLIC_KEY *pPubKey,
      R_RSA_PRIVATE_KEY *pPriKey);
```

Parameters:

pPubKey	[out] Pointer to R_RSA_PUBLIC_KEY struct, used to store the Public Key.
pPriKey	[out] Pointer to R_RSA_PRIVATE_KEY struct, used to store the Private Key.

Return values:
RE_SUCCESS or error code

Remarks:
RSA key pair generated by this function can be used in the software version cryptographic algorithms. It can also be of the format defined in API header file (*sense4.h*) after some simple processing.

R_RSA_PUBLIC_KEY struct is defined as follows:

```
typedef struct {
unsigned int bits;
unsigned char modulus[MAX_RSA_MODULUS_LEN];
unsigned char exponent[MAX_RSA_MODULUS_LEN];
} R_RSA_PUBLIC_KEY;
```

R_RSA_PRIVATE_KEY struct is defined as follows:

```
typedef struct {
unsigned int bits;
unsigned char modulus[MAX_RSA_MODULUS_LEN];
unsigned char publicExponent[MAX_RSA_MODULUS_LEN];
unsigned char exponent[MAX_RSA_MODULUS_LEN];
unsigned char prime[2][MAX_RSA_PRIME_LEN];
unsigned char primeExponent[2][MAX_RSA_PRIME_LEN];
unsigned char coefficient[MAX_RSA_PRIME_LEN];
} R_RSA_PRIVATE_KEY;
```

Compared the two above structs with the structs of RSA key pair defined in header file *sense4.h* and you will find that the only difference between them lies on one struct member variable: unsigned int bits.

Requirement:
Hardware Version: hardware-independent

X_Pub2Cos

Convert a RSA Public Key of PKCS#1 format to the format supported in EL hardware

```
int WINAPI X_Pub2Cos(
      COS_RSA_PUBLIC_KEY *pCosPubKey,
      R_RSA_PUBLIC_KEY *pPubKey);
```

Parameters:

pCosPubKey	[out] Pointer to struct COS_RSA_PUBLIC_KEY, storing the Public Key of EL format.
pPubKey	[in] Pointer to struct R_RSA_PUBLIC_KEY, storing the Public Key of PKCS#1 format.

Return values:
RE_SUCCESS or error code

Remarks:
None

Requirement:
Hardware Version: v2.xx, v5.xx

X_Pri2Cos

Convert a RSA Private Key of PKCS#1 format to the format supported in EL hardware

```
int WINAPI X_Pri2Cos(
      COS_RSA_PRIVATE_KEY *pCosPriKey,
      R_RSA_PRIVATE_KEY *pPriKey);
```

Parameters:

pCosPriKey	[out] Pointer to struct COS_RSA_PRIVATE_KEY, storing the Private Key of EL format.
pPriKey	[in] Pointer to struct R_RSA_PRIVATE_KEY, storing the Private Key of PKCS#1 format.

Return values:
RE_SUCCESS or error code

Remarks:
None

Requirement:
Hardware Version: v2.xx, v5.xx

X_Cos2Pub

Convert a RSA Public Key of the format supported in EL hardware to PKCS#1 format

```
int WINAPI X_Cos2Pub(
      R_RSA_PUBLIC_KEY *pPubKey,
      COS_RSA_PUBLIC_KEY *pCosPubKey);
```

Parameters:

pPubKey	[out] Pointer to struct R_RSA_PUBLIC_KEY, storing the Public Key of PKCS#1 format.
pCosPubKey	[in] Pointer to struct COS_RSA_PUBLIC_KEY, storing the Public Key of EL format.

Return values:
RE_SUCCESS or error code

Remarks:
None

Requirement:
Hardware Version: v2.xx, v5.xx

X_Cos2Pri

Convert a RSA Private Key of the format supported in EL hardware to PKCS#1 format

```
int WINAPI X_Cos2Pri(
      R_RSA_PRIVATE_KEY *pPriKey,
      COS_RSA_PRIVATE_KEY *pCosPriKey);
```

Parameters:

pPriKey	[out] Pointer to struct R_RSA_PRIVATE_KEY, storing the Private Key of PKCS#1 format.
pCosPriKey	[in] Pointer to struct COS_RSA_PRIVATE_KEY, storing the Private Key of EL format.

Return values:
RE_SUCCESS or error code

Remarks:
None

Requirement:
Hardware Version: v2.xx, v5.xx

Cryptographic Algorithm Functions

RSAPublicEncrypt

RSA Public Key encryption using PKCS#1 mode

```
int WINAPI RSAPublicEncrypt(
      unsigned char *output,
      unsigned int *outputLen,
      unsigned char *input,
      unsigned int inputLen,
      R_RSA_PUBLIC_KEY *publicKey);
```

Parameters:

output	[out] Output cipher.
outputLen	[out] Length of the output. It will be 128 if the RSA key is of 1024 bits.
input	[in] Input plaintext
inputLen	[in] Length of input. It cannot exceed 117 if the RSA key is of 1024 bits.
publicKey	[in] Pointer to struct R_RSA_PUBLIC_KEY, the Public Key used for encryption.

Return values:
RE_SUCCESS or error code

Remarks:
This function has completely the same PKCS encryption mode as SES *_rsa_enc()*.

Requirement:
Hardware Version: hardware-independent

RSAPrivateDecrypt

RSA Private Key decryption using PKCS#1 mode

```
int WINAPI RSAPrivateDecrypt(
      unsigned char *output,
      unsigned int *outputLen,
      unsigned char *input,
      unsigned int inputLen,
      R_RSA_PRIVATE_KEY *privateKey);
```

Parameters:

output	[out] Output plaintext.
outputLen	[out] Length of the output. It cannot exceed 117 if the RSA key is of 1024 bits.
input	[in] Input cipher.
inputLen	[in] Length of input. It will be 128 if the RSA key is of 1024 bits.
privateKey	[in] Pointer to struct R_RSA_PRIVATE_KEY, the Private Key used for decryption.

Return values:
RE_SUCCESS or error code

Remarks:
This function has completely the same PKCS encryption mode as SES *_rsa_dec()*.

Requirement:
Hardware Version: hardware-independent

Sign

RSA signing using PKCS#1 mode

```
int WINAPI Sign(
      int digestAlgorithm,
      unsigned char *plain,
      unsigned int plainLen,
      unsigned char *signature,
      unsigned int *signatureLen,
      R_RSA_PRIVATE_KEY *privateKey);
```

Parameters:

digestAlgorithm	[in] Hash algorithm used in signing.
plain	[in] Data to be signed
plainLen	[in] Length of the data to be signed.
signature	[out] Signature.
signatureLen	[in] Length of the signature. It will be 128 if the RSA key is of 1024 bits.
privateKey	[in] Pointer to struct R_RSA_PRIVATE_KEY the Private Key used for signing.

Return values:
RE_SUCCESS or error code

Remarks:
This function has completely the same PKCS signing mode as SES *_rsa_sign()*.

Requirement:
Hardware Version: hardware-independent

Verify

RSA signature verification using PKCS#1 mode

```
int WINAPI Verify(
      int digestAlgorithm,
      unsigned char *plain,
      unsigned int plainLen,
      unsigned char *signature,
      unsigned int signatureLen,
      R_RSA_PUBLIC_KEY *publicKey);
```

Parameters:

digestAlgorithm	[in] Hash algorithm used in signing.
plain	[in] Original data
plainLen	[in] Length of the original data.
signature	[in] Signature.
signatureLen	[in] Length of the signature. It will be 128 if the RSA key is of 1024 bits.
publicKey	[in] Pointer to struct R_RSA_PUIBLIC_KEY, the Public Key used for signature verification.

Return values:
RE_SUCCESS or error code

Remarks:
This API does not have a corresponding SES. The SES *_rsa_veri()* cannot handle the plaintext directly and thus has minor difference with this API.

If you intend to use SES, you have to call *_sha1_xxx()* series functions to complete hash calculation first and then call *_rsa_veri* (mode two) to verify the signature.

Requirement:
Hardware Version: hardware-independent

Digest

Hash algorithms, including SHA1, MD5, MD2

```
int WINAPI Digest(
      int digestAlgorithm,
      unsigned char *plain,
      unsigned int plainLen,
      unsigned char *digest,
      unsigned int *digestLen);
```

Parameters:

digestAlgorithm	[in] Hash algorithm used: DA_MD2 (MD2) DA_MD5 (MD5) DA_SSenselock (SHA1)
Plain	[in] Message to be hashed.
plainLen	[in] Length of the message.
Digest	[out] Digest.
digestLen	[out] Length of the digest.

Return values:
RE_SUCCESS or error code

Remarks:
None

Requirement:
Hardware Version: hardware-independent

DES

DES algorithm–ECB mode

```
int WINAPI DES(
      unsigned char *key,
      int encrypt,
      unsigned char *output,
      unsigned char *input,
      unsigned int inputLen);
```

Parameters:

key	[in] 8 octets secret key.
encrypt	[in] flag, 1 means encryption and 0 means decryption.
output	[out] output of encryption/decryption.
input	[in] input data.
inputLen	[in] Length of the input data, must be a multiple of 8.

Return values:
RE_SUCCESS or error code

Remarks:
When being used for encryption, this API is equal to SES *_des_enc()*and for decryption, it is equal to SES *_des_dec()*.

Requirement:
Hardware Version: hardware-independent

TDES

TDES algorithm in ECB mode

```
int WINAPI TDES(
      unsigned char *key,
      int encrypt,
      unsigned char *output,
      unsigned char *input,
      unsigned int inputLen);
```

Parameters:

key	[in] 16 octets secret key.
encrypt	[in] flag, 1 means encryption and 0 means decryption.
output	[out] output of encryption/decryption.
input	[in] input data.
inputLen	[in] Length of the input data, must be a multiple of 8.

Return values:
RE_SUCCESS or error code

Remarks:
When used for encryption, this API is equal to SES *_tdes_enc()*and for decryption, it is equal to SES *_tdes_dec()*.

Requirement:
Hardware Version: hardware-independent

Error Code List of Cryptographic API

Error Code	Value	Definition
RE_SUCCESS	0x0000	Operation succeeded.
RE_CONTENT_ENCODING	0x0400	Incorrect content encoding.
RE_DATA	0x0401	Invalid data.
RE_DIGEST_ALGORITHM	0x0402	Invalid digest algorithm.
RE_ENCODING	0x0403	Incorrect encoding.
RE_KEY	0x0404	Invalid key.
RE_KEY_ENCODING	0x0405	Invalid key encoding.
RE_LEN	0x0406	Incorrect length.
RE_MODULUS_LEN	0x0407	Incorrect Modulus length.
RE_NEED_RANDOM	0x0408	Random number needed.
RE_PRIVATE_KEY	0x0409	Error in Private Key encryption.
RE_PUBLIC_KEY	0x040a	Error in Public Key decryption.
RE_SIGNATURE	0x040b	Error in signing.
RE_SIGNATURE_ENCODING	0x040c	Incorrect signature encoding.
RE_ENCRYPTION_ALGORITHM	0x040d	Incorrect encryption algorithm.
RE_PARAMETER	0x040e	error parameter
RE_MEMORY	0x040f	alloc memory failed