2014/12

EL ドングルマニュアル
プログラム開発編

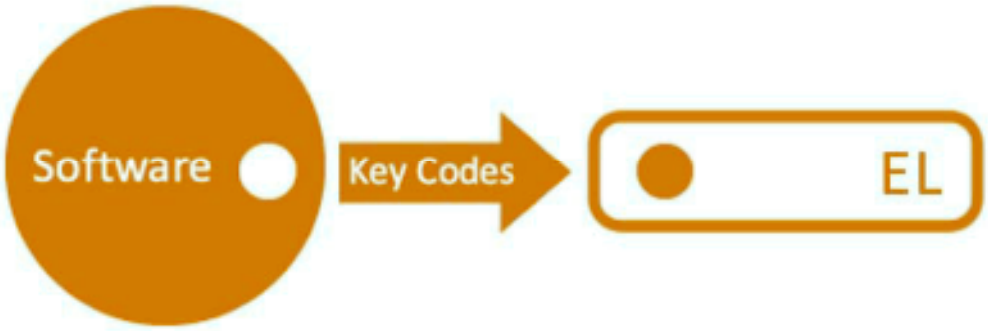
有限会社リビッグ
横浜市港南区上大岡西 1-12-2-801
Tel: 045-843-7122　Fax: 045-843-7142
http://www.ribig.co.jp/el/

目次

EL について

Elite EL シリーズは強力で効果的なプロテクションを実現するドングルです。コードポートとスマートカードの組み合わせによってハイエンド市場向けのプロテクトドングルに必要な機能を提供します。

コードポート技術



コード(プログラム)を ELにポート(転送)すると、EL内では独立した実行ファイル (EXF) として保存されます。コンピュータ側プログラムは、引数を付けて EL 内の実行ファイル (EXF)を呼び出すことができます。呼び出された EXF は EL 内の OS によって実行され、処理終了後に呼び出し元に戻り値を返します。EL が接続されていなければ、コンピュータ側のプログラムは実行を続けるために必要なコードが実行されないため、正常に動作しません。

EL には複数プログラム（コード）をポートできます（EL 内には複数の EXF を保存できます)。 EXF は C で作成します。PC 用の標準関数を使った C プログラムは、わずかな変更で EXF 作成に再利用できます。

開発手順例

1. プロテクトを考えずに C/C++でコンピュータ側のアプリケーションプログラムを作成します。
2. 作成したコードの一部を EL で実行させることを考えます。EL で実行させるコード部分を取り出して、EL プログラムを作成します。最終的には EL の CPU 用の C クロスコンパイラでコンパイルして、EL にコードポートします。

３．　コンピュータ側のアプリケーションプログラムでは、EL に移した部分を EL 内プログラムの呼び出しに置き換えます。

スマートカード

NXP(Philips) の 16bit CPU,RAM,EEPROM, USB 通信モジュールをすべて搭載したスマートカードを心臓部に使っています。スマートカード（IC カード）のため、各種ハードウェア攻撃に対して有効的な耐性があります。利用しているスマートカードは EAL 5 以上の評価保障レベルに準拠しています。

自動セルフロッキング機構

スマートカードへのアクセスは PIN によって保護されています。一定回数以上の誤った PIN でアクセスを試みると、スマートカードは自動ロックして外部からアクセスできないようになります（　解除はできません　）

GUSN（グローバルー意シリアル番号）

スマートカードには変更不可、上書き不可の GUSN が設定されています。

暗号処理コプロセッサ

EL は RSA(1024bit)/TDES 暗号処理用のコプロセッサを搭載しているため、暗号処理は高速です。また、ハードウェアによる真の乱数を生成できます。

内臓タイマー

EL にはタイマーを内蔵した機種があります。

HID モード/専用ドライバモード

EL は HID モードかドライバモードで動作します。HID モードでは、EL は OS のドライバによって自動認識されます。ドライバモードの EL を利用するには専用ドライバをインストールする必要があります。ドライバモードでは、複数のプロセスが同時に EL を呼び出せますが、HID モードでは複数のプロセスが EL にアクセスすることはできません。

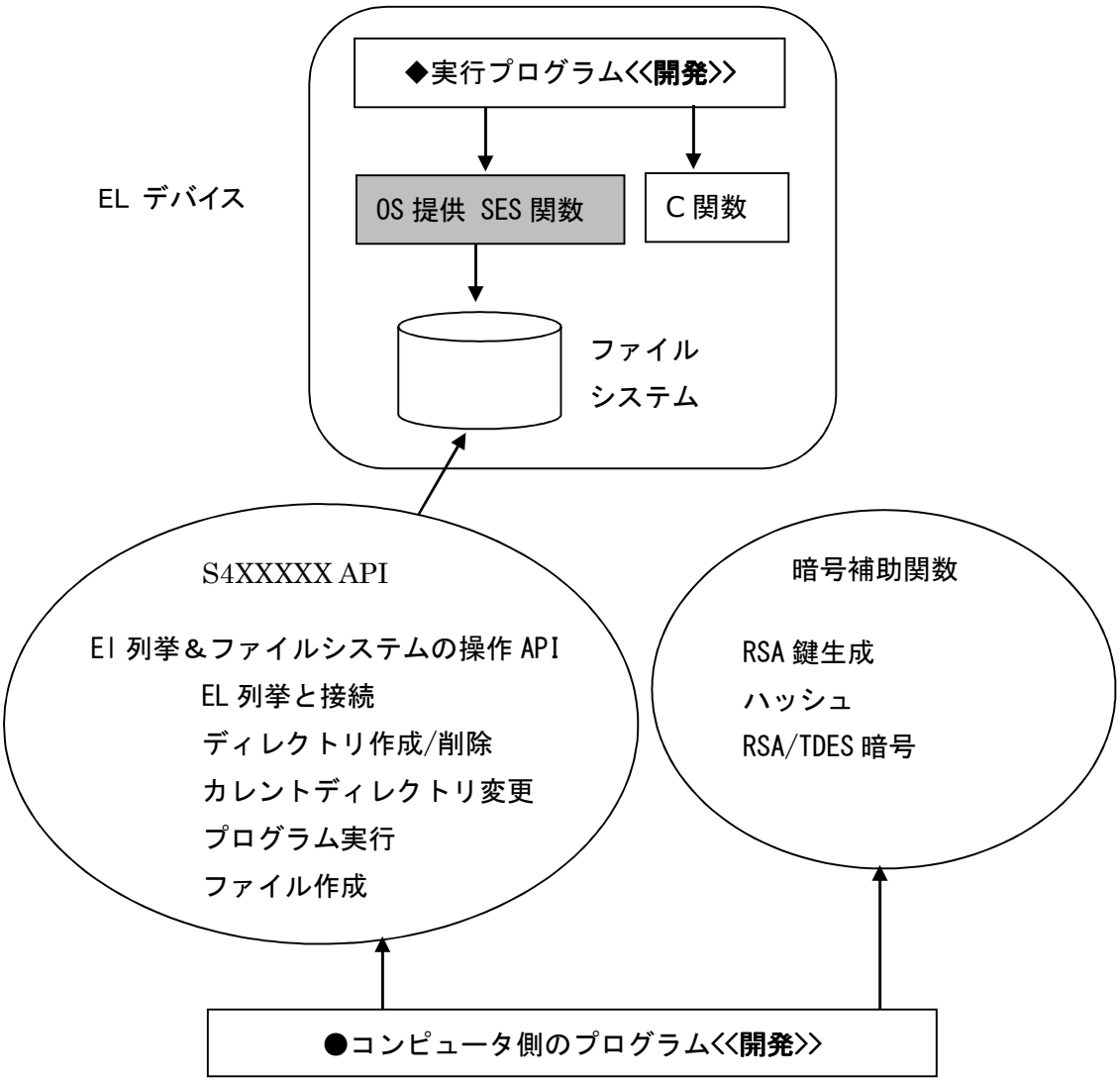
EL 向けプログラム開発

EL 内部で実行するプログラム

EL で実行するプログラムは Keil C51 を使って IDE(統合開発環境)環境で開発できます。Kei C51 に関する詳細は Keil Software の Web ページ http://www.keil.com をご覧ください。

コンピュータ側から EL デバイスを操作するプログラム

付属 C 用の API を使って各種言語で作成したプログラムから EL デバイスを操作します。



EL の機能を利用するには、EL 内で実行するプログラムを作成します。

- ◆ EL 内で実行するプログラムは、EL の OS 提供関数や C 標準関数を呼び出すことができます。
- ◆ 各種ファイルを作成できます。
- ◆ SDK に含まれる Keil C51 向けパッチをあてると IDE 環境で EL プログラムをデバッグできるようになります。

コンピュータ側のプログラムは、EL 内のプログラムを呼び出すようにします。

- ● コンピュータ側では EL を操作するための API や暗号補助関数を使って EL 機能を利用したプログラムを作成します。
- ● 付属の DevTool(DevTest.exe)プログラムをつかって、GUI で EL のファイルシステムに対する操作を行えます。

このマニュアルでは、EL 内部で実行するプログラムを作成するために必要な EL に関する知識、プログラム作成/デバッグ方法、EL デバイスへのプログラムのポート、そして、コンピュータ側から EL 内部プログラムを呼び出す方法について説明します。コンピュータ側からの EL 操作には DevTest.exe を使います。

このマニュアルとは別に、SDK には豊富な EL プログラムのサンプルがケーススタディという形式で用意されています。

コンピュータ側プログラムによる EL 操作は、主に EL 内プログラムの呼び出しになります。開発者側での EL 設定には、API による EL 操作が必要になるはずです。EL に対する操作 API は S4XXXX 関数のマニュアルをご参照ください。

開発に必要な EL に関する知識

ファイルシステム

EL はディレクトリ - ファイルの階層構造で管理します。各ディレクトリには ディレクトリ ID を割り当てます。ディレクトリに重複した ディレクトリ ID を割り当てることはできません。

１つのルートディレクトリと 3 階層までのディレクトリを作成できます。ファイルシステムがデイバスのすべてのストレージメモリを占有します。管理が複雑になるため、本当に必要なケースを除き、複数のディレクトリは作らずに単一ディレクトリで使うことをお勧めします。

ディレクトリ作成時にディレクトリが占めるメモリサイズを指定します。ディレクトリ作成後、ディレクトリサイズは変更できません。

ルートディレクトリを削除すると、ディレクトリ階層のファイルとディレクトリすべてが削除されます。

サブディレクトリは直接削除できません。内容がクリアされるだけです。ディレクトリがクリアされると、ディレクトリ内のファイルやディレクトリは削除されます。また指定ディレクトリの開発者 PIN とユーザ PIN はリセットされます。ディレクトリを削除して、占めている領域を解放するには、上のディレクトリをクリアします。

ファイルタイプ

EL には３つのファイルタイプがあります。

1．実行ファイル(EXF)
2．データファイル
3．鍵ファイル

すべてのファイルは外部から直接、読み出すことはできません。開発者 PIN を指定することで、ファイル作成 / 変更は直接外部から行えます。

１．実行ファイル

実行ファイルは内臓 OS によってロード、実行されます。同じディレクトリにある他のファイルに対して OS 提供のシステム関数を使って操作（作成、読み書き）できます。ただし、他の実行ファイルに対する操作は、その実行ファイルの読み書き属性に影響を受けます。

２．データファイル

データファイルはバイナリ形式のみサポートします。外部からデータファイルを直接読み込むことはできません。データを読み出すには、EL 内のプログラム経由で行います。

３．鍵ファイル

RSA の鍵ファイルです。詳細は SES RSA 関連関数をご参照ください。

権限と読み書き属性

ファイルタイプ		開発者 PIN	ユーザ PIN	EXF
実行ファイル	読み書き禁止	W	E	N
	読み書き許可	W	E	R/W
データファイル		W	N	R/W
鍵ファイル		W	N	W

R:　読み込み　　W:書込み　　E:実行　　N:不可

ファイル操作は、コンピュータ側からの操作なのか、EL 内部での操作なのか明確にするようにしてください。この区別を意識しないと混乱の元となります。

[コンピュータ側からの操作]開発者 PIN で EL にログインすると、すべてのファイルに対して書き込み操作が可能

[コンピュータ側からの操作]ユーザ PIN でログインすると、実行ファイルを実行できますが、データファイルと鍵ファイルに対する操作は行えません。

[EL 内部の操作]実行ファイルは、データファイルの読み書き、鍵ファイルの書込み、読み書きが許可された実行ファイルに対する読み書きの操作が行えます。

PIN　**認証**[コンピュータ側からの操作が対象となります]

2 種類の PIN があります。外部から EL を操作するには PIN でログインしなければなりません。ただし、コンピュータに接続している EL の検索、オープン、ディレクトリ変更、クローズ、コントールコード送信などには PIN ログインは不要です。

1. 開発者 PIN
2. ユーザ PIN

1. 開発者 PIN

開発者 PIN は 24 バイトです。既定値は "123456781234567812345678" です。開発者 PIN でログインすると、ファイルの読み出しはできませんが、EL に対するその他のすべての操作が行えます。このため開発者以外が開発者 PIN を入手できないようにしてください。

- 開発者 PIN は、開発側で EL を設定するときだけ必要です。絶対に開発者 PIN を配布プログラム内に記述するようなことはしないでください。
- EL のすべてのディレクトリの開発者 PIN は、ユーザに渡す前に必ず既定値から別の値に変更してください。

外部に開発者 PIN が知られてしまっても、簡単に EL 内のデータが読みだされることはありません。直接読み込むことは不可能です。EL 内のファイル名の一覧を得る手段は用意されていません。しかし、プログラムの書き換え、ディレクトリ削除などどのような事態が発生するか予測できません。開発者 PIN 管理には細心の注意を払ってください。

２．　ユーザ PIN

ユーザ PIN は 8 バイトです。既定値は "12345678" です。ユーザ PIN で行えるのは、EL 内の実行ファイル（EXF）を呼び出すことだけです。EL プログラム呼び出し前のユーザ確認をするための PIN で、絶対に外部に分かってしまってはいけないようなものではありません。

３．　PIN の設定先

PIN は各ディレクトリに独立して設定します。デバイス全体に対して設定されませんのでご注意ください。それぞれのディレクトリは開発者 PIN とユーザ PIN を持ちます。ルートディレクトリの 開発者 PIN を変更してもサブディレクトリの 開発者 PIN は変更されません。また、ルートディレクトリに開発者 PIN でログインしても、サブディレクトリに開発者 PIN でログインしたことにはなりません（EL を単一ディレクトリで利用するように推奨する理由は複雑な PIN 管理を避けるためです）。

複数のプログラムが１つの EL ドングルを利用するときに、それぞれのプログラムが異なるディレクトリを利用すると（異なる開発者 PIN/ユーザ PIN）、プログラム間の干渉を避けることができます。

４．　PIN ロック

15 回連続してルートディレクトリに誤った開発者 PIN でログインしようとすると、EL はロックします。ロックは解除できません。サブディレクトリに誤った開発者 PIN でログインするとディレクトリがロックしますが、ディレクトリは削除できます。ユーザ PIN はロック動作を引き起こしません。

５．PIN 初期化

ルートディレクトリを削除、再作成したり、サブディレクトリをクリアしたりすると、そのディレクトリの開発者/ユーザ PIN は規定値に戻ります。ファイルシステムに変更した場合、必ず PIN を既定値から変更することを忘れないでください。

EL プログラム作成の概要

１．通常の C プログラムを作成

```
#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]);
        }
}
```

２．EL にコードポートする部分を選択。

その部分を再利用して EL プログラムを作成。

ここでは　void bubble_sort(unsigned char *p, int len)　をEL側に移すことにしたとします

ELプログラム

```
#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();
}
```

典型的な EL プログラムでは、main() で pbInBuff, pInLen マクロで引数を受け取り、処理終了後, _set_response で結果（バッファとバッファ長）を返してプログラムを _exit() 終了します。

pbInBuf はコンピュータ側から送られたデータを保持するコミュニケーションバッファを指しています。pInLen はバッファの長さが入っています。

_set_response と _exit は内臓 OS 提供関数(SES 関数)です。SES 関数を利用するため

ヘッダファイル"ses_v3.h"をインクルードしています。その他の部分は通常の C プログラムと同じです。

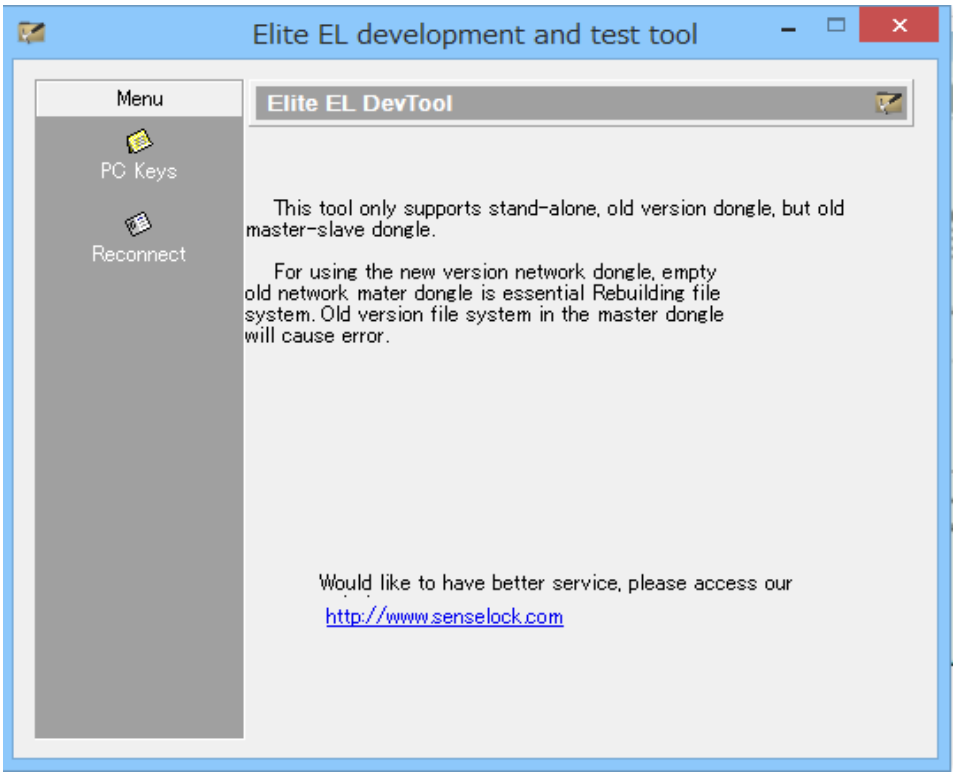
３．EL プログラムのビルド

uVision IDE でプロジェクトを作成してファイル demo1.c を追加してから、プログラムのソースを入力します。また、ライブラリファイル Ses51L.lib とヘッダファイル ses_v3.h をプロジェクトディレクトリにコピーしてから、Ses51L.lib をプロジェクトに追加してください。

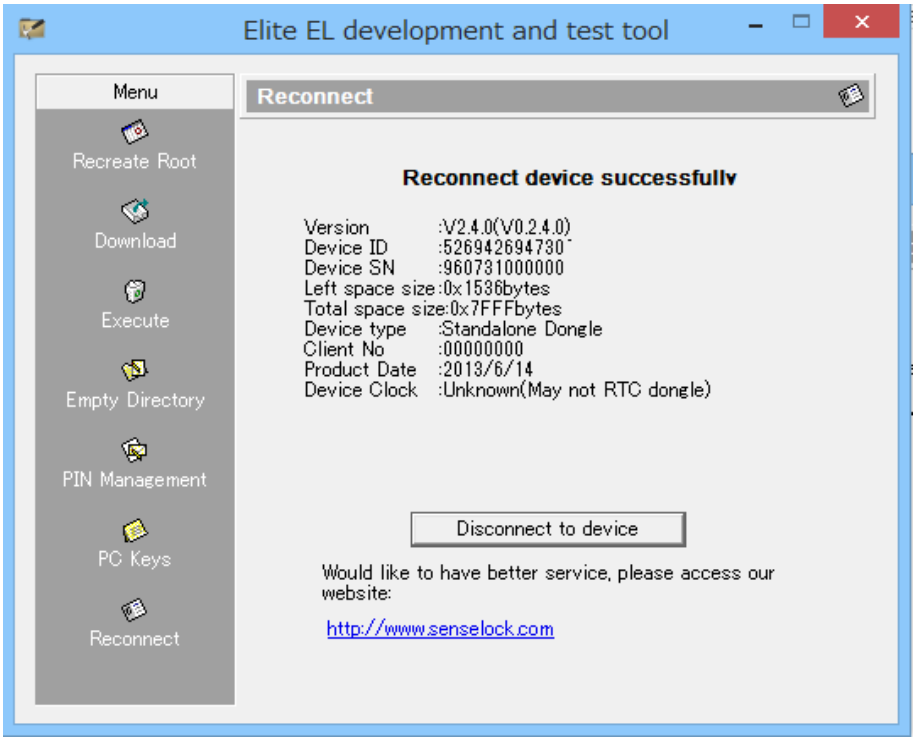
Project->Build target でコンパイルして成功すると demo1.hex ができあがります。詳細は "コード開発"の章をご参照ください。

４．EL へコードポート

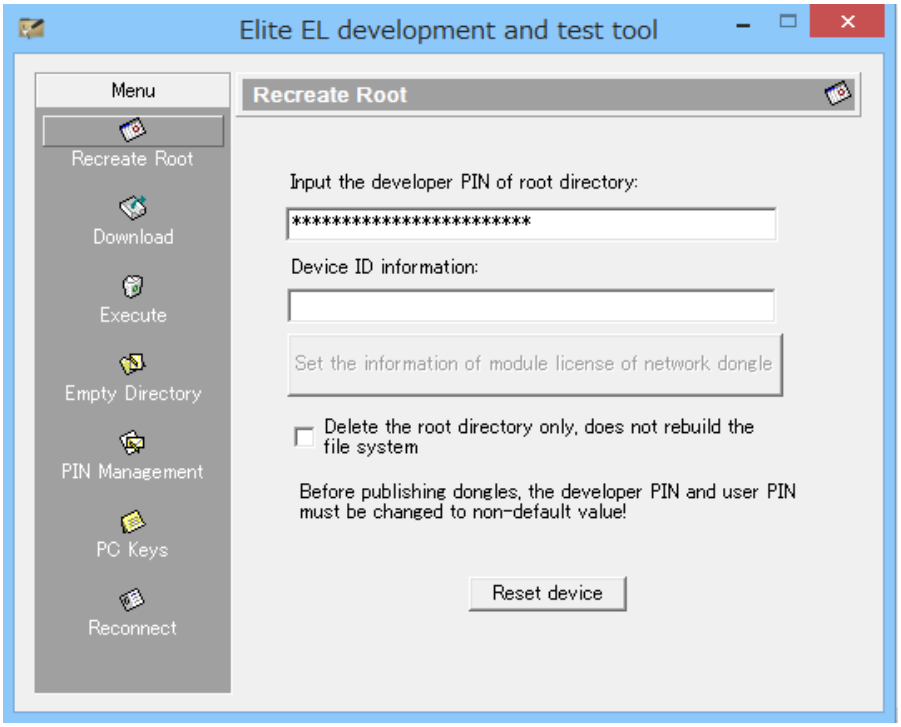
EL を接続して、EL 開発キットの /tools/devtest.exe を実行してください。



Menu の Reconnect 選択すると Menu で各種操作が行えるようになります。

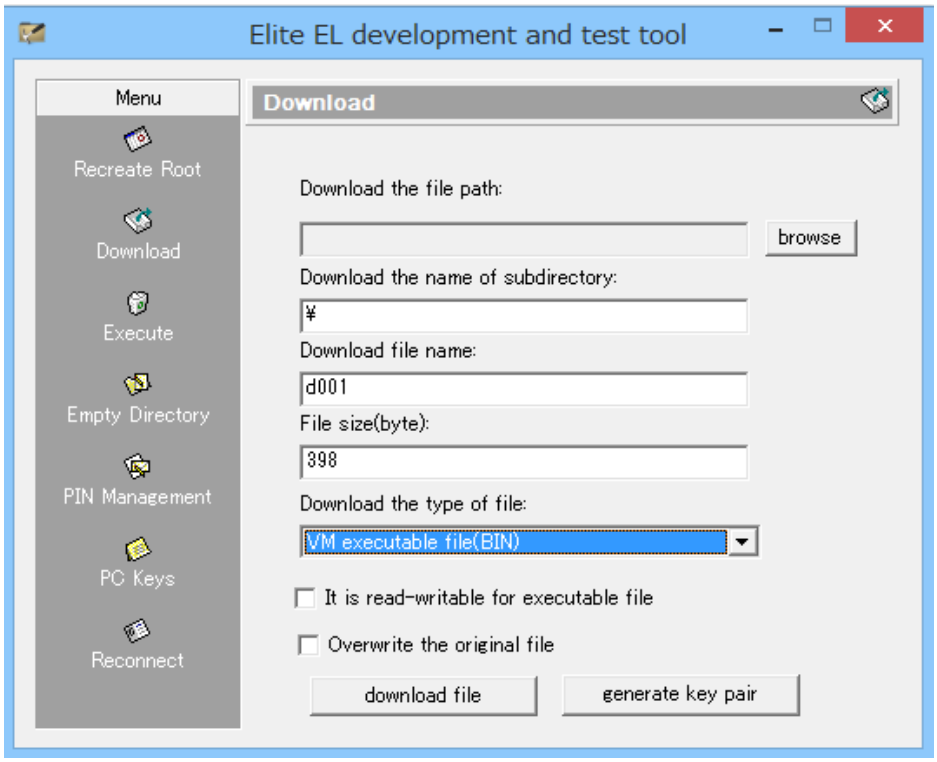


プログラムを転送する前に、ルートディレクトリを作成します。Menu の Recreate Root を選択します。



開発者 PIN の規定値が自動設定されますので、任意のデバイス ID(例えば ribig)だけを入力して "Reset Device"ボタンを押します。

ルートディレクトリ作成後、転送は Download メニュで行います。Demo1.hex をルートディレクトリに 0xd001 という名前で転送するとします。



Browse ボタンで demo1.hex を選択します。ダウンロード先は ¥ とします。ファイルサイズは自動計算されます。既に同じ名前で EL に転送されたプログラムがあれば、Overwrite the original file にチェックしてください。

５．コンピュータ側のプログラム変更

EL に移した部分を EL プログラム呼び出しに置き換えます。

```
#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)
{
//
}
```

bubble_sort(unsigned char *p, int len) 関数呼び出しの代わりに EL側の 実行ファイル0xd001 を呼び出すようにします。

```
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;
}
```

呼び出すコードをポートした EL をオープンして、対象ディレクトリに移動します。そして、ユーザPIN でログイン後、Execute でプログラム 0xd001 を呼び出します。

一般的なELプログラム呼び出しまでの手続
Enum->Open->ChangeDir->VerifyPin->Execute

この関数の引数　char *fid　でプログラムファイル名を変えて呼び出すことで、ルートディレクトリの別名のEXFを呼び出すことができます。

このコードをビルドするにはヘッダファイル sense4.h と EL API ライブラリファイル が必要です。

EL 用プログラム開発の詳細

C 言語で作成します。

プログラムの基本構造

```
#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();

}
```

pbInBuff, pInLen マクロでコンピュータからの引数を受け取り、処理後、結果を _set_response で返します。

データを受け渡すコミュニケーションバッファは 250 バイトを超えることはできません。250 バイト以上のデータを受け渡しするには、何回かに分けて呼び出さなければなりません。

実行モード

EL の実行ファイル(EXF)は　VM 実行ファイルと呼ばれるものです。コンパイル時にコードのモードを VM モード(Virtual Machine Mode)に指定します。

EL のメモリ構造

EL の RAM メモリは 2K バイトです。オーバフローしないように使わなければなりません。

メモリの割り当て

VM モードではメモリーには内部 RAM と外部 RAM の２種類あります。内部 RAM は 256 バイト、残りが外部 RAM になります。コード内で xdata, idata キーワードを使って、どちらのメモリを使うかを指定できます。内部 RAM はプログラム実行中にスタックで使われますので、通常はオーバフローを避けるために外部 RAM を使います。

```
xdata unsigned long x;   // 外部RAM
idata unsigned long i;   // 内部RAM
```

xdata, idata キーワードは必ず指定する必要はありません。コンパイル時に Memory Model に Large （EL 用プログラムは Large で作成するようにします）を設定しておくと、すべての変数は外部 RAM に配置されます。内部 RAM に配置する変数にだけ idata を指定します。

メモリアドレス

VM モードでのメモリアドレスは以下の通りです。

0 ……. 0xFF	0 …… 0x7FF	0x8000	0x8001…….. 0x80FA
内部RAM	外部RAM	データ長	コミュニケーション バッファ

メモリ共有

VM モードでは　EL をリセットしない限り、外部 RAM の内容はクリアされません（内部 RAM はクリアされます)。この特徴をつかって、複数のプログラムや同一プログラムの複数回の呼び出し（同一プログラムの複数インスタンス）でメモリを共有できます。

例えば、EL のプログラムが 250 バイト以上のデータを受け取る方法として、外部 RAM に 500 バイトのバッファを配置して、200 バイト、200 バイト,100 バイトの 3 回に分けて、この領域にデータを書き込むことができます。外部 RAM は呼び出し毎にクリアされないため、3 回分のデータを保持しつづけます。

```
#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();
}
```

このコードでは２つのマクロを使っています。

DEFINE_AT	指定 RAM の指定アドレスに変数を定義
LE16_TO_CC	リドルエンディアンから EL の数値フォーマットに変換

unsigned char の 512 バイト配列変数を外部 RAM に配置しています。呼び出し側が指定した配列内での書込み位置、長さ、データを使ってこの配列にデータをセットします。

コンピュータ側では Offset, Len, buff を適切にセットして、この EL プログラムを呼び出します。

1. Offset = 0, len = 200, buff[200]
2. Offset = 200, len = 200, buff[200]
3. Offset = 400, len = 100, buff[100]

３回目の呼び出し後、big_buff にはコンピュータ側から送られてきた 500 バイトのデータがセットされています。

コードメモリ

これまで説明したメモリとは別に、読み出し専用のコードメモリというものがあります。初

期化済の大きな変数を保持するために利用できます。他のメモリと同じように使うことができます。

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

内臓 OS 提供関数（SES 関数）でコードメモリに対して操作を行うものはありません。コードメモリから RAM メモリにコピーは標準の C 関数（ strcpy(), memcopy() ） をご利用ください。

アライメントの問題

コンピュータと EL 間で構造体を受け渡しするにはバイト配置（アライメント）に配慮してください。PC のコンパイラは効率よく処理するため構造体を機械語命令のサイズに合わせて配置します。

例えば、PC コンパイラで構造体

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

をコンパイルすると構造体のサイズは char が１バイト、long が４バイトだとして、合計 5 バイト、にはなるわけではありません。機械語命令サイズが４バイトだとすると、long は４バイト目に配置（アライン）され、結果４バイト＋４バイトで８バイト構造体になります。一方、EL で同じ構造体を定義するとサイズは５バイトになります。PC 側と EL 側では構造体のアライメントが異なるため、構造体をそのままの形で受け渡すことはできません。

PC のコンパイラでは構造体のアライメントをバイト単位になるように指定するオプションが備わっています。Visual Studio では、*#pragma pack (push, 1)* を指定できます。

```
#pragma pack(push,1)
struct{
        char c;
        long l;
} x;
#prama pack(pop)
```

このような形で構造体を宣言して EL 側と構造体のアライメントを一致させてデータの受け渡しができるようにしてください。

ビックエンディアン と リトルエンディアン

ビックエンディアンは、データを上位バイトから順にメモリに順に配置します。
0x12345678（１６進数）というデータは、メモリ上に 12 34 56 78 と配置されます。

リトルエンディアンはデータを下位バイトから順にメモリに順に配置します。
0x12345678（１６進数）というデータは、メモリ上に　78 56 34 12 と配置されます。

Intel CPU を使ったコンピュータはリトリエンディアン方式です。一方、Keil　C51 はビッグエンディアン方式を採用しています。PC 側の数値を EL プログラムにそのままの形で渡すと、受け取り側の EL プログラムは誤った値を受け取ってしまいます。例えば、PC(Intel CPU)の 0x12345678 はメモリ上では 78 56 34 12 となって EL プログラム に渡されます。ビッグエンディアンとして数値を解釈する EL プログラムは、0x78563412 としてデータを受け取ります。正しく複数バイトの数値(short, int, long,float 等)を受け渡しするには、どちらかで変換しなければなりません。

開発キットには変換マクロが複数用意されています。LE16_TO_CC　でリトリエンティアン 16 ビット数値を LE 側のフォーマットに変換できます。CC_TO_BE16 は EL 側フォーマットの 16 ビット数値をビッグエンディアンに変換します。 CC は EL で使っている C Computer、LE はリトルエンディアン、BE はビッグエンディアンを表します。

倍精度浮動小数点数はコンパイラがサポートする標準データタイプではありません。SDK では倍精度浮動小数点数を表す DOUBLE_T という構造体が用意されています。このデータ型はどんなコンパイラを用いた場合でもリトリエンディアン形式になります。

データ型

型	char	integer	Short	Long	Float	double
長さ	1	2	2	4	4	8

ライブラリファイル

プログラムのメモリモデルに応じて３つのライブラリが用意されています。

Large	ses51l.lib
Small	ses51s.lib
Compact	ses51c.lib

EL プログラムは Large Model で作成してください。ライブラリファイルは最初にプロジェクトフォルダにコピーして、それからプロジェクトに追加してください。

EL プログラム作成時の留意点

1. スタックオーバフローを避けるため、多くのローカル変数（特に配列と構造体）を使わない
2. 多くのメモリを使う配列と構造体の変数はできるだけ少なくする
3. 深い関数呼び出しのネスティングは避ける
4. 外部 RAM に変数を置く
5. 構造体のアライメントと CPU アーキテクチャ（ビッグエンディアンとリトルエンディアン）に注意する
6. コメントは英語で（マルチバイト言語をコンパイラはサポートしていない）
7. sprint, sscanf の使用は避ける

コンピュータ側からの EL へのアクセス

1. 一般アクセス（無認証アクセス）

一般アクセスとは認証無しで EL にアクセスすることを指します。一般アクセス権限で可能な操作は以下の通りです。

1. EL デバイスの検索 / 列挙、
2. デバイスへの接続(オープン)
3. ディレクトリ変更、
4. コントールコード送信、
5. デバイス切断（クローズ）

無認証アクセスで EL を操作するサンプルプログラム

```
#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;
}
```

検索/列挙	S4Enum	通常、2 度呼び出します。 1 度目の呼び出しでは、第 1 引数を

		NULL, 第 2 引数を 0 にします。接続しているすべての EL を列挙して、列挙に必要なデバイスコンテキストのサイズを返します。例えば 5 つの EL が接続されていたら、5 つ分のデバイスコンテキストのサイズが返されます。いくつの EL が接続されているかは、返されたデバイスコンテキストのサイズを、1 つのデバイスコンテキストサイズで除算します。 ctx_size/sizeof(SENSE4_CONTEXT) 2 度目の呼び出しては、接続しているすべての EL のデバイスコンテキストを保存するために必要分のメモリーを確保してから、第 1 引数に渡します。 デバイスコンテキストにはシリアル番号が含まれます。これを使ってデイバスを識別できます。 コンピュータ側から EL を操作する API では第 1 引数に対象 EL のデバイスコンテキストを指定します。
デバイスのオープン	S4Open	対象デバイスのデバイスコンテキストを渡します。
コントールコード送出	S4Control	対象デバイスにコントールコードを送出します。LED 点滅/消灯やハードウェアシリアル番号取得など各種操作を行えます。
ディレクトリ変更	S4ChangeDir	オープン直後、カレントディレクトリはルートディレクトリです。
デバイスのクローズ	S4Close	処理終了後にクローズします。

EL 操作の一般的な手順は、対象となる EL デバイスを列挙後、オープンします。対象ディ

レクトリに変更してから、ユーザ PIN でログインしてプログラムを実行します。最後にデバイスをクローズします。HID モードではプログラムがオープンしているデバイスを別のプログラムがオープンすることはできません。

対象 EL に接続（オープン）する処理は頻繁に必要になります。1 つの呼び出しで済むようにしておくと便利です。

```
unsigned long OpenDdongle(SENSE4_CONTEXT *ctx, int index )
{
	SENSE4_CONTEXT* ps4ctx = NULL;
	unsigned long ctx_size = 0; /* size of device context list */
	unsigned long ret = 0; /* return value */

	if( ctx == NUL )
		return S4_INVALID_PARAMETER;

	/*enumerate all the EL devices connected. */
	ret = S4Enum(NULL, &ctx_size);
	if (ret != S4_SUCCESS && ret != S4_INSUFICIENT_BUFFER)
	{
		return ret;
	}
	if (ctx_size == 0)
	{
		return S4_KEY_REMOVED;
	}

	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));

	/* 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);

	return ret;
}
```

2. 開発者レベルアクセス

開発者 PIN でログイン（認証）後、開発者レベル権限で EL を操作ができるようになります。

1. ディレクトリの作成 / クリア / 削除
2. ファイルの作成/変更/削除

などが開発者権限を必要とする操作です。ログインは S4VerifyPin に開発者 PIN を与えて呼び出します。成功すると開発者権限を取得します。

```
#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[] = "123456781234567812345678";
	unsigned char old_dev_pin[] = "123456781234567812345678";
	unsigned char new_dev_pin[] = "876543218765432187654321";
	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;
}
```

＊このコードは問題ないように見えますが、１つ大きな欠陥があります。S4VerifyPin で開発者権限を取得すると、デバイスをS4Closeで閉じても権限状態はそのままになります。S4Closeは認証状態をクリアしません。そのため、次にS4Openでデバイスをオープンすると開発者レベル権限のままなので、開発者権限操作が可能です。認証状態をクリアするにはS4Controlでコントロールコード S4_RESET_DEVICE を送出してください。

開発者 PIN は外部に絶対に漏れないようにします。開発者側が EL の設定を行うコードでは使っても、ユーザに渡すプログラム内で絶対に使わないようにしてください。

3. ユーザレベルアクセス

このアクセスレベルの目的は、EL 内部の実行ファイル(EXF)を呼び出すことです。

```
#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 a root dir. */
        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 Developer PIN to get full access 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);

	// Reset Device if necessary

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

	return 0;
}
```

Keil C51 の使い方

Keil C51 の特徴

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.

ビルドの結果は HEX ファイルになります。このままでは EL では実行できません。コードポートする前に、hexbin.exe で BIN (バイナリ)フォーマットに変換してください。付属の DevTest.exe は HEX ファイルが指定されると、BIN ファイルに自動変換します。

EL 開発用パッチ

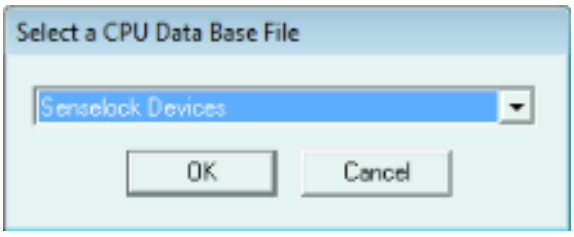
Keil C51 をインストールしたら、IDE¥keil ディレクトリにある wizard.exe を実行してください。Keil で EL プログラム開発に必要なセットアップを行います。

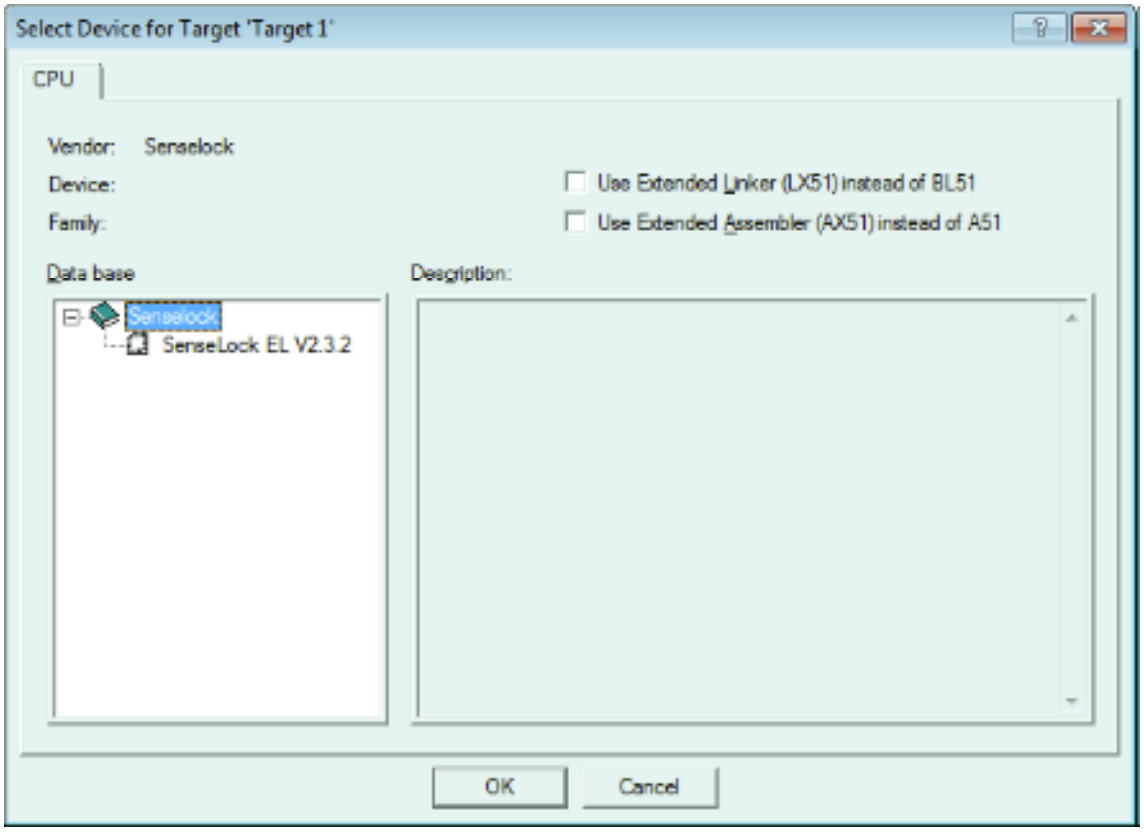
Keil C51 の IDE - uVision

プロジェクト作成

1. **Project->New Project** でプロジェクト名を設定して、保存します。

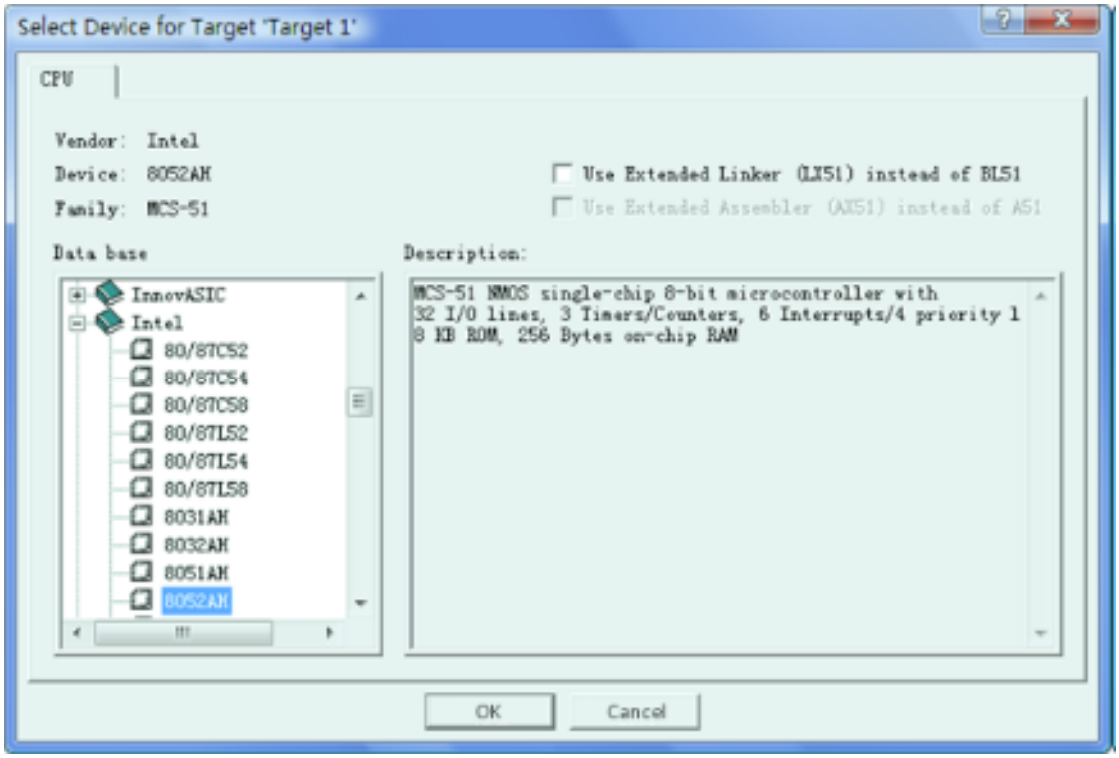
2. **Wizard.exe** でパッチを導入済みであれば、”Select a CPU Data Base File”ウィンドウが表示されます。”Senselock Device” を選択してください。





Data base リストで Senselock->SenseLock EL V2.3.2 を選択してください。

3. パッチを導入していなければ、Data　base リストボックスで CPU として **Intel->8052AH**　を選択してください。

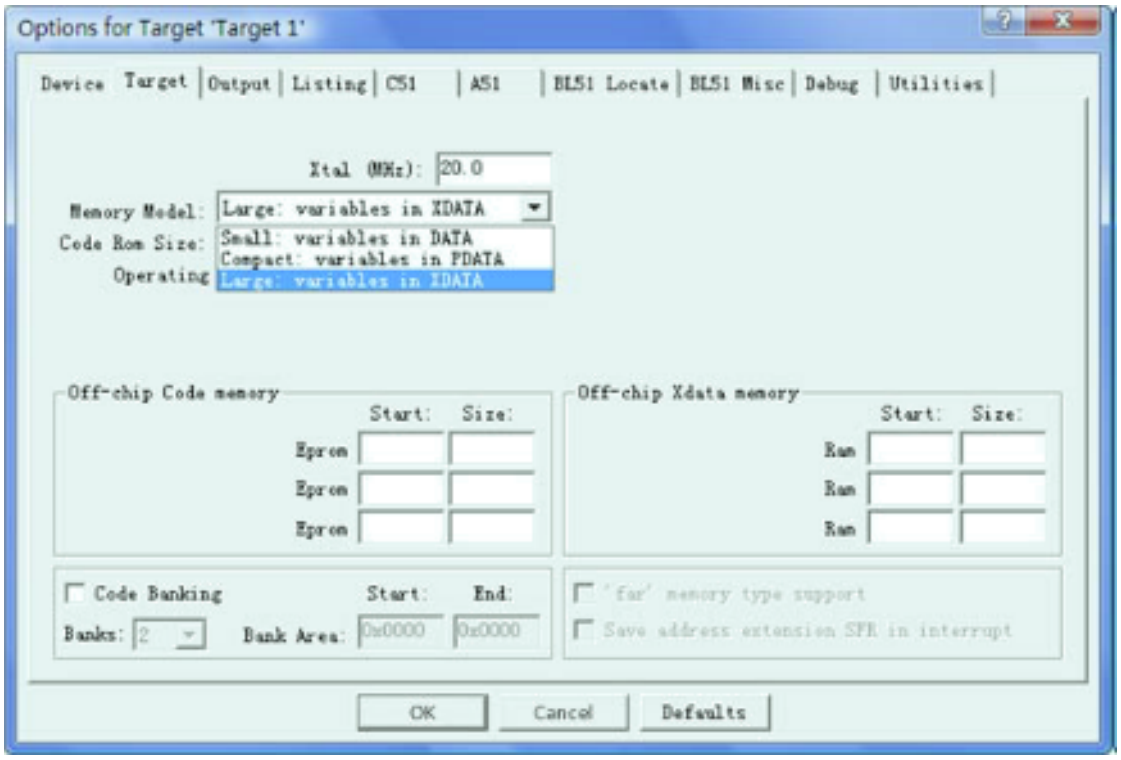


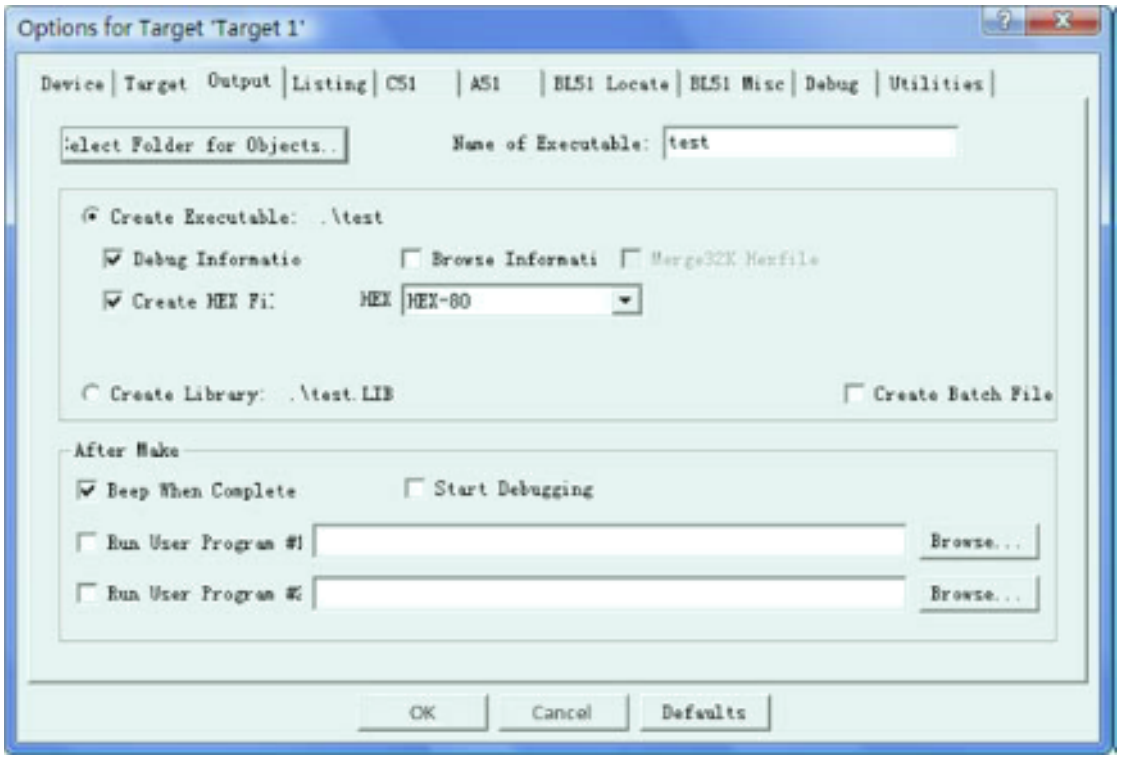
*****このダイアログは、**Project->Select Device for Target 'Target1'**で表示できます。

４．プロジェクトのオプション設定

IDE の左リストで Target を選択してから、P**roject->Options for Target 'Target 1'** でオプションダイアログが表示させます。

a. Target タブでMemory Model を Large に設定。
b. Output タブで Create HEX File をチェック





４．　ヘッダファイルとライブラリファイルの追加

Ses_v3.h をプロジェクトディレクトリか、C51の INC ディレクトリにコピーしてくだ

さい。また、**Project->Components, Environment, Books** でライブラリファイル*ses51L.lib*をプロジェクトに追加してください。ライブラリは開発キットの IDE/keil/lib 内にあります。

７．C のソースコードファイルを追加してください。

プログラムのビルトとデバッグ

Project->Build target でプログラムをビルドします。成功したらプロジェクトディレクトリに HEX ファイルが出来上がります。

デバッグするための設定

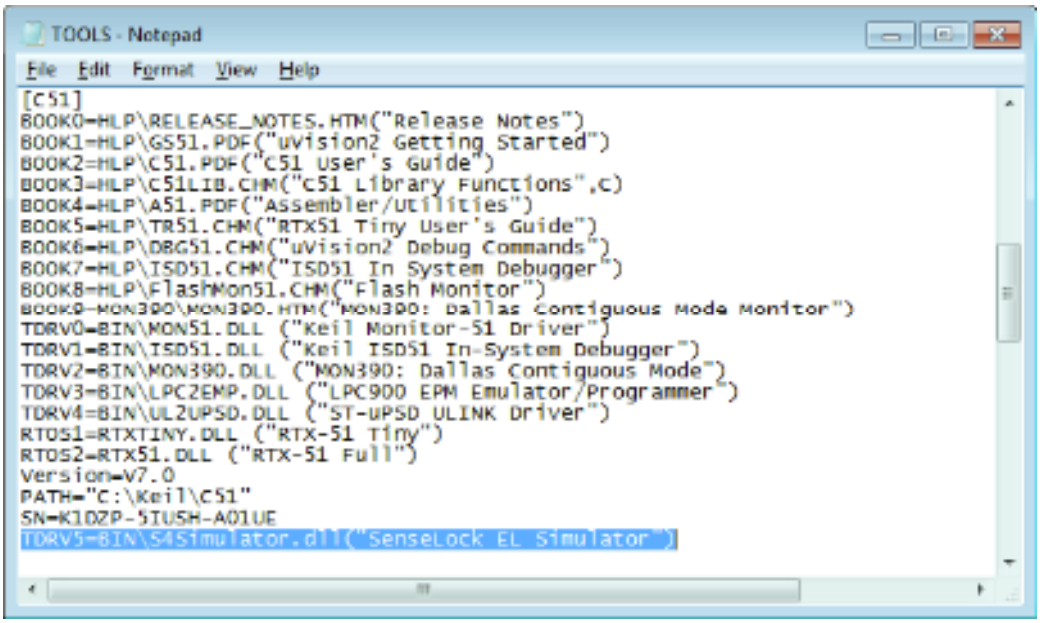
デバッグにはソフトウェアシミュレータを利用できます(uVision のバージョンによっては利用できない可能性があります)。パッチ導入時にシミュレータはセットアップされます。また、手動での設定手順は以下の通りです。

a. 開発キットの IDE/Keil/bin 内の S4Situmulator.dll と vfsView.exe を Keil の BIN ディレクトリ(…¥Keil¥C51¥BIN¥) にコピー

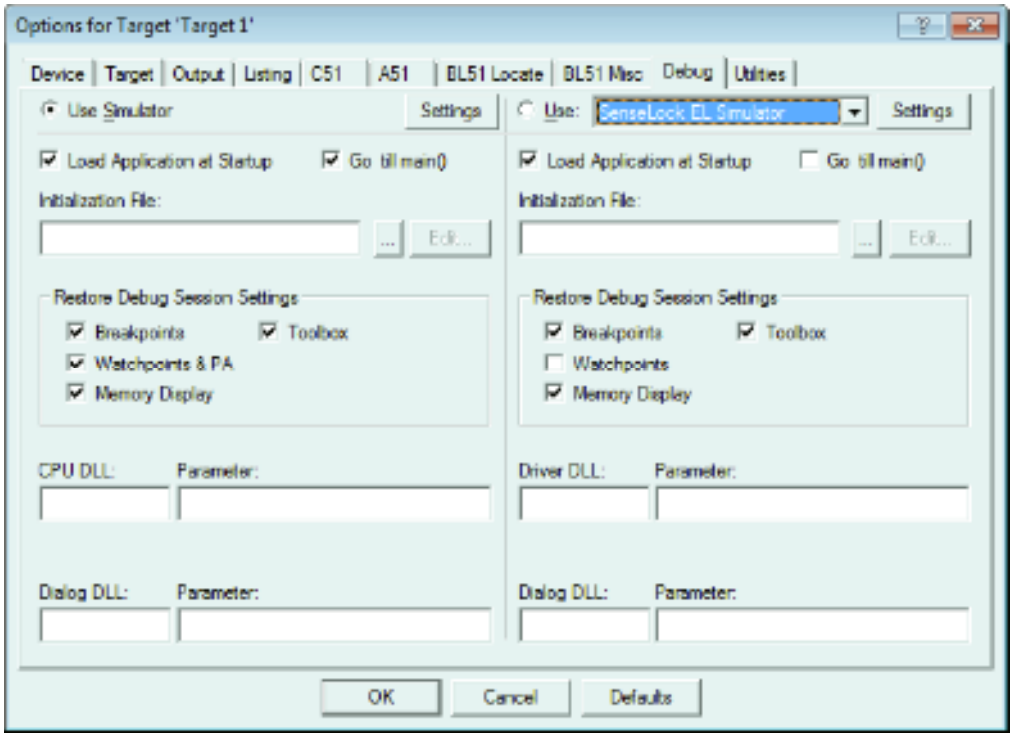
b. 設定ファイル(…¥Keil¥Tools.ini) の [C51] セクションに

TDRV0=BIN¥S4Simulator.dll("SenseLock EL Simulator")

を追加。既に TDRV0 という名前で DLLが登録されていたら、まだ使われていない番号に変更



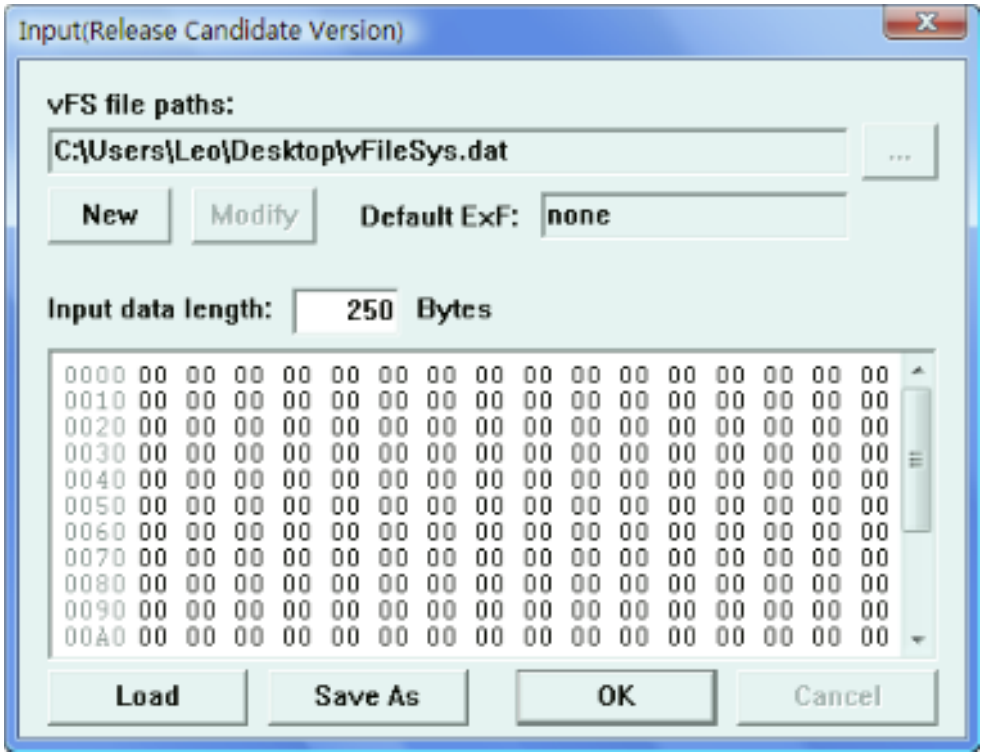
c. uVision を再起動後、**Project->Options for Target 'Target 1'** で表示されるダイアログで Debug タブを開き、SenseLock EL Simulator を選択。



d. OK ボタンでダイアログを閉じて完了です。

デバッグ操作

Debug->Start/Stop Debug Session メニュを選択。EL用のデバッグ設定が正しくされていれば、Input ウィンドウが開きます。

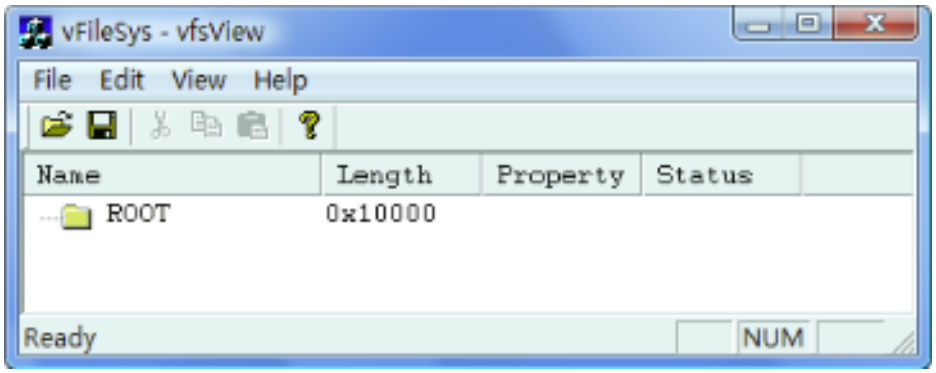


このウィンドウ上半分は EL ファイルシステムのシミュレーション設定、下半分はコミュニケーションバッファの内容表示になっています。

ファイルシステムのシミュレーション

EL ファイルシステムはプロジェクトディレクトリ内の vFileSys.dat というファイルをつかってシミュレーションされます。EXF が EL のファイルシステム関数を呼び出していて、ファイルシミュレーションを利用したければ New ボタンで vFileSys.dat を作成してください。

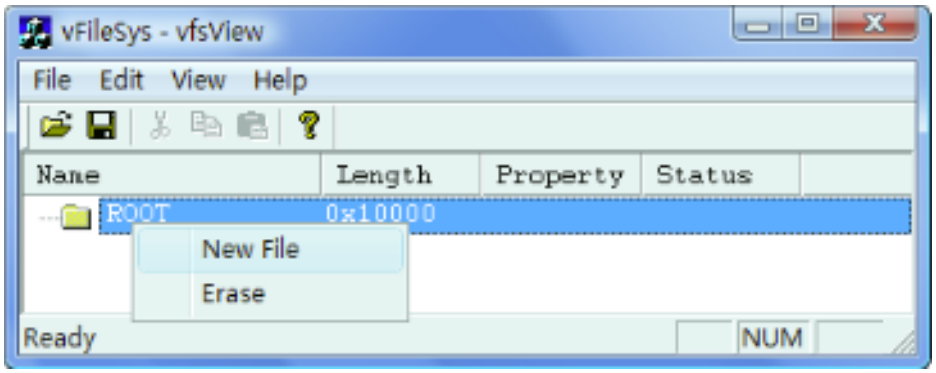
Modify ボタンでルートディレクトリに対する操作を行えます。シミュレートされたファイルシステムでは ROOT ディレクトリのみ利用できます。サブディレクトリは作成できません。



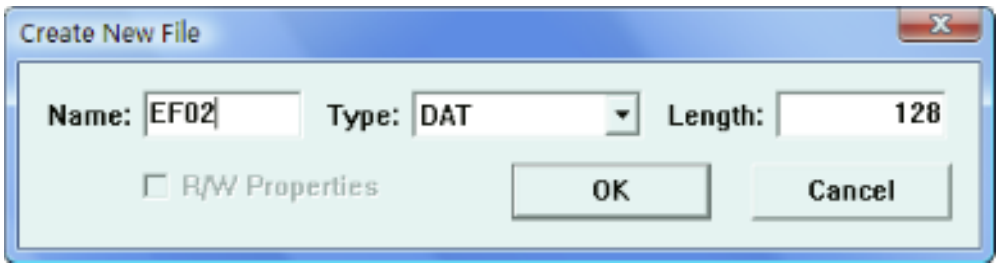
ROOT ディレクトリには 4 種類のファイルを作成できます。すべて読み書き許可になります。

1. 実行ファイル(EXF)
2. データファイル(DAT)
3. RSA 公開鍵ファイル(既定サイズ 136 バイト)
4. RSA 秘密鍵ファイル(既定サイズ 330 バイト)

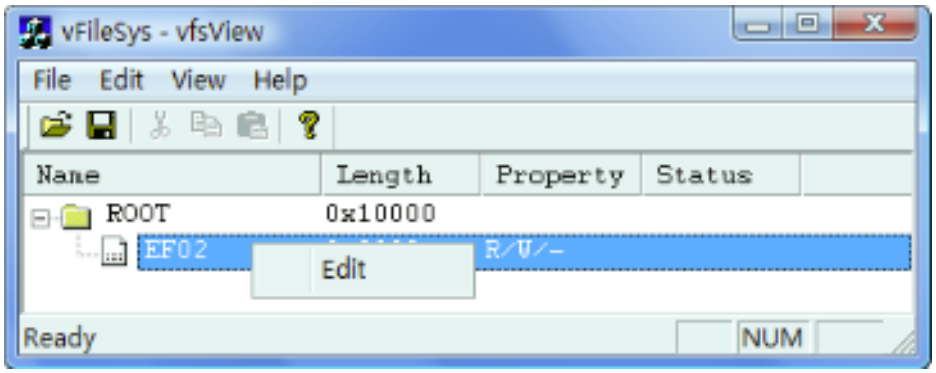
ファイルを作成するには、ROOT を選択して、右クリックで表示されるメニューで New File を選択します。



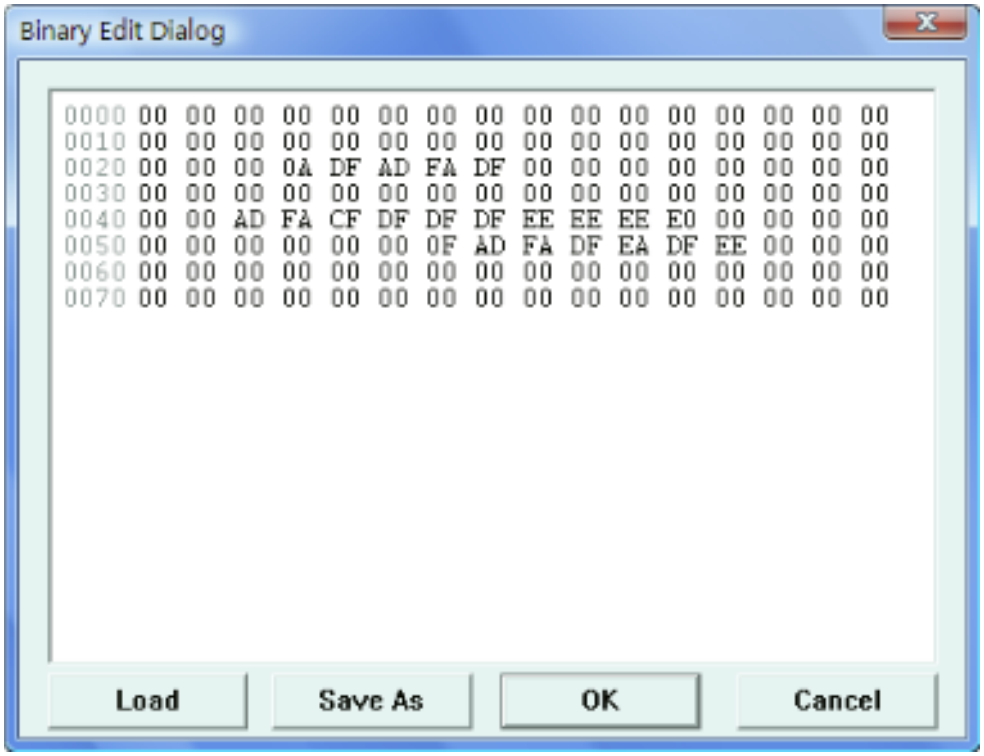
作成するファイルを指定して OK で作成が完了します。



作成したファイルを編集するには、ファイルを選択して右クリックで表示されるコンテキストメニューで Edit を選択します。



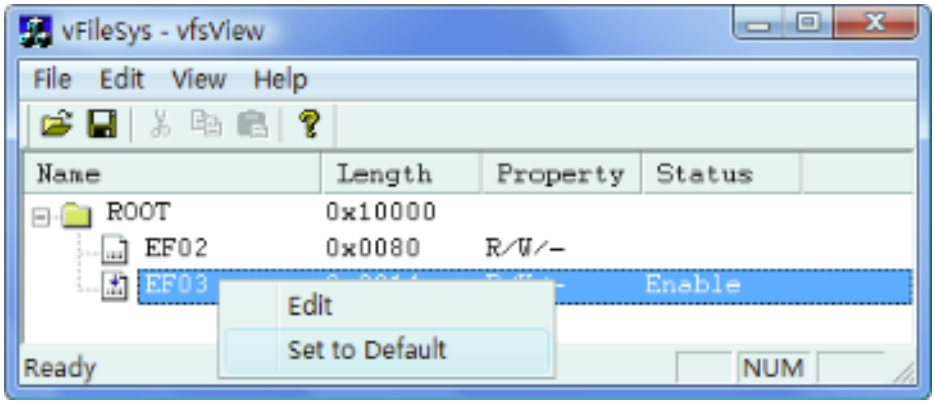
16 進データを直接編集、ディスクファイルからインポート、ディスクファイルへエクスポートなどの操作が可能です。



EXF プログラムによる自身のプログラムファイルに対する操作

実行ファイル EXF は、SES 関数 _create で自身の EXF ファイルをオープンすることができます。しかし、シミュレートされたファイルシステムには、実行中プログラムの EXF は保存されませんので、そのような _create 呼び出しは失敗します。この処理をシミュレートするには、ファイルシステムに実行ファイルを作成してデフォルトファイルとして

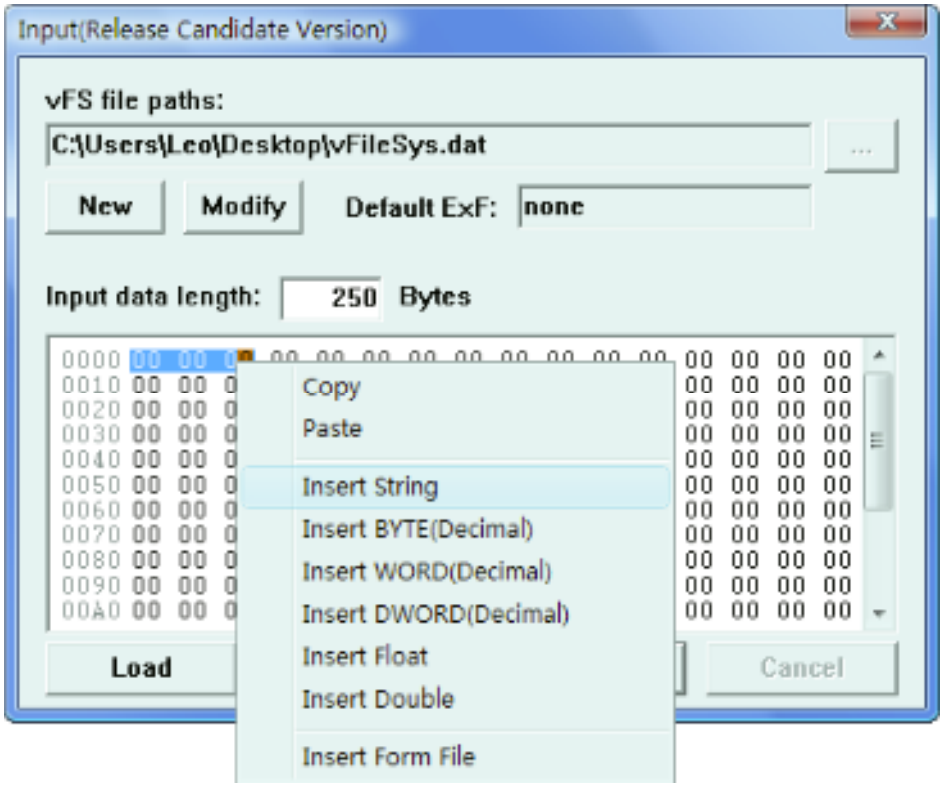
設定します（ EXF を選択後、右クリックメニュで Set To Default を選択 ）。EXF が自身のファイルをオープンすると、このデフォルトファイルが開かれます。



コミュニケーションバッファ

シミュレータは EL のコミュニケーションバッファをシミュレートします。コンピュータ側プログラムから EL に転送されるデータはシミュレータのコミュニケーションバッファにコピーされます。このコミュニケーションバッファ内容は手動で変更可能です。

1. コミュニケーションバッファウィンドウのアドレスを右クリック
2. ポップアップメニュで選択したアドレスに挿入するデータを指定



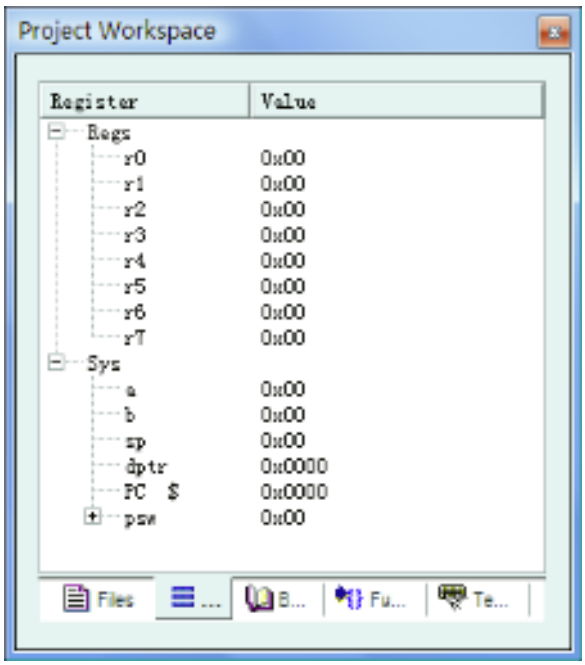
デバッグ実行

ファイルシステムとコミュニケーションバッファを設定したら、[OK]ボタンでデバッグ処理を開始します。以降 Keil 側の処理になります。詳細は Keil のマニュアルをご参照ください。

実行エラー

デバッグ中に実行エラーのダイアログが表示されたら、エラコードを確認してください。また、**View->Project Window** で表示される Project Workspace ウィンドウの Regs タブで表示される内容も確認してみてください。このタブには CPU の各種レジスタ値が表示されます。

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).



一般的に遭遇するエラー:

dptr が範囲外 (許容範囲は 0x0000 から 0x08ff)

SP オーバフロー（許容範囲は 0x07 から 0xff、スタックにデータが積まれるとポインタ値は減ります ）

PC が範囲外 (コード範囲外を指している可能性があります)

デバッグ終了

メニュー **Debug**->**Start/Stop Debug Session** でデバッグ状態から抜けることができます。

ELプログラムが SES関数 _exit() を呼び出してプログラムが終了するとデバッグが終了します。この時、ELプログラムが _set_response() でセットしたデータを表示するダイアログが現れます。

付属ツール

HEX ファイル変換ツール

Keil でプログラムをビルトすると HEX 形式の実行ファイルが出来上がります。この形式のファイルを EL は実行できません。転送する前に BIN ファイルにしなければなりません。

SDK 内に hexbin.exe というコマンドラインユーティリティが付属します。このユーティリティで HEX ファイルから BIN ファイルに変換できます。

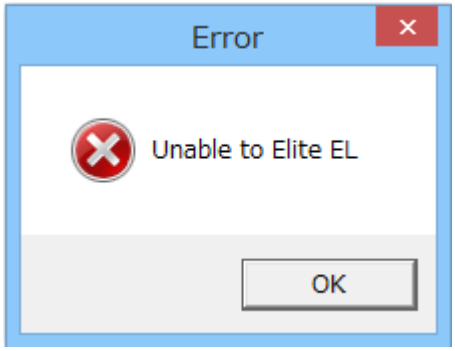
- Hexbin.exe [HEX ファイル] [BIN ファイル]

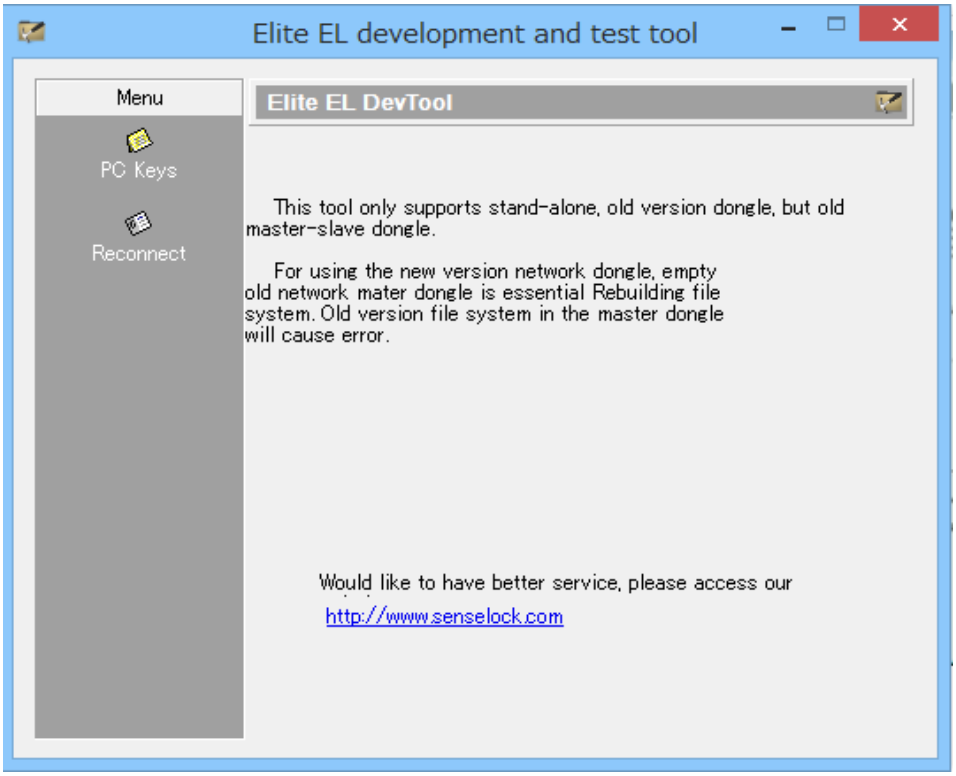
API と付属のツールは HEX が指定されると、内部で HEX ファイルから BIN ファイルに変換します。例えば S4WriteFile API にファイルタイプとして S4_HEX_FILE を指定して HEX ファイルを渡すと、BIN ファイルに変換してから EL に書き込みます。

EL デバイス設定ツール （DevTest.exe ）

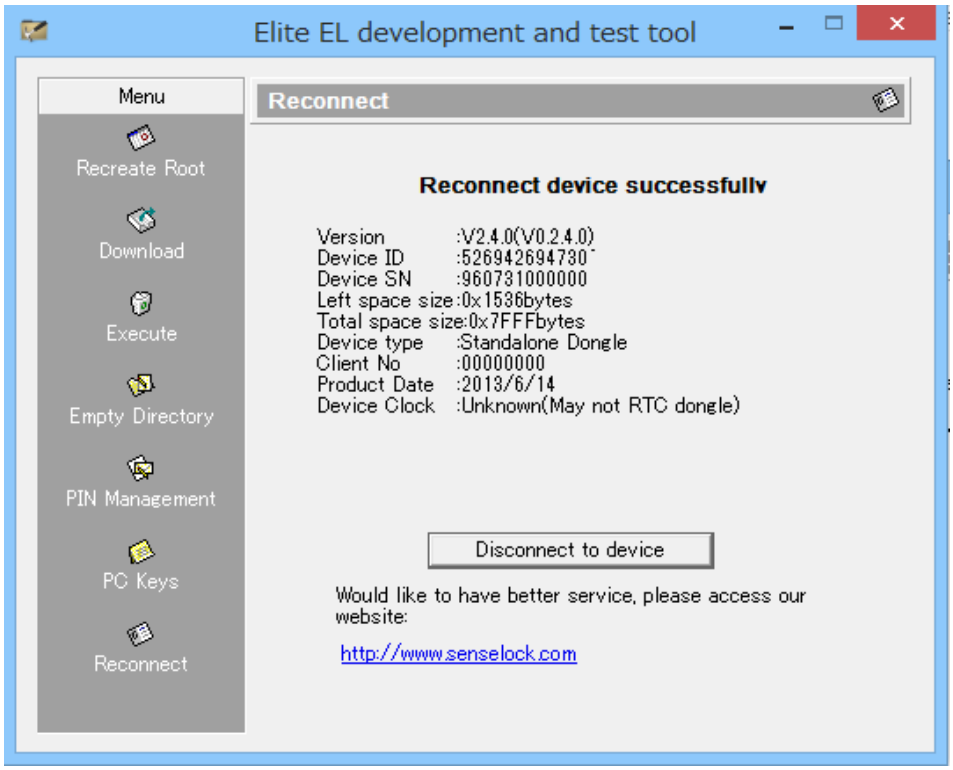
tool フォルダ内の devtext.exe を使うと GUI で EL に対して各種操作を行えます。

DevTest.exe を起動するとコンピュータに接続している EL を検索します。もし、接続していればプログラムのメインウィンドウが表示されます。接続されていなければ Error ダイアログが表示されてから、メインウィンドウが表示されます。



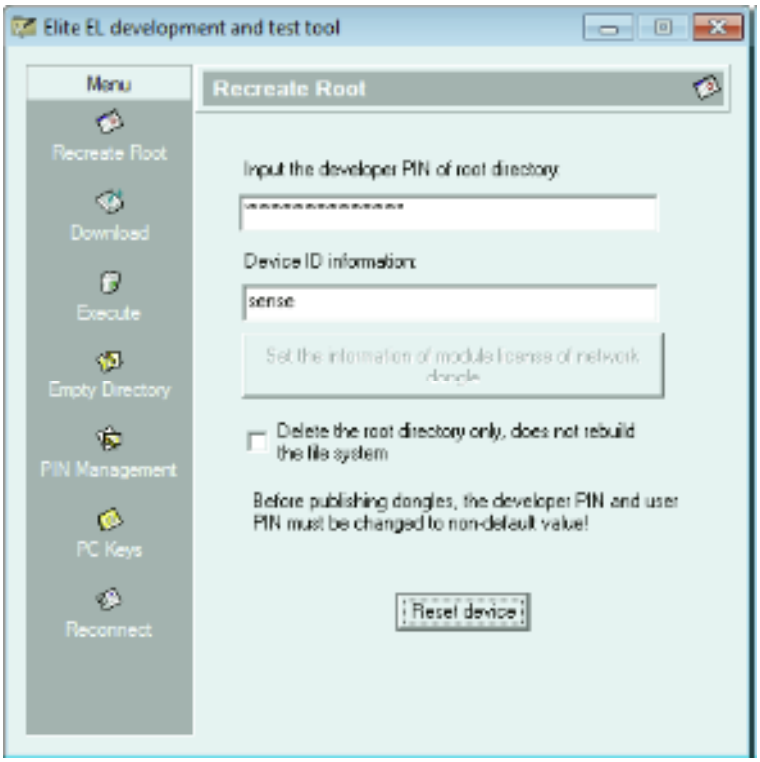


Reconnect でデバイスに接続して、接続したデバイスの情報を表示します。[Disconnect to device]ボタンで接続を解除します。



1．EL デバイスの初期化 / 再初期化

EL を初めて使う前にルートディレクトリを作成しなければなりません。また、既に利用している EL はルートディレクトリを削除すると初期化できます。DevTest では Recreate Root メニューでルートティレクトリの作成、削除が行えます。



最初のフィールド（Developer PIN） にはルートディレクトリの既定の開発者 PIN（“123456781234567812345678”）が自動入力されます。初めて EL にルートティレクトリを作成する場合、Developer PIN フィールドは既定値から変更する必要はありません。Device ID には 8 バイトの任意の ID を設定します。この ID は EL 列挙 API が取得する EL 情報の一部としてセットされます。コンピュータ側のプログラムは、この ID でプログラムが操作対象とする EL デバイスかどうかを識別できます。

ルートディレクトリ作成済みの EL デバイスでは、ルートディレクトリの正しい開発者 PIN を設定してください。

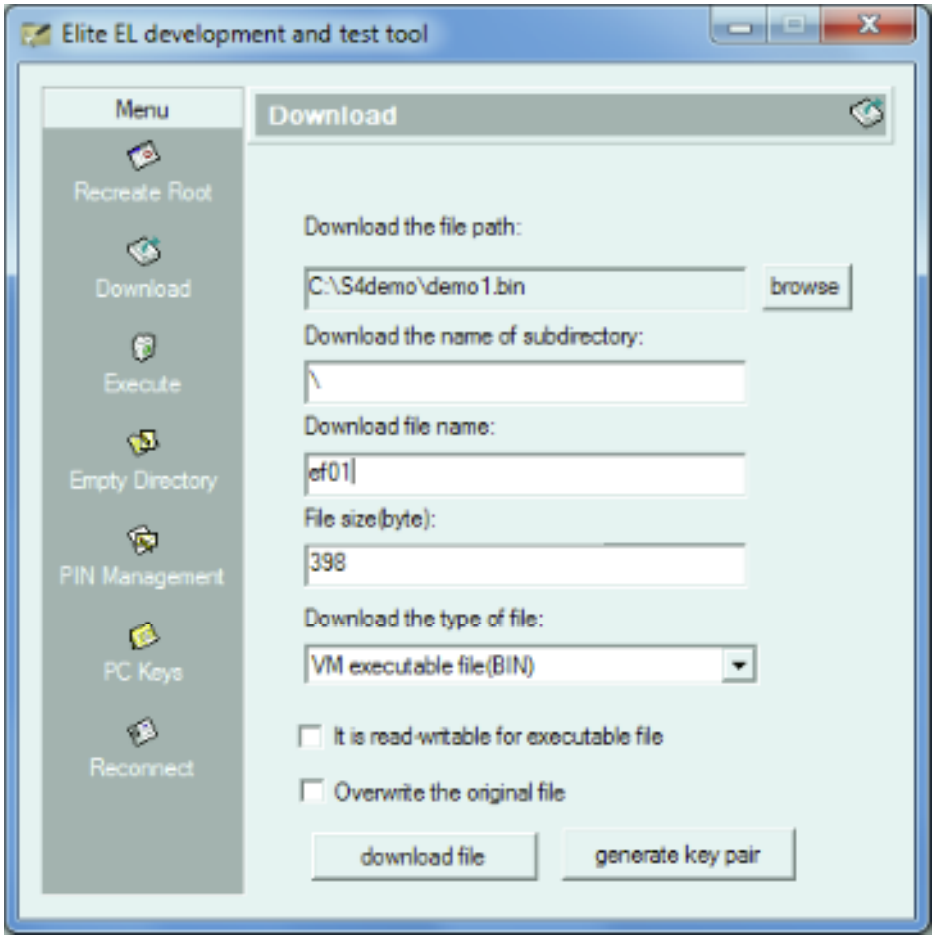
ルートディレクトリが再作成されると、ルートディレクトリの開発者 PIN と ユーザ PIN は既定値にリセットされます。なお、このツールではサブディレクトリは作成できません。

２．ファイルのダウンロード

Download メニュでコンピュータ上のディスクファイルを EL に転送できます。

転送可能なファイル（ Download　the type of file コンボで指定 ）

1. 実行ファイル(HEX/BIN)
2. データファイル
3. RSA 鍵ファイル(S4_RSA_PUBLIC_KEY / S4_RSA_PRIVATE_KEY 型)



実行ファイルは HEX か BIN 形式のファイルのみを指定可能です。HEX 形式のファイルが指定されると BIN 形式に変換して転送します。変換後の BIN ファイルはカレントディレクトリに保存されます。

RSA 鍵ファイルのフォーマットは S4_RSA_PUBLIC_KEY か S4_RSA_PRIVATE_KEY でなければなりません。このフォーマットの鍵は EL 内部の鍵の形式とは異なりますが、転送前に変換後、転送します。

転送するファイル名を指定する前にファイルタイプを選択してください。ファイルタイプ

選択後に転送ファイル名を設定すると、ファイルサイズが計算され File Size に自動的に設定さます。この値は、ファイルが実際に EL に保存されるときのサイズです。手動でファイルサイズを自動計算値より大きな値に変更しても構いません。指定サイズでファイルは作成されます。

ディレクトリとファイル名は２バイトの１６進数で指定します（ルートディレクトリは ¥ を指定します）

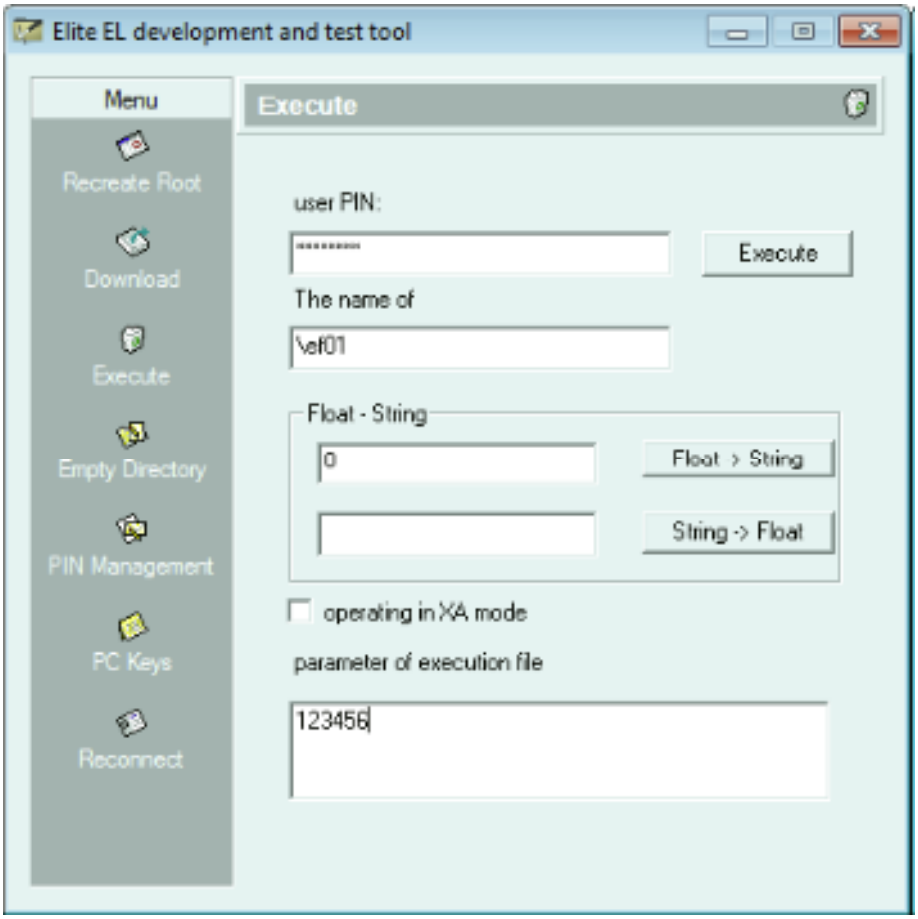
指定ファイルが実行ファイルの時、It is read-writable for executable file チェックボックスをチェックすると、実行ファイル属性が read-writable になります。既定は read only です。

EL 内の既存ファイルを更新するには、Overwrite the original file チェックボックスをチェックしてください。転送するファイルのサイズは既存ファイルサイズより小さくなければなりません。

３．実行ファイルの起動

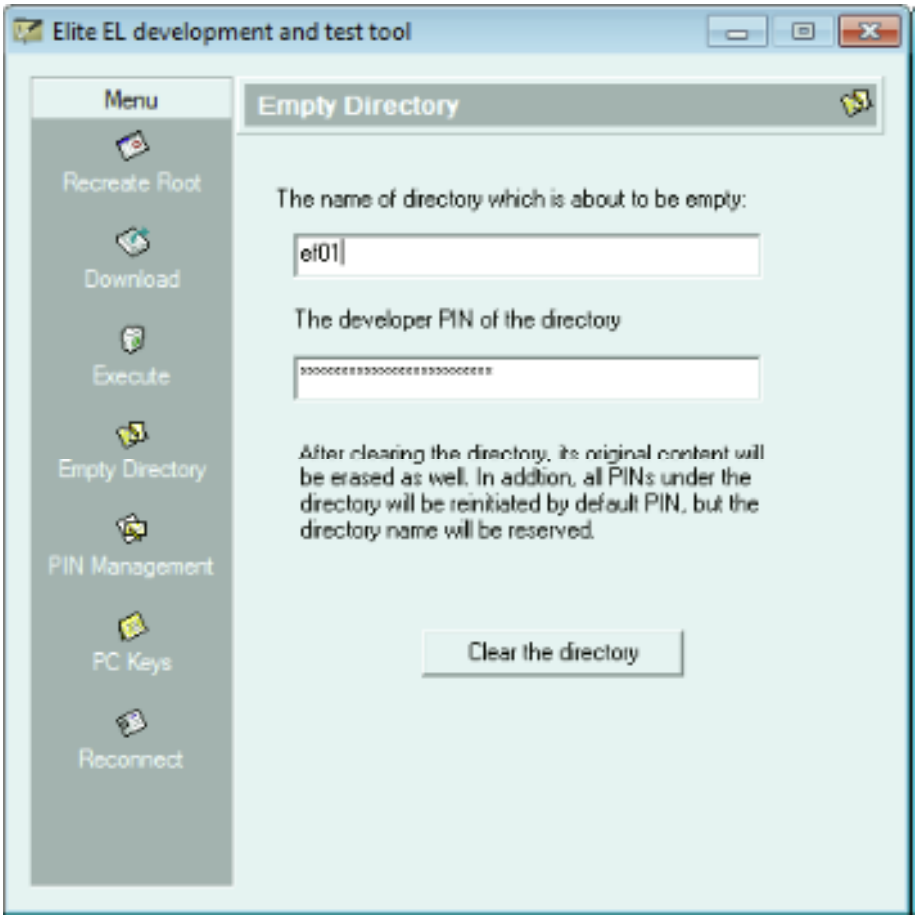
Execute メニューで EL 内の実行ファイルを起動することができます。User PIN フィールドには既定値（ 12345678 ）が自動設定されます。ファイル名 (The name of)は絶対パスで指定してください。実行ファイルへの引数は１６進で指定します。

実行ファイルからの結果は Parameter of execution file フィールドに表示されます。



４．ディレクトリのクリア

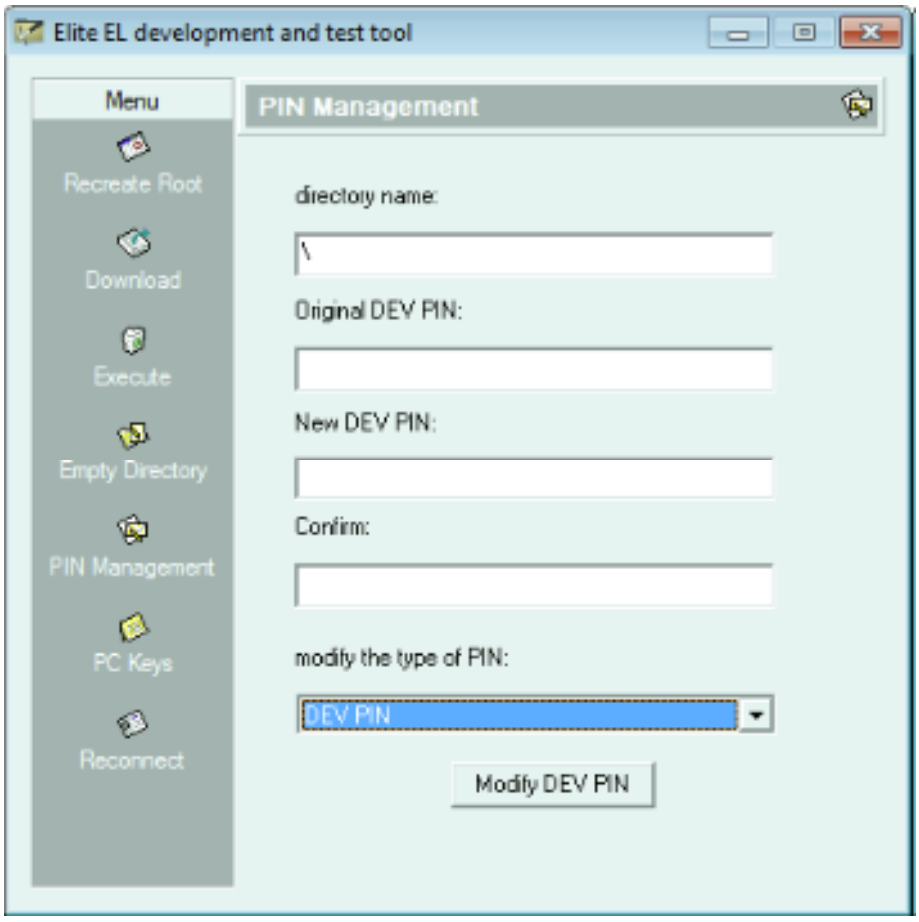
Empty Directory で、指定ディレクトリをクリア（空にすることが）できます。クリアすると指定ディレクトリの 開発者/ユーザ PIN は既定値に戻ります。



サブディレクトリをクリアしても、ディレクトリ内のファイルとディレクトリは削除されますが、削除対象として指定したディレクトリは削除されません。PIN はリセットされます。ディレクトリを削除するにはそのディレクトリを含む上のディレクトリをクリアしてください。

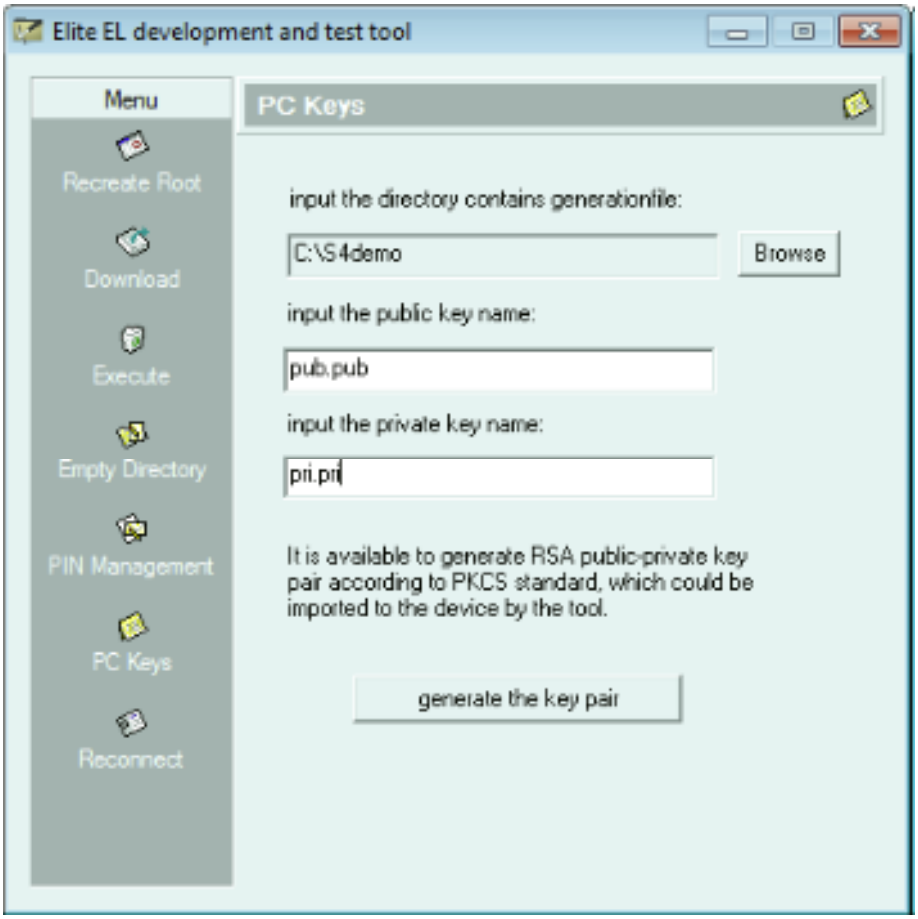
5．PIN の変更

PIN Management で指定ディレクトリの開発者 PIN とユーザ PIN を変更できます。



6．RSA 鍵の生成

PC Keys メニューで RSA 鍵ペアをコンピュータ側で生成できます。
生成される鍵は S4_RSA_PUBLIC_KEY、S4_RSA_PRIVATE_KEY フォーマットになっています。



生成した鍵ペアを保存するディレクトリとファイル名を指定してください。