ソフトウェアライセンシング導入の手引き

日本セーフネット株式会社

目次

ソフトウェアライセンシング

ソフトウェアの著作権侵害

ソフトウェアの不正コピー、カジュアルコピー、また、ライセンス認証を行わずに利用するなどの著作権侵害は、ソフトウェアベンダやメーカーの収入減を招くだけでなく、ソフトウェアの利用者がサポートなどのメーカー保証を受けられないなど、すべての人々に被害を及ぼす行為です。

ソフトウェアライセンス管理ソリューション

一般的に普及しているソフトウェア不正使用防止のためのライセンス認証方式として、いわゆる「セキュリティドングル」と呼ばれる USB トークンを利用したハードウェアベースのライセンス認証方式と「アクティベーション」によるマシンのノードロックを利用した認証方式などがあり、ソフトウェアの利用方法、配布方法などによってそれぞれが利用されています。

SafeNet では、これらのライセンス認証ソリューションを「Sentinel」ブランドの「ソフトウェアライセンス管理（SRM）ソリューション」として、世界中のソフトウェアベンダに供給しています。

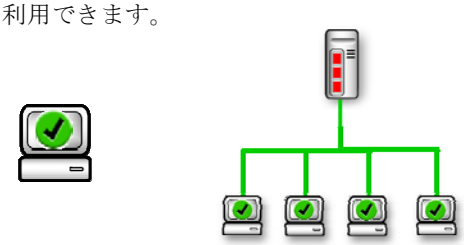
ソフトウェアライセンス管理ソリューションの導入により、ソフトウェア不正利用のリスクを軽減するだけでなく、ユーザがソフトウェアを利用するための条件を規定したソフトウェアライセンス契約が順守され、ソフトウェアによる収益の安定を図ることができます。
また、ユーザライセンスを管理することにより、ユーザのニーズをリアルタイムに把握でき、新製品や新機能を開発するタイミングを逃しません。

ライセンシングされたソフトウェアの供給は、ソフトウェアベンダの企業コンプライアンスの向上を図り、同種のソフトウェアとの差別化が生まれ、市場での優位性を確保します。

アクティベーションによるライセンス認証

「アクティベーション」は、ソフトウェアの機能を特定のマシンに「有効化」するという意味から、USB トークンなどを利用したハードウェアによるライセンス認証に対し、ノードロックによりソフトウェアの利用を特定のマシンに固定するライセンス認証方式を一般的に意味しています。

アクティベーションによるソフトウェア認証では、ハードウェアトークンが利用できない場合や、スマートフォンなど、物理的にコネクタがない機器での利用、また、Web を経由した認証を行うことで、遠隔地のマシンでの利用など、さまざまな機会でソフトウェアのライセンス認証を行うことができます。
ライセンス認証方式には、ノードロックによりスタンドアロンマシンのみを利用対象とする場合と、ネットワーク上の特定のマシンをライセンスサーバとして、利用マシンを限定しないフローティング形式、また、両者の混在などを利用できます。



スタンドアロン　　フローティング

フィンガープリント

Sentinel ソフトウェアライセンス管理ソリューションが提供するアクティベーションは、マシンを特定するための情報である「フィンガープリント」に、ネットワークアドレス、CPU シリアル、ハードディスクシリアルといった一般的な定義と、アプリケーションから設定できるカスタム定義、ハードウェアキーの ID を利用するなどの、様々な「ロックセレクタ」から任意に生成することができます。

また、仮想化環境での利用を踏まえ、クラスタの仮想イメージの識別が可能な UUID（Universally Unique Identifier）をロックセレクタに加えることなどが可能です。

Sentinel RMS

Sentinel RMS は、SefeNet が提供するアクティベーションを利用したソフトウェアライセンス管理ソリューションです。
ライセンス認証には、マシンを特定するためのフィンガープリントに、様々なシステム情報や固有情報から生成される「ロッキングコード」を利用しており、ソフトウェアの実行マシンを固定し、不正使用を防ぎます。

アプリケーションへのライセンス実装

Sentinel RMS では、アプリケーションの実行ファイルに暗号化によるラッピングを行うか、ソースコードに Sentinel RMS のプロテクションコード組み込むことによりライセンス実装を行います。
アプリケーションが暗号化され、改ざん防止が施されるとともに、アクティベーション後にマシンに実装されるライセンス情報「ライセンスプロパティ」に応じた動作が行われるように設定されます。

ロッキングコード生成

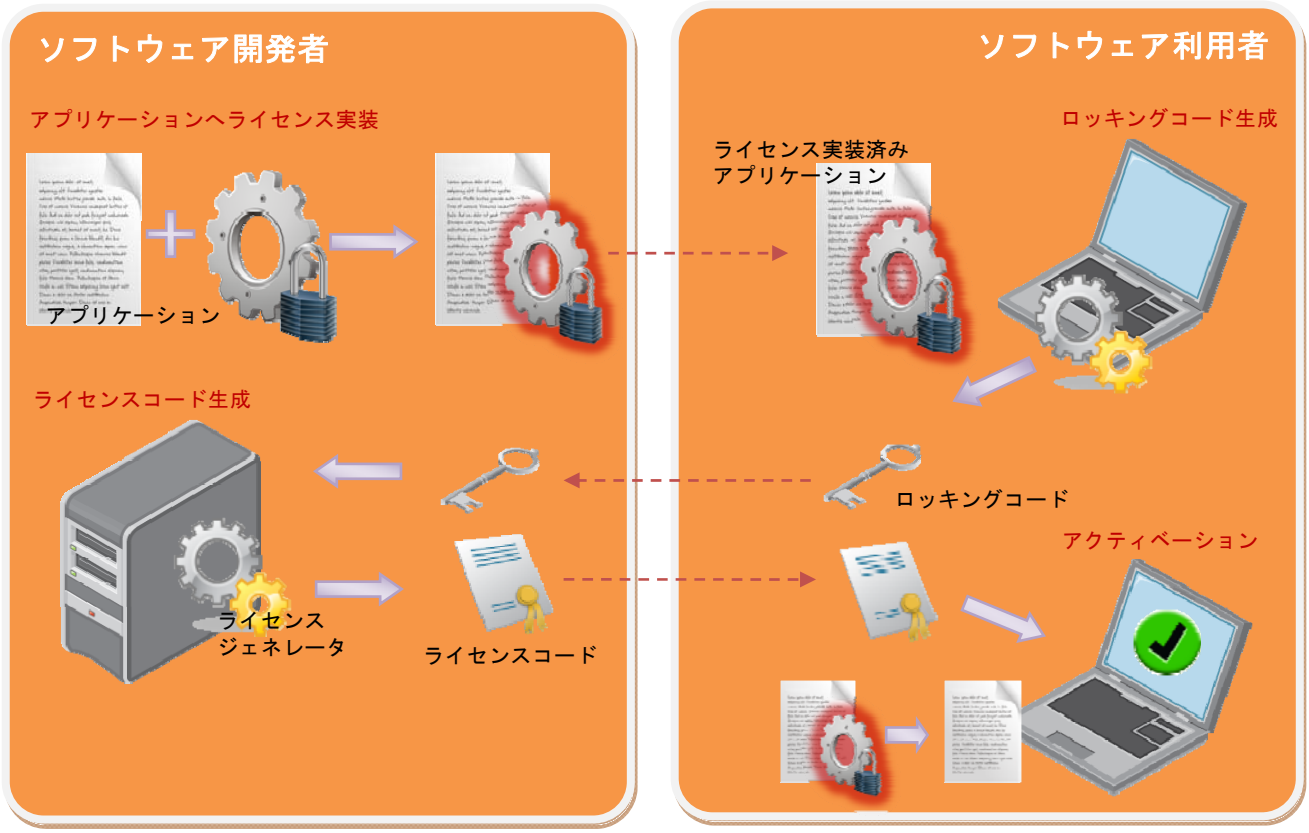
ソフトウェアの利用者は、Sentinel RMS のロッキングコード生成ツールを利用し、利用マシン（またはネットワーク）の情報から構成されるフィンガープリントをもとにシステム固有の「ロッキングコード」を生成します。

ライセンスコード生成

ライセンスモデル（永久ライセンス、トライアルなど）が設定された「ライセンスプロパティ」と「ロッキングコード」を統合して、それぞれの利用マシンに対してユニークな「ライセンスコード」を生成します。

アクティベーション

Sentinel RMS では、ライセンスコードをユーザの環境にインストールすることでアクティベーションを実現しています。
アプリケーションのライセンスモデルによっては、使用期限や実行回数などの固有のデータを利用する場合があります。 これらのデータは、実行マシンまたはライセンスサーバの「セキュアストレージ」に保管され、ライセンスインストール時にこの領域のイニシャライズが行われます。



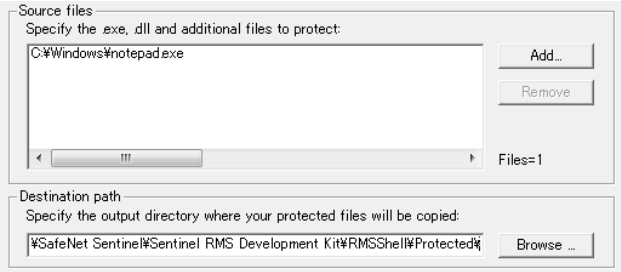
CodeCover によるプロテクション

Sentinel RMS では、アプリケーションへのライセンス実装に「CodeCover」という GUI ツールを利用できます。

CodeCover は、アプリケーションのバイナリデータを暗号化し、アクティベーションの際にロッキングコードやライセンスプロパティと紐付される「フィーチャ名」などの情報を含んだライセンスストリングとラッピングすることで、ソースコードを変更することなくライセンスを実装します。
アプリケーションを暗号化して、改ざん防止、不正使用防止の措置が取られるライセンス実装を特に「プロテクション」と呼びます。

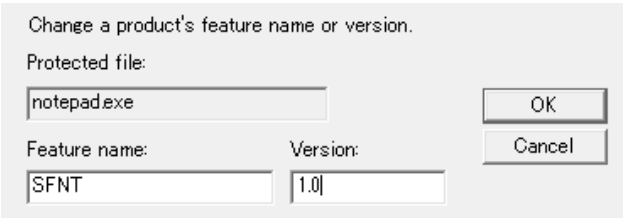
ファイルの設定

プロテクトするアプリケーションの実行ファイル（exe ファイル）またはライブラリファイル（dll ファイル）を指定します。
また、プロテクトされたアプリケーションの出力先ディレクトリを指定します



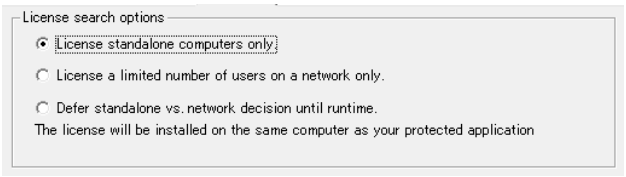
フィーチャ名とバージョンの設定

アプリケーションの使用許諾を定義するライセンスでは、アプリケーションはフィーチャ名（オプションでバージョン）によって識別されます。プロテクトする個々のアプリケーションに対して固有のフィーチャ名および、バージョンを設定します。
フィーチャ名とバージョンは大文字／小文字が区別されます。



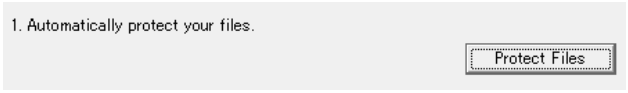
ライセンスサーチオプションの設定

アプリケーションが、スタンドアロンマシンにノードロックされるか、ネットワークのライセンスサーバにインストールされフローティングを利用するか、または、その両方について設定します。

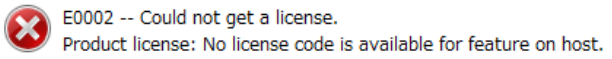


プロテクトしたアプリケーションの実行

アプリケーションのプロテクションの段階で最低限必要な設定は、前項で説明したフィーチャ名とライセンスサーチオプションです。
これらが設定されたアプリケーションが、指定された出力先ディレクトリに生成されます。



この段階で、プロテクトされたアプリケーションを実行すると、下記のようなライセンス認証エラーが発生することを確認してください。



ライセンスコードを生成し、アクティベートを行ったマシンでは、このエラーが発生せず、アプリケーションを利用することができるようになります。

ライセンスコードの生成

ソフトウェアの利用マシン、または、ライセンスサーバのシステム固有情報である「ロッキングコード」と、アプリケーションのライセンス設定を含む「ライセンスプロパティ」を統合して、利用システム（ソフトウェア利用者）に対してユニークな「ライセンスコード」を生成します。

ライセンスコードの生成は、Sentinel RMS が提供する GUI ツール「Wlscgen」ツールで行います。

ライセンスメーターキー

Sentinel RMS Development Kit パッケージ（SDK）には、ハードウェアキー（ライセンスメーターキー）が付属しています。ハードウェアキー用のドライバは SDK のインストール時に一緒にインストールされます。

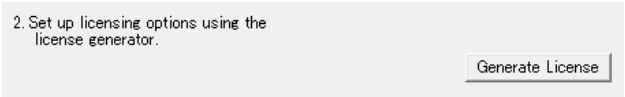
SDK をインストールしたマシンの USB ポートにライセンスメーターキーを接続します。ライセンスメーターキーは、ライセンスを発行するマシンに必ず接続しておく必要があります。

ライセンスメーターキー

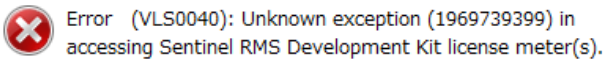
このことにより、固有のソフトウェアベンダ以外がライセンスを発行することを防ぎます。

Wlscgen の起動

Wlscgen ツールは、前述の CodeCover から起動することができます。

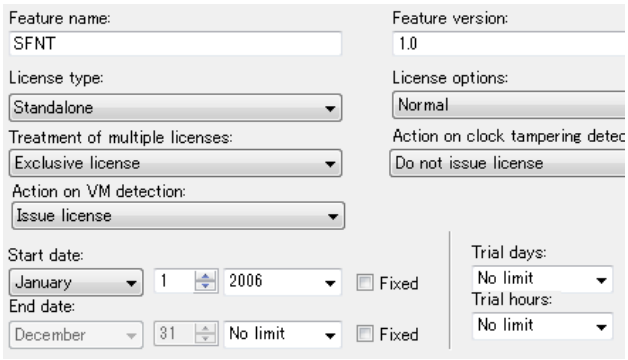


ライセンスメーターキーが正しく装着されていない場合や、Windows でハードウェアキーが検出されない場合には、次のようなエラーメッセージが表示されます。



ライセンスプロパティの設定

メーターキーが正常に検出されると、Wlscgen が起動し、ライセンス設定用のテンプレートが開きます。



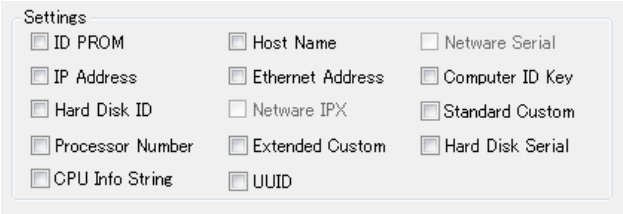
「Feature name（フィーチャ名）」と「Feature version（バージョン）」により、プロテクトアプリケーションを特定します。これが CodeCover で設定した値と合致しない場合、アプリケーションのライセンス認証は行われません。

それ以外に下記のようなライセンスプロパティを設定することができます。

`License Type`：ライセンスサーチオプション。
スタンドアロンかネットワーク、または、両方。
`Action on VM detection`：VM などの仮想環境でのアプリケーション利用の検出方法。
`Start date`－`End date`：ライセンスの有効期間
`Trial days`、`Trial hours`：ライセンスの試用期間

ロックセレクタの設定

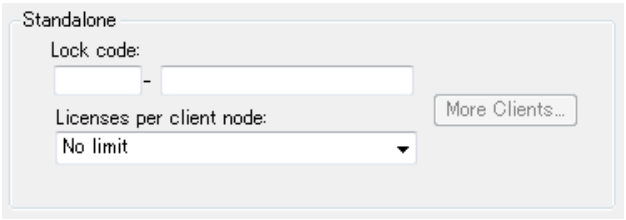
Wlscgen ツールから、実行マシン、または、ライセンスサーバをノードロックするための「ロッキングコード」のロックセレクタ（組み合わせるシステム情報の種類）を設定することができます。



複数のロックセレクタを選択して、強固なロッキングコードを設定することを推奨します。

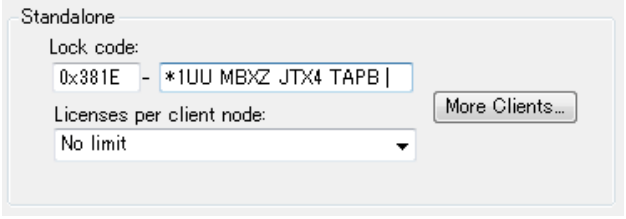
ロッキングコード入力

ライセンスコードを生成するためには、実行マシンのロッキングコードを取得し、Wlscgen ツールの「Lock code」フィールドにその値を入力する必要があります。



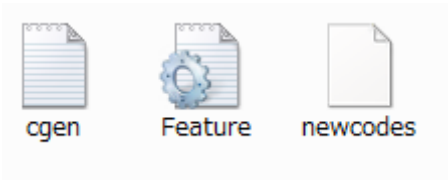
そのためには、アプリケーションを実行する実際のマシン、またはライセンスサーバからロッキングコードを取得します。この詳細は「ロッキングコードの取得」の項で説明します。

実行マシンのロッキングコードを入力することで、ライセンスコードに、ロッキングコードを取得したマシンのフィンガープリントが含まれ、これが一致するシステムでのみライセンスアプリケーションを実行できるようになります。
します。



ライセンスコード生成

すべての設定を終えると、ライセンスコードが生成されます。
ライセンスコードは、「ライセンスファイル」というテキストファイルに保存されます。
デフォルトでは「newcodes」というファイル名で保存されます。



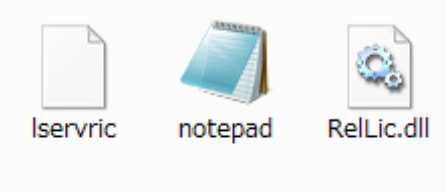
アクティベーション

ライセンスファイルをアプリケーションの実行マシン、または、ライセンスサーバへインストールすることにより、ライセンス認証が行われ、アプリケーションが利用できます。

ライセンスサーチオプションを「スタンドアロンライセンス」とした場合、最短の方法としては、ライセンスファイルをライセンスアプリケーションと同じフォルダに保存することでアクティベーションが完了します。

CodeCover でプロテクトしたアプリケーションは、デフォルトで「lservric」という名称のライセンスファイルを参照します。

Wlscgen ツールで生成されたライセンスファイル「newcodes」をファイル名「lservric」に変更します。

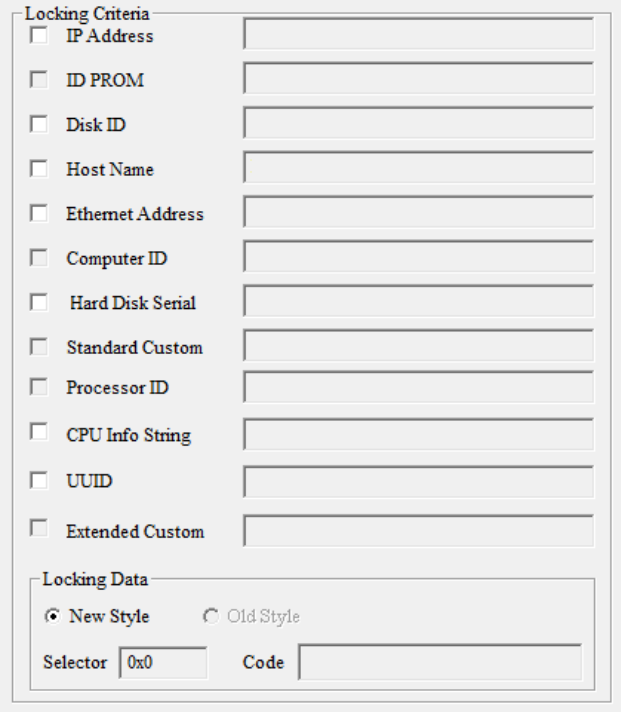


実行マシンがライセンス認証され、プロテクトされたアプリケーションファイルが実行できることを確認してください。

ロッキングコードの取得

Wechoid ツール

ロッキングコードを取得するためには、Wechoid ツールを利用します。



Wechoid ツールは、取得したいロックセレクタをチェックすることで、チェックされたロックセレクタの情報をマシンから取得し、それらのデータからそのマシンのフィンガープリントとなるロッキングコード（Locking Data）を生成します。

ロッキングコードは、選択されたロックセレクタを表す「Selector」と、マシン情報のデータを組み合わせて暗号化した「Code」で構成され、これらの両方を使用してライセンスコードを生成します。

ライセンス生成時にロックセレクタを指定した場合は、そのロックセレクタをチェックしてロッキングコードを取得します。
ロックセレクタの指定がない場合は、任意のロックセレクタをチェックしてロッキングコードを生成します。

表示された「Selector」と「Code」の値を Wlscgen ツールに入力し、ライセンスコードを生成します。

Sentinel RMS ライセンシングライブラリ API

Sentinel RMS では、CodeCover を利用したアプリケーションへのライセンス実装を行う他に、ライセンシングスキームをアプリケーションのソースコードに埋め込んでライセンス実装する「Sentinel RMS ライセンシングライブラリ API」を用意しています。
CodeCover ツールを利用した場合より、柔軟性が高いライセンシングが実現できます。

ライセンシングライブラリ API の概要

以下のコードは、ライセンス実装に使用される最も基本的なAPI 関数を示しています。

```
int main ()
{
  status = VLSInitialize();
  status = LSRequest(...);
  status = LSUpdate(...);
  status = LSRelease(...);
  status = VLScleanup();
  return 0;
}
```

VLSinitialize(): クライアントライブラリの初期化
アプリケーションの初期化中に最初に呼び出される関数です。
この関数はパラメータを持たず、正常に実行されると LS_SUCCESS というステータスを返します。

VLSsetContactServer(): ライセンス サーバの検出
特にネットワークライセンスを使用する場合に、ライセンス認証するアプリケーションが接続するライセンスサーバを指定します。
スタンドアロンライセンスの場合は、”no-net”を指定します。

LSRequest(): ライセンスリクエスト
PublisherName(会社名)、FeatureName（フィーチャ名）、Version（バージョン）などを特定して、ライセンス生成時に指定するものと一致したライセンスに接続します。リクエストが正常に実行されると、クライアントハンドルが返されます。これは、ライセンスが実装されたアプリケーションとライセンスサーバ間で確立された接続を識別するものです。

このハンドルは、それ以降のすべての接続やライブラリ呼び出しに使用されます。
このアーキテクチャによって、ライセンスが実装されたアプリケーションはライセンス サーバとの間で複数の接続を確立することができます。

LSUpdate(): ライセンス使用状況の更新
LSUpdate が呼び出されると、アプリケーションは、取得されたライセンスの有効期限が切れる前にライセンスを自動で更新します。それにより、アプリケーションが頻繁にライセンスを更新する必要がなくなります。
ネットワークライセンスでは、LSUpdateを定期的に呼び出すことにより、ライセンス更新が正常に実行されたこと、およびライセンスサーバが引き続き動作していることを確認することができます。
LSUpdate はスタンドアロンライセンスには必須ではありませんが、アプリケーションがスタンドアロンモードとネットワークモードの両方で動作する場合には LSUpdate を含めておいても問題ありません。

LSRelease()、VLScleanup()：ライセンス解放とクリーンナップ
ライセンスされたフィーチャやアプリケーションをユーザが使用し終わったら、LSRelease を呼び出してライセンス（正確にはライセンストークン）を解放します。
それにより、ネットワークライセンスの場合、他のユーザがそのライセンスを使用できるようになります。
最終的に、すべてのライセンスが解放されてプログラムが終了する準備ができた後、VLScleanupを呼び出し、割り当てられたリソースが解放可能であることがライブラリに通知されます。

ライブラリファイル

Sentinel RMS ライセンシングライブラリAPIのライブラリは、ダイナミックリンクライブラリとスタティックリンクライブラリの両方を提供しています。

lsapiw32.lib：32bit用スタティックリンクライブラリ
lsapiw64.lib：64bit用スタティックリンクライブラリ
lsapiw32.dll：32bit用ダイナミックリンクライブラリ
lsapiw64.dll：64bit用ダイナミックリンクライブラリ

これらのライブラリを使用した場合、スタンドアロンライセンスにアクセスすることも、ネットワーク上のライセンスサーバからライセンスを取得することもできます。

変数

標準的な関数を利用する場合､以下の変数を宣言します。

```
unsigned char    *license_system;
unsigned char    *publisher_name;
unsigned char    *puroduct_name;
unsigned char    *version;
unsigned long    units_read=1;
unsigned char    *log_comment;
char             *Server name;

LS_CHALLENGE     challenge;
LS_HANDLE        lshandle;
```

Sentinel RMS ライセンシングライブラリ　サンプル①　－　ライセンス取得

実行マシンにインストールされているライセンスファイルにアクセスし、Feature：SFNT、Version：1.0、が含まれるライセンスをリクエストして実況許可を取得します。

実際の運用において、アプリケーションが「Feature」、「Version」を指定して、ライセンスコードにリクエストを送り、ライセンス認証を行う場合に利用できます。

サンプルを実行する場合「ライセンスコード生成」の項に沿って、Wlscgenツールを利用し、Feature：SFNT、Version：1.0、が含まれたスタンドアロンライセンスを生成して、ファイル名：lservricとしてアプリケーションの実行ファイルと同じフォルダに保存してください。

ライセンスコードの例

```
*D
eTJN96pslfbEqiBvNV4yVzrK,0Ul1d4GWZoIuU1nmHVTkCWcicU8yUt5U:SZUCTLLTkL4iAZDbpMpbXzsCzFLUVEXa8szU9k
Jn:N2ojCvMncfd# "SFNT" version "1.0", no expiration date, exclusive
```

サンプルコード

```
void RMSSample01()
{
  LS_STATUS_CODE     status;
  LS_HANDLE          handle;

  status = VLSinitialize();
  if( status != LS_SUCCESS ){
    printf("¥n初期化に失敗しました。¥n");
    return;
  };

  status = VLSsetContactServer("no-net");
  if( status != LS_SUCCESS ){
    printf("¥nライセンスにアクセスできません。¥n");
    return;
  };

  status = LSRequest(LS_ANY, NULL, (unsigned char *) "SFNT", (unsigned char *) "1.0", NULL,
  (unsigned char *) "Requesting license", NULL, &handle);
  if( status == LS_SUCCESS ){
    printf("¥nライセンス取得に成功しました。¥n");
  }else{
    printf("¥nライセンス取得に失敗しました。¥n");
    return;
  };

  status = LSRelease(handle, 1, NULL);
  if( status != LS_SUCCESS ){
    printf("¥nライセンスリリースに失敗しました。¥n");
    return;
  };

  status = VLScleanup();
  if( status != LS_SUCCESS ){
    printf("¥n終了処理エラー¥n");
    return;
  };
}
```

Sentinel RMS ライセンシングライブラリ　サンプル②　－　フィーチャによる機能振分け

ライセンスコードに含まれるフィーチャIDを取得します。
フィーチャIDに「White」が含まれている場合「白」と表示、「Black」が含まれる場合「黒」と表示します。

実際の運用では、個々のエンドユーザに配布するアプリケーションの機能を限定したい場合などで、アプリケーションにはすべての機能を含めて配布しておき、ライセンスコードに含まれる「Feature」を各々のユーザによって変更することで、使用可能な機能を制限するようなライセンススキームを構築する場合に利用できます。

「ライセンスコード生成」の項の通り、Wlscgenを利用し、Feature：White、または、Black、Version：1.0が含まれたスタンドアロンライセンスを生成して、ファイル名：lservricとして、アプリケーションの実行ファイルと同じフォルダに保存してください。

ライセンスコードの例

```
*D
eTJN96pslfbEqiBvNV4yVzrK,0Ul1d4GWZoIuU1nmHVTkCWcicU8yUt5U:SZUCTLLTkL4iAZDbpMpbXzsCzFLUVEXa8szU9k
Jn:N2ojCvMncfd# "White" version "1.0", no expiration date, exclusive
*D
,LBYRTCRraFR2ZUhB7Dw3BgpI3FHhtYYGU4wJrOrzOLZZPIl7gamht60cWfgJE,Qf5Hy8WF2mnhWxQyeqMNT5tfaO0Dd1gIZ
KpAQCSd7OgEA3E# "Black" version "1.0", no expiration date, exclusive
```

サンプルコード

```
void RMSSample02()
{
  LS_STATUS_CODE     status;
  LS_HANDLE          handle;

  status = VLSerrorHandle(VLS_OFF);
  status = VLSinitialize();
  status = VLSsetContactServer("no-net");

  status = LSRequest(LS_ANY, (unsigned char *)"Company name", (unsigned char *)"White",
  (unsigned char *)"1.0", NULL,(unsigned char *)"Requesting license", NULL, &handle);
  if ( status != LS_SUCCESS ){
    printf("¥n白¥n");
  }

  status = LSRelease(handle, 1, NULL);
  status = VLScleanup();

  status = VLSerrorHandle(VLS_OFF);
  status = VLSinitialize();
  status = VLSsetContactServer("no-net");
  status = LSRequest(LS_ANY, (unsigned char *)"Company name", (unsigned char *)"Black",
  (unsigned char *)"1.0", NULL,(unsigned char *)"Requesting license", NULL, &handle);
  if ( status != LS_SUCCESS ){
    printf("¥n 黒¥n");
  }

  status = LSRelease(handle, 1, NULL);
  status = VLScleanup();
  return;
}
```

Sentinel RMS ライセンシングライブラリ　サンプル③　－　ロッキングコード取得

実行マシンから、ロッキングセレクタに「IP Address」と「DiskID」を選択した場合のロッキングコードを取得します。

実際の運用では、「ロッキングコードの取得」の項で利用した「Wechoid」ツールの代わりに、アプリケーションからロッキングコードを自動的に取得する場合に利用できます。

サンプルコード

```
void RMSSample03()
{
  LS_STATUS_CODE  status;

  VLSmachineID   machineID;
  unsigned long  lock_selector = 0x6; // ロッキングセレクタ：IP Address & DiskID
  char           lockCodeNew[VLS_LOCK_CODE_SIZE] = "";

  status = VLSinitMachineID(&machineID);
  status = VLSgetMachineID(lock_selector, &machineID, &lock_selector);
  status = VLSmachineIDToLockCodeEx(&machineID, lock_selector, lockCodeNew, sizeof (lockCodeNew),
   0);
  if( status == LS_SUCCESS ){
    printf("¥nロックコード: lock_selector, lockCodeNew¥n");

  }else{
    printf("¥nロックコード取得エラー : status¥n");
    return;
  };
}
```