WIBU-KEY
ワイブキー

ユーザーズガイド　Ver. 7.11
（プロフェッショナル版・スケルトン版対応）

No. 3006

2011 年 9 月 27 日

サンカーラ株式会社
www.suncarla.co.jp

目次

Chapter 1

はじめに

1-1. はじめに

このたびは、高機能コピープロテクトツール「ワイブキー」をご購入いただき、誠に有難うございます。この「ワイブキー」は、世界最高レベルのセキュリティ機能を持つ最も優れたコピープロテクトツールです。必ず、貴社のセキュリティニーズにお応えできるものと確信しております。

1-2. ワイブキーの特長

ワイブキーには、不正コピーを防止するために必要な機能が豊富に搭載されています。

1. 最強の自動暗号化ツール「AxProtector」を用意

Windows 32bit/64bitアプリケーション、.NETアセンブリ、Mac OS X、Javaアプリをプログラムのソースコードを変更せずに、強力に自動暗号化する「AxProtector」が用意されています。アンチデバッグ機能やアドバンストセキュリティ機能などプロテクトに必要な高度なセキュリティ機能が豊富に搭載されています。また、IxProtector/WUPIと組み合わせることにより、メモリー上の「オンデマンド復号」を実現します。

2. メモリー上での「オンデマンド復号」を実現する IxProtector 機能搭載

メモリー上で展開されるプログラムコードを常に暗号化しておき、必要な時に必要なファンクションモジュールを復号し、実行したあとは再び暗号化しておく「オンデマンド復号」機能を搭載。AxProtectorで暗号化されたプログラムコードが、メモリー上でも常に暗号化されているため、クラッキングに対して非常に強力なセキュリティを実現できます。

3. WUPI ファンクションが登場

コードメータと共通で使用できるユニバーサルなAPIファンクションWUPI (Wibu Universal Protection Interface)が登場しました。AxProtector/IxProtectorと組み合わせて、メモリー上の「オンデマンド復号」を実現するだけでなく、モジュール単位のライセンス管理、「Pay per Use」による課金システム管理が行えます。

4. ２種類のコードによる確実なプロテクト

ワイブキーに登録されたファームコード/ユーザーコードと、プログラムに組み込まれたファームコード/ユーザーコードを照合します。この２種類のコードが一致しないとプログラムは起動しません。ファームコードは貴社専用のコードになります。

5. 1 個のキーに 10 種類の異なるコードを登録可能

1個のワイブキーに異なるコード（ファームコード/ユーザーコード）を10種類まで登録できます。さらに、マスターコード機能を使うことで、最大190種類までのコード登録が可能です。
* スケルトン版の場合は、登録できるコードは１種類までです。

6. 貴社専用の FSB (Firm Security Box)

ワイブキーにコードを登録するために必要なFSB(Firm Security Box)は、貴社専用です。第三者が不正に貴社のコードを使ってワイブキーを作成することができません。確実に、貴社独自のセキュリティを実現します。

7. 強力なアンチデバッグ機能

ハッカーによる解析を防ぐために、強力なアンチデバッグ機能が搭載されています。長年によるハッキング対策の経験から、考えられるアンチデバッグのノウハウをできるだけ多く搭載しております。

8. 豊富なセキュリティオプション

プログラム暗号時のオプション機能として、使用回数制限、使用有効期限、ランタイムチェック機能、アンチデバッグ機能、ウイルスによる改ざんチェック機能、拡張メモリーなど、セキュリティニーズに応じた多数のセキュリティオプションを用意しています。

9. 高機能なコア API を多数用意

自動暗号化ツール「AxProtector」やユニバーサルなWUPIファンクションのほかに、ワイブキー専用のコアAPIファンクションを多数用意しています。これらのAPIは、32ビット/64ビットのWindows/Linux/Macに共通なクロスプラットフォームセキュリティAPIで、WUPIファンクションとの連携が可能です。WUPIファンクションでは行えないきめの細かいプロテクトチェックに利用できます。

10. リモートアップデート機能

遠隔地からファイル操作だけで、ユーザー先にあるワイブキーの内容を更新することができます。更新のために、キーを送ったり、送り返す必要がないため、商品コード（ファームコード／ユーザーコード）の追加更新、使用期限の更新、使用回数の更新などがスピーディに行えます。

11. 1 個のキーで最大 2,250 台までのクライアント制御が可能

ネットワーク上のサーバーにワイブキーを1個装着することで、1台 - 2,250台の範囲でクライアントライセンス（フローティングライセンス）を制御することができます。ライセンス数は、ワイブキーの中に登録します。ローカルPCに１つ１つ装着する必要がないので、プロテクトコストを大幅に削減することが可能になります。（このネットワーク機能はプロフェッショナル版のみで、スケルトン版にはありません。）

12. RoHS 指令対応済み

ワイブキーは、RoHS指令で規制されている鉛、水銀、カドミウム、六価クロム、ポリ臭化ビフェニール、ポリ臭化ジフェニルエーテルの使用基準を満たしているWEEE/RoHS指令適合製品です。

13. ISO9001:2008 認証取得

ワイブキーは、ISO9001:2008認証されたWIBU-SYSTEMS社で開発・製造されており、高い品質と信頼性をご提供いたします。

1-3. プロフェッショナル版とスケルトン版の違い

ワイブキーには、プロフェッショナル版とスケルトン版の２種類のタイプがあります。プロフェッショナル版（PRO版）は、従来より販売しているワイブキーの新しい呼称で、機能的には従来のワイブキーと変わりません。スケルトン版は、あらたに追加されたUSBタイプで、プロフェッショナル版に比べ、ある一部の機能を削っています。ケース本体がスケルトングリーンのためスケルトン版と呼称しています。

▶型名

プロフェッショナル版（PRO版）には、下記のタイプがあります。
WKUSB (USBポートタイプ)
WK25P (LPT/プリンタポートタイプ)
WK25ST (COMポートタイプ)

スケルトン版には、下記のタイプがあります。
WKUSB/R (USBポートタイプ)

▶登録できるエントリの数

プロフェッショナル版は、登録できるエントリ数は10個です。
スケルトン版は、登録できるエントリ数は1個です。

▶ファームコード

プロフェッショナル版のファームコードは、４桁の整数値です。
スケルトン版のファームコードは、６桁の整数値で、上２桁は25ではじまります。(25xxxx)

▶ FSB (Firm Security Box)

プロフェッショナル版FSBとスケルトン版FSBは互換性がなく、それぞれ独立しています。プロフェッショナル版FSBを使って、スケルトン版ワイブキーを編集したり、スケルトン版FSBを使ってプロフェッショナル版ワイブキーを編集することはできません。従い、プロフェッショナル版のファームコード・ユーザーコードをスケルトン版に登録したり、スケルトン版のファームコード・ユーザーコードをプロフェッショナル版に登録することはできません。

▶開発ツールとランタイムキット

自動暗号化ツールAxProtectorやAPIなどの開発ツール、ユーザー側で必要なランタイムキットなどはプロフェッショナル版とスケルトン版は共通です。

▶スケルトン版に無い機能

スケルトン版に無い機能は下記になります。

１. ネットワーク機能
ネットワーク機能をサポートしていません。サーバーに装着し、複数のクライアントを制限することはできません。スケルトン版は、必ずローカルポートに装着して使用する必要があります。

２. マスター機能
マスター機能をサポートしていません。１個のキーには、１種類のコード（ファームコード/ユーザーコード）しか登録できません。

上記２項目以外は、プロフェッショナル版と同じ機能が使用できます。

機能比較表

機能	プロフェッショナル版 WKUSB, WK25P, WK25ST	スケルトン版 WKUSB/R
AxProtector / IxProtector	○	○
WKLIST32	○	○
ファームコード	4 桁	6 桁
ユーザーコード	0 ～ 16777215	0 ～ 16777215
マスター機能	○	×
エントリ数	10	1
ネットワーク機能	最大 2,250 クライアント	×
FSB	プロフェッショナル版専用	スケルトン版専用
使用回数制限	○	○
使用期限	○	○
ユーザーデータ	○	○
リモートアップデート機能	○	○
API ファンクション	○	○
ランタイムチェック機能	○	○
拡張メモリー	○	○
アンチデバッグ機能	○	○
ウィルスチェック機能	○	○
対応 OS (64/32bit)	Windows/Linux/Mac	Windows/Linux/Mac
ランタイムキット	○	○

1-4. 前ユーザーズガイドからの主な変更点

No.3006 (2011 年 9 月 27 日変更)

1. AxProtector 7.11にバージョンアップ

AxProtectorが6.50から7.11にバージョンアップし、下記の点が機能追加されました。

- 「アドバンス」メニューに、「IxProtectorのみ(.NET)」と「Mixed Mode」を追加。
- .NET 4.0 Frameworkをサポート。
- Native Codeと.NETのMixed-Modeアセンブリを含む全ての.NETアセンブリの自動暗号化が可能になりました。
- Win32/Win64のMixed-Modeアセンブリの自動暗号化が可能になりました。
- 静的なスレッドローカルストレージ(TLS)を利用したアプリケーションの暗号化がより安定しました。
- PEヘッダの自動拡張。セキュリティを高める際、PEヘッダに十分なスペースが確報できない場合に自動的にPEヘッダを拡張します。
- User Message for .NETの機能拡張。
- Mac OS X 10.6の自動暗号化をサポート。
- Linux 32bit/64bitの自動暗号化をサポート。
- その他バグ修正。

2. ワイブキーランタイムキットをバージョンアップ

ワイブキーランタイムキットを6.00aから6.00bにバージョンアップし、下記の点を解決しました。

- Windows 7 (64bit) における日本語アカウントでのインストール問題を解決。(Windows)[FB10375]
- CD/DVD上からのSetup32/64.exeのインストール問題を解決。(Windows)[FB8338]
- Windows XP上でのワイブキーサーバーマネジャーの文字化けを解決。(Windows)[FB10376]
- Windows Vista/7上でのワイブキーサーバーマネジャーの日本語化を解決。(Windows)[FB12000]

No.3005 (2010 年 2 月 25 日変更)

1. AxProtector 6.50にバージョンアップ

AxProtectorが6.40cから6.50にバージョンアップし、下記の点が追加されました。

- 「データファイル暗号化」機能を正式リリース。

AxProtectorのアドバンスオプション/"ファイル暗号化を有効にする"オプションが利用できるようになりました。暗号化したアプリケーションが利用するデータファイルに対しても暗号化（プロテクト）することが可能になります。

2. ワイブキーランタイムキットにWindows OS 32bit/64bitの統合版を追加

Windows OSが32bitか64bitかを自動判断し、そのOSに合わせた内容をインストールする統合版ランタイムキットWkRuntime.exeを追加し、ランタイムキットが３種類になりました。

WkRuntime.exe (Windows OS 32bit/64bit兼用: 18.6MB)
WkRuntime32.exe (Windows OS 32bit専用: 12.9MB)
WkRuntime64.exe (Windows OS 64bit専用: 13.9MB)

3. 64bit OS (Windows Vista/7, Server 2008, Mac OS X, Linux) 環境下での安定性をより強化

- 64bit OS環境下での一部バグ修正を行い、安定性とセキュリティをより強固にしました。

No.3004 (2009 年 4 月 15 日変更)

1. AxProtector 6.40cにバージョンアップ

AxProtectorが6.30から6.40cにバージョンアップし、下記の点が追加されました。

- .NETアセンブリがWUPI+IxProtectorをサポート (FB18)
- WupiQueryInfo()/WupiQIMappedLastError()による詳細なエラーハンドリングを提供
  WibukeyエラーをCodeMeterエラーに転送可能 (FB3100)
- CmAccess()とWkbAccess2()によるサーバーリストのバッファリングをサポート (FB119)
  (複数のLANアクセスが存在する場合)
- WupiQueryInfo()にWupiQICopyProtectionSystemフラグが追加されました。(FB3180)
  (WupiGetLicenseType()と同機能)
- 一部バグ修正

スケルトン版WKUSB/Rによる.NETアセンブリ自動暗号化において、暗号化アルゴリズム=5 (reduced license)を正式サポート。

2.「Chapter 3 sample.exeにプロテクトをかける」の一部加筆修正

3.「Chapter 5 自動暗号化ツール AxProtectorについて」

- 「Hotfixについて」を追加 (5-1. 自動暗号化ツール AxProtectorについて)
- 「6. エラーメッセージ」の内容を修正 (5-4. AxProtectorの各入力画面の説明)

4. Mac OS Xの開発キット・ランタイムキットをバージョンアップ

- 「13-2. Mac上でワイブキーを使用する場合」一部加筆修正

No.3003 (2009 年 2 月 12 日変更)

1. AxProtectorが6.21からAxProtector6.30にバージョンアップ

AxProtectorが6.30にバージョンアップし、下記の点が追加されました。

- 「5. セキュリティオプション」/「アンチデバッグの設定」で、Generic debugger detectionが追加。
- 「5. セキュリティオプション」/「アンチデバッグの設定」で、Virtual machine detectionが追加。
- 「Advanced option」で、IxProtector設定を改良。

2. WUPI/IxProtector

「Chapter 10 IxProtectorについて」を新規追加。

No.3002 (2008 年 2 月 11 日変更)

1. スケルトン版ワイブキーの追加

従来のワイブキー (プロフェッショナル版) にスケルトン版WKUSB/Rが加わりました。

2. AxProtectorが6.0からAxProtector6.21にバージョンアップ

AxProtectorが6.21にバージョンアップし、下記の点が更新されました。

- .NETアプリの自動暗号化をサポート
- 日本語画面のサポート
- IxProtectorの画面の追加

３.「Chapter6 ネットワーク機能について」の「6-2. ベーシック方式」の内容を一部修正
サポートするクライアント数が最大250台から最大2,250台になりました。（ベーシック方式）

４.「Chapter3 コード設定ツール Wklist32.exeについて」の「3-3.マスター機能について」の内容を一部修正
マスター機能の説明部分をより分かりやすく修正しました。

Chapter 2

ワイブキー開発ツールをインストールする

2-1. ワイブキー開発ツールをインストールする

2-1. ワイブキー開発ツール (SDK) をインストールする

ワイブキー開発ツール(SDK)のインストール作業は以下の2ステップを行ないます。

1. ワイブキー開発ツール Ver6.0cをインストールする
2. ライセンスファイルWkFirm.wbcをコピーする

1. ワイブキー開発ツール (SDK) をインストールする

① ワイブキーCDを起動する

ワイブキーCD をCD/DVDドライブに挿入すると右の画面が自動的に立ち上がります。自動的に立ち上がらない場合は、エクスプローラから"CDStart.exe"をクリックしてください。"CDStart.exe"は、ワイブキーCDのルートにあります。

右の画面で、"English"をクリックします。

② ワイブキーSDKをインストールする

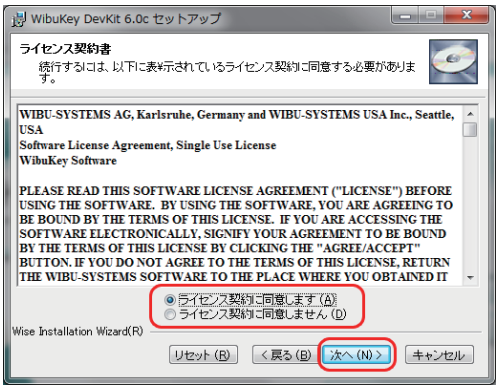
右の画面で「WibuKey SDK」をクリックします。

③ウィザード画面が立ち上がる

「WibuKey DevKit 6.0c セットアップ」画面で、[次へ]をクリックします。

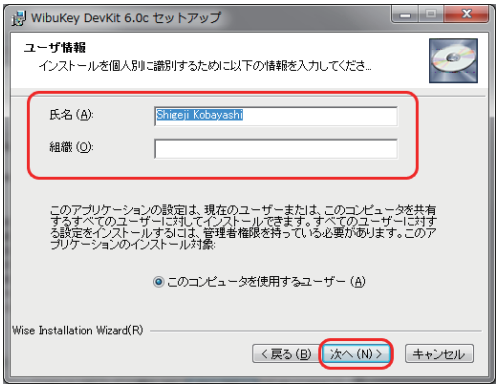
④ライセンス契約書

ライセンス契約書が表示されます。ライセンス契約に同意のうえ、次に進みます。



⑤ユーザー情報を入力する

ユーザー名などを入力します。

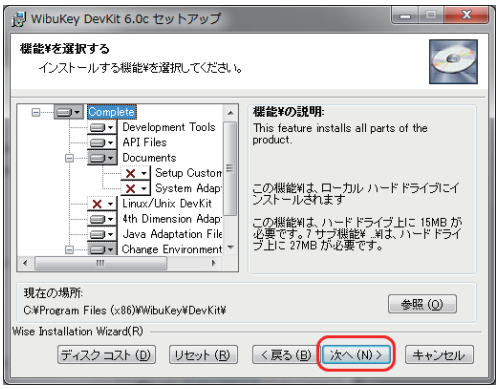


⑥インストール先を指定する

インストール先のフォルダに変更がなければ、そのまま[次へ]をクリックします。

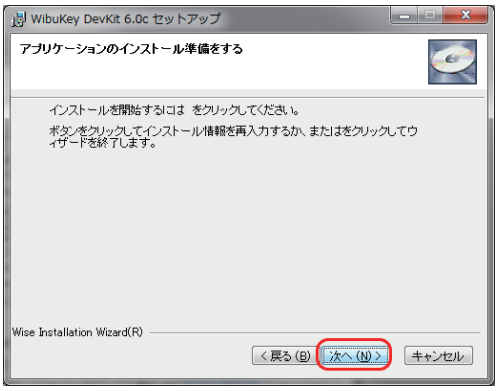
⑦インストールする機能を選択する

インストールする機能を選択します。ここでは、デフォルトのまま進めてください。



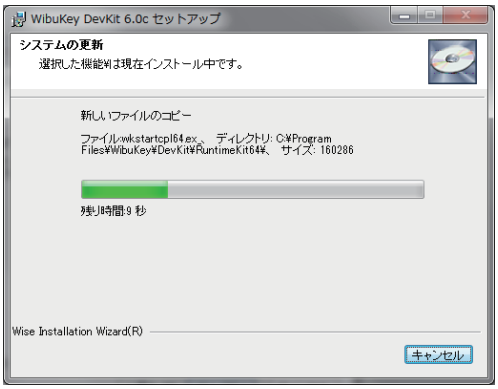
⑧インストールを開始する

「アプリケーションのインストール準備をする」画面で［次へ］をクリックしてください。ワイブキー開発キットがインストールされます。



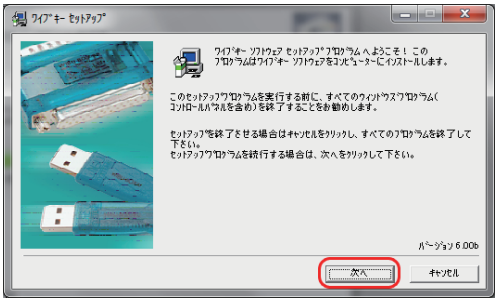
⑨インストール作業が行われる

インストール作業が開始されます。インストール作業が開始されるまで、数秒待たされることがありますが、そのままお待ちください。



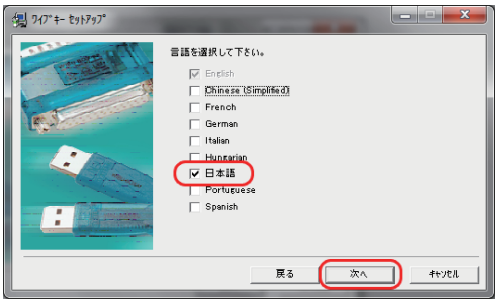
⑩ランタイムキットをインストールする

続けて、ワイブキーランタイムキットのインストールを行います。右の画面が表示されたら、「次へ」ボタンをクリックします。



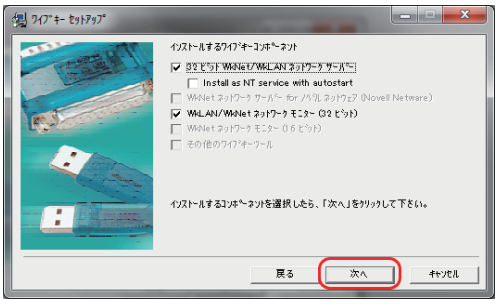
⑪日本語を選択する

言語選択画面から、日本語（デフォルト）にチェックを入れます。



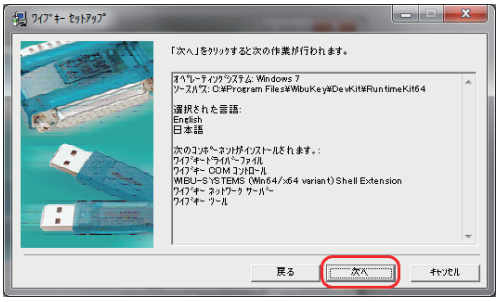
⑫インストールするコンポーネントを選択

インストールするワイブキーコンポーネントを選択します。通常はデフォルトのまま（推奨）、「次へ」をクリックします。



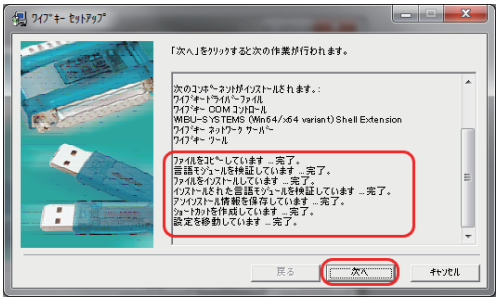
⑬内容を確認し、次へ進める

内容を確認し、「次へ」ボタンをクリックします。



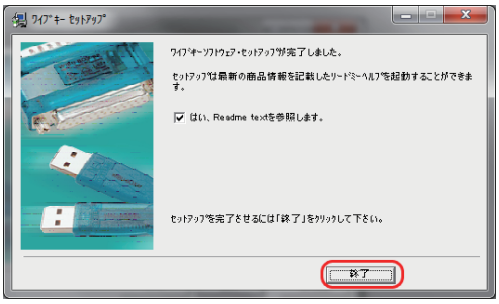
⑭必要なファイルがインストールされる

必要なファイルのインストールが完了します。「次へ」ボタンをクリックします。



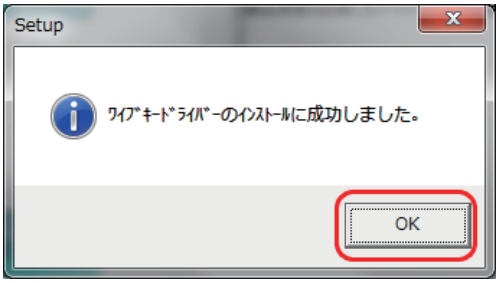
⑮インストールが完了する

インストールが完了すると、右の画面が表示されます。「終了」ボタンをクリックします。



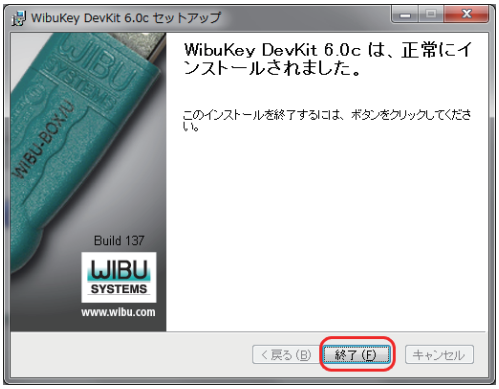
⑯無事にインストール成功

「終了」ボタンをクリックすると、右の画面が表示され、ワイブキーランタイムキットが無事にインストールされました。



⑰ DevKit インストール画面を閉じる

「終了」ボタンをクリックして、「WibuKey DevKit 6.0c セットアップ」画面を閉じます。



これでワイブキー開発ツール Ver6.0cのインストールは完了です。
次に、同梱のライセンスCDから貴社専用のライセンスファイル（WkFirm.wbc）をコピーします。評価版(WKUSBまたはWKUSB/R)の場合は、この作業は必要ありませんので、次の「Chapter3 sample.exeにプロテクトをかける」に進んでください。

3. ライセンス CD から貴社固有のライセンスファイル WkFirm.wbc をコピーする

ワイブキー開発キットに同梱されているライセンスCDをPCのドライブに入れ、貴社のライセンスファイルWkFirm.wbcを、先ほどインストールしたWibukeyフォルダのサブフォルダDevKitの下のBinフォルダ(¥Program Files¥Wibukey¥Devkit¥Bin)の中に上書きコピーします。Binフォルダは、Wibukeyフォルダのすぐ下にも存在しますが、**必ずDevKitの下のBinフォルダ**に保存するようにしてください。Binフォルダの中には既にWkFirm.wbcファイルが存在していますが、これは評価用のファイルですので、ここで貴社専用のライセンスファイルWkFirm.wbcを上書きします。

［注意］
「評価版」をご利用のお客様は、ライセンスCDは付属しません。すでにBinフォルダに存在している評価用ライセンスファイルWkFirm.wbcをそのままご使用ください。（プロフェッショナル版、スケルトン版ともに使用できます。）

［注意］
WkFirm.wbcファイルの保存場所を間違えると、ワイブキーにコードが書き込めませんので、必ずDevKitの下のBinフォルダに保存するようにして下さい。Binフォルダは２つ存在しますので、ご注意ください。

Chapter 3

sample.exe にプロテクトをかける

3-1. sample.exe にプロテクトをかける

ワイブキーには、実行形式プログラムを強力に暗号化する「AxProtector」が用意されています。Windows 32bit/64bitプログラム、.NET プログラム、Mac OS X、Javaアプリをプログラムのソースコードを変更せずに自動的に暗号化できます。また、必要な時に必要なモジュールを復号して使用するメモリー上における「オンデマンド復号」機能も追加され、ますますセキュリティが強化されています。

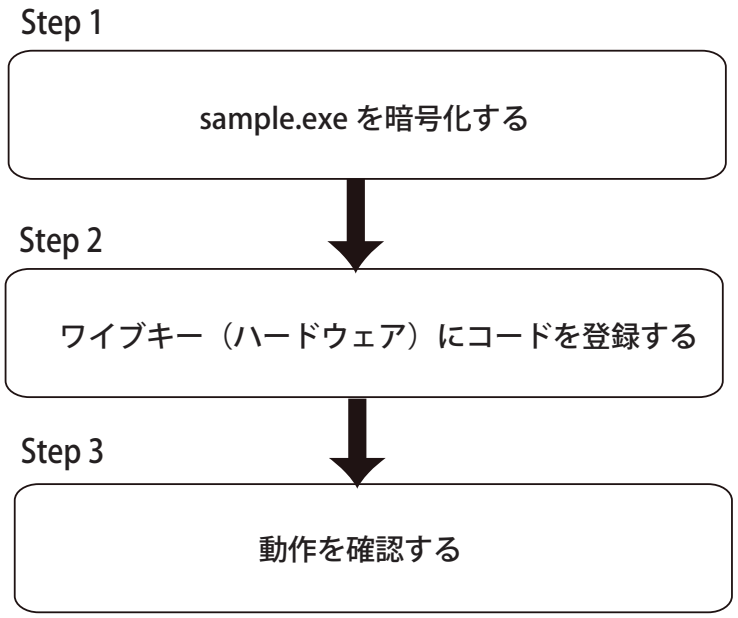
ワイブキーによる実行形式プログラムのプロテクト方法は次の３通りの方法があります。
１. 自動暗号化ツール「AxProtector」を使ってプログラム全体を暗号化する方法
２. WUPIファンクションやワイブキーAPIファンクションをソースコードに組み込む方法
３. 自動暗号化ツール「AxProtector」とWUPI/ワイブキーAPIファンクションを同時に使用する方法

セキュリティ的には３の自動暗号化ツール「AxProtector」とWUPI/ワイブキーAPIファンクションを同時に使用する方法が一番強力ですが、１の自動暗号化ツール「AxProtector」で暗号化するだけでも強力なプロテクトを実現することができます。また、WUPIファンクションやワイブキーAPIファンクションを使って、モジュール単位のプロテクトなどきめの細かいプロテクトを実現することができます。貴社のセキュリティニーズに応じて使い分けてください。

ここでは、ワイブキーCDの中にあるsample.exeを使って、自動暗号化ツール「AxProtector」の基本的な使用方法をご説明いたします。sample.exeは、ワイブキーCDのTools/AxProtectorフォルダの中に格納されています。PCのローカルディスクにコピーしてお使いください。また、弊社下記サイトからもダウンロードできます。

http://www.suncarla.co.jp/wibukey/manual/v711/tool.zip

作業の流れとして、以下のようになります。



3-2. sample.exe を暗号化する

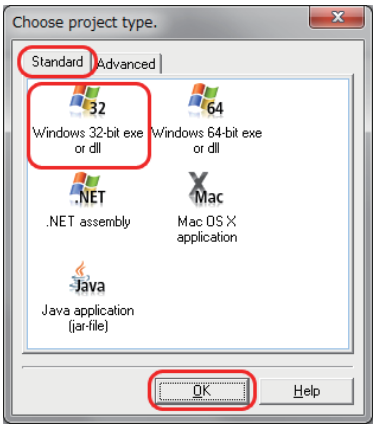
sample.exeを暗号化（プロテクト処理）します。プログラムを暗号化するには、自動暗号化ツール「AxProtector」を使用します。この「AxProtector」は、プログラムファイル全体を強力に暗号化し、デバッガからの解析を非常に困難にします。ソースコードを変更する必要がないのでとても便利です。

ワイブキーCDのTools¥AxProtectorフォルダからsample.exeを任意のフォルダにコピーします。ここでは、testフォルダを作成し、そこにコピーします。

0. 最初に AxProtector を起動する

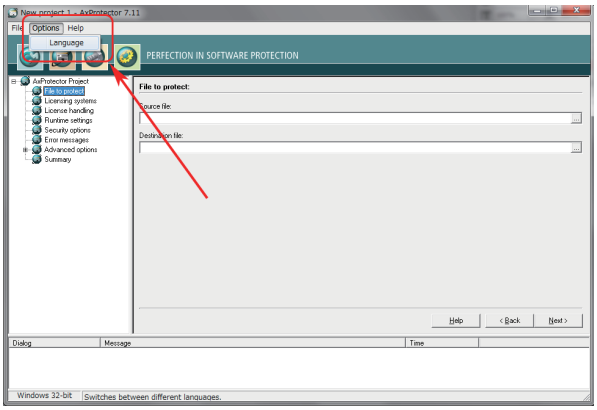
まずはじめに、【スタート】→【すべてのプログラム(P)】→【AxProtector】→【AxProtector】からAxProtectorを起動し,プロテクトタイプの選択画面で「Windows 32-bit exe or dll」を選択します。

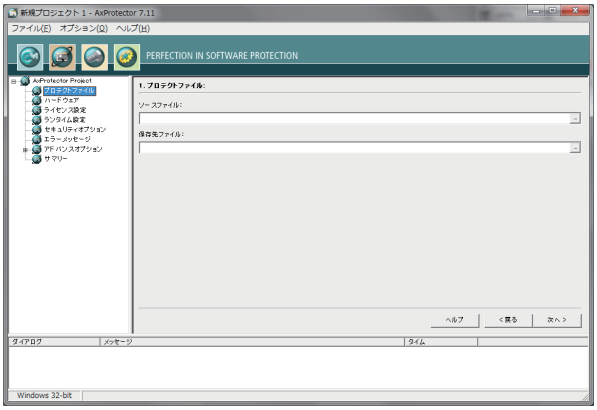
"AxProtectorGui.exe"は、インストール先の¥Program Files¥WIBU-SYSTEMS¥AxProtector¥DevKit¥binフォルダの中にあります。直接ダブルクリックして起動することもできます。



画面を日本語モードにする

[Options]-[Language]を選択し、「言語選択」画面で"Japanese"を選択し、OKをクリックします。AxProtectorが日本語モードに変換されます。

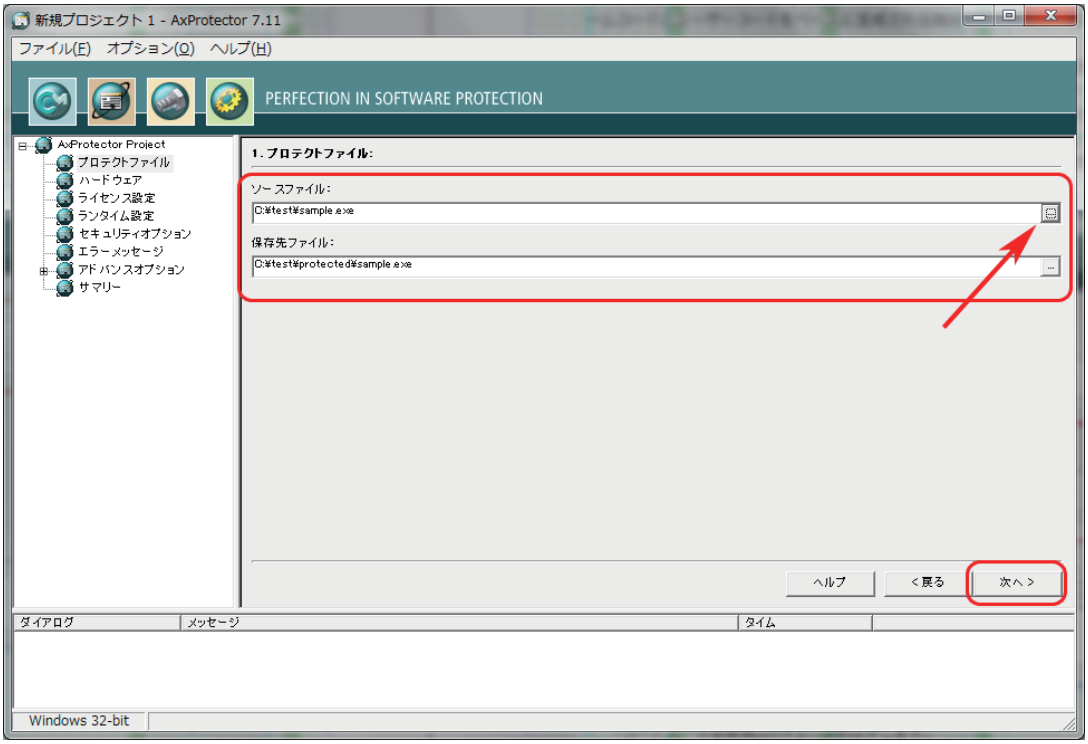




1. プロテクトファイル：

「1. プロテクトファイル」画面で、「ソースファイル」に、オリジナルファイル名を入力（右側参照ボタンより）し、「保存先ファイル」にプロテクト処理後に作成されるファイル名を入力（右側参照ボタンより）します。ここでは、「ソースファイル」に「test¥sample.exe」を設定し、「保存先ファイル」に「test¥protected¥sample.exe」を設定します。ファイル名の設定が終了したら［次へ］ボタンをクリックして先に進めます。

［NOTE］
オリジナルファイルを上書きするのを防ぐために、必ず保存先ファイル名はソースファイル名と別名にするか、または異なるパス名（異なるフォルダ）にしてください。ソースファイルを指定すると、同一フォルダにprotectedフォルダが自動的に作成されます。

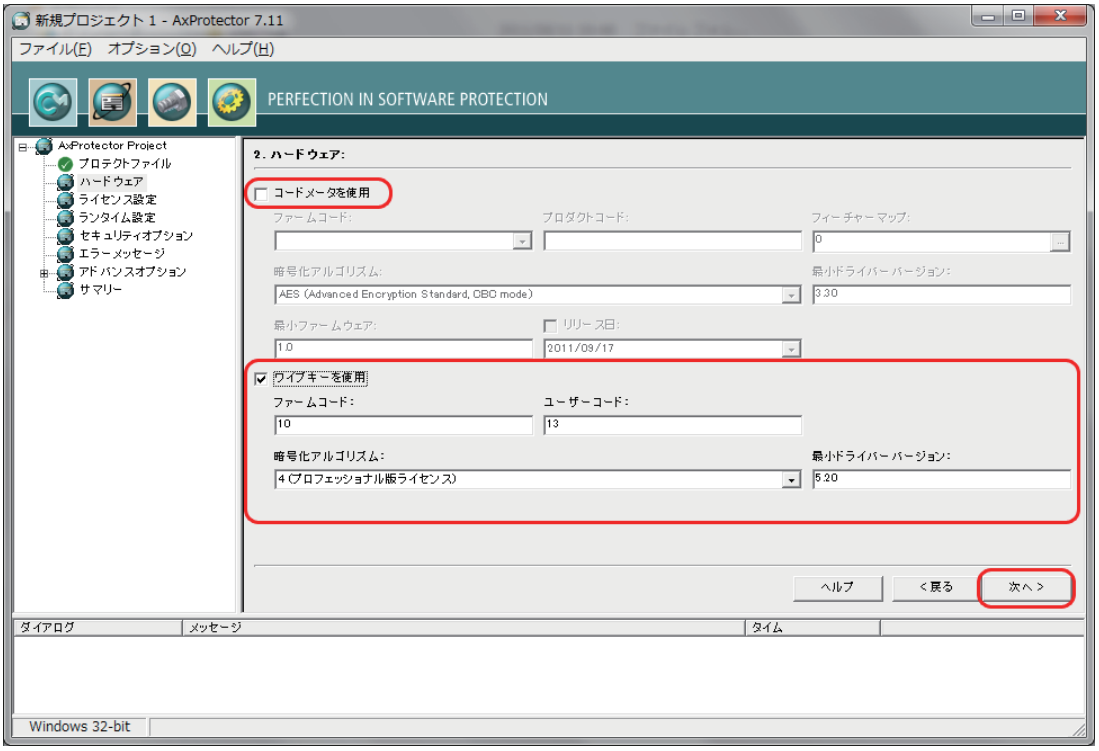


設定が終了したら、「次へ」ボタンをクリックして次に進みます。

2. ハードウェア：

「2. ハードウェア」画面で、暗号化されたプログラムが起動するときに必要とするハードウェアキーの指定を行います。ここで入力したコード（ファームコード+ユーザーコード)と同じコードを持つワイブキーがPCに装着されていないと、暗号化されたプログラムは起動しません。プログラムは、入力されたファームコードとユーザーコードをベースに生成される独自のランダム値で暗号化されます。

自動暗号化ツールAxProtectorは、ワイブキーとコードメータで共通に使用することができます。ここではワイブキーだけを使用し、プロフェッショナル版の場合は、ファームコード=10（評価用コード）、ユーザーコード=13 を設定します。スケルトン版の場合は、ファームコード=250010（評価用コード）、ユーザーコード=13を設定します。また、「暗号化アルゴリズム」には、プロフェッショナル版の場合は「4 (プロフェッショナル版ライセンス)」を、スケルトン版の場合は「5 (スケルトン版ライセンス)」を選択します。



1. 「コードメータを使用」のチェックをはずします。
2. 「ワイブキーを使用」にチェックを入れます。
3. ファームコード欄に10 を入力します。スケルトン版(WKUSB/R)の場合は、250010を入力します。
4. ユーザーコード欄に13 を入力します。（実際には、0-16777215の整数値が入力可能です）
5. 暗号化アルゴリズム欄では、プロフェッショナル版（WKUSB、WK25P、WK25ST、WKCARD)の場合は「4 (プロフェッショナル版ライセンス)」を指定し、スケルトン版（WKUSB/R）の場合は「5 (スケルトン版ライセンス)」を指定します。
6. 「最低限必要なドライバーバージョン」を指定します。ここでは 5.20 を指定します。（デフォルトの状態）

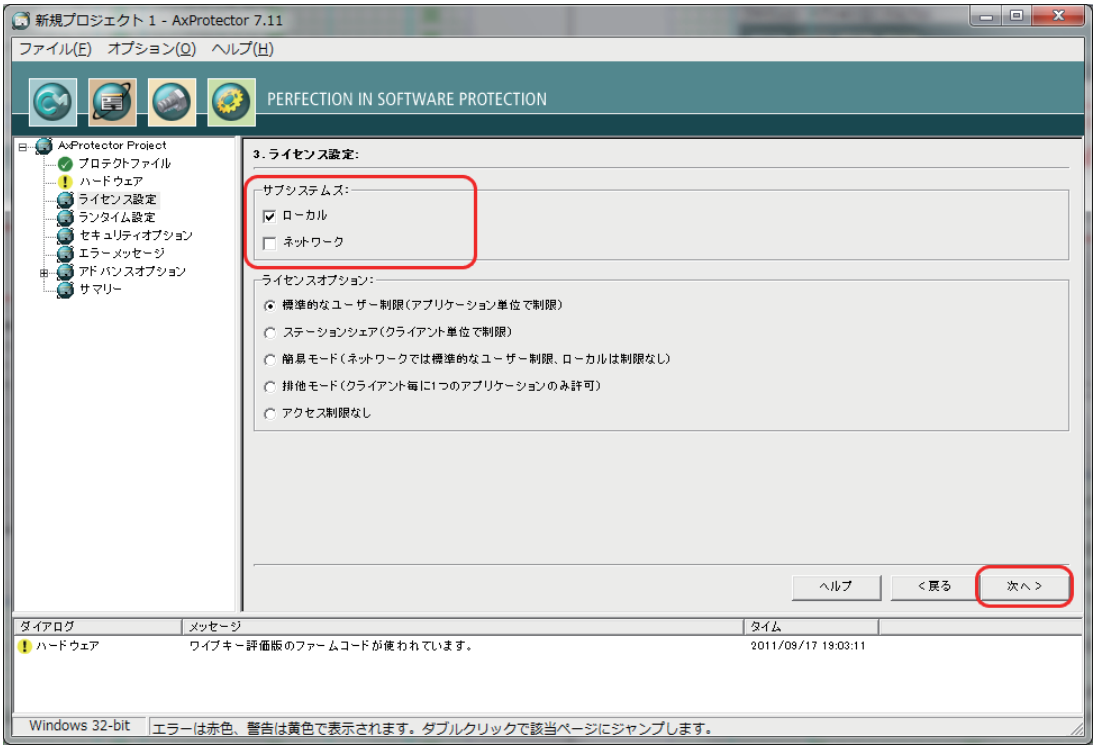
設定が終了したら、「次へ」ボタンをクリックして次に進みます。

3. ライセンス設定

「3. ライセンス設定」画面でライセンス体系を決めます。暗号化されたプログラムが、ローカルPC上のワイブキーだけを検索するか、ネットワーク上（LAN上）のワイブキーを検索するかの設定をします。ここでは、ローカルPC上のワイブキーを検索しますので、"サブシステムズ"の"ローカル"にチェックを入れます。（デフォルトの状態）

ここで、"ネットワーク"にもチェックを入れると、暗号化されたプログラムがローカルPC上でワイブキーを見つけられなかった場合、ネットワーク上（LAN上）のワイブキーサーバーにアクセスします。そこで指定したワイブキーが見つかると暗号化されたプログラムは起動します。"ネットワーク"オプションは、クライアントのフローティングライセンス管理をするときに役に立ちます。ただし、このネットワーク機能はプロフェッショナル版だけの機能で、スケルトン版(WKUSB/R)にはありません。

"ライセンスオプション"は、「標準的なユーザー制限（アプリケーション単位で制限）」（デフォルトの状態）を選択します。



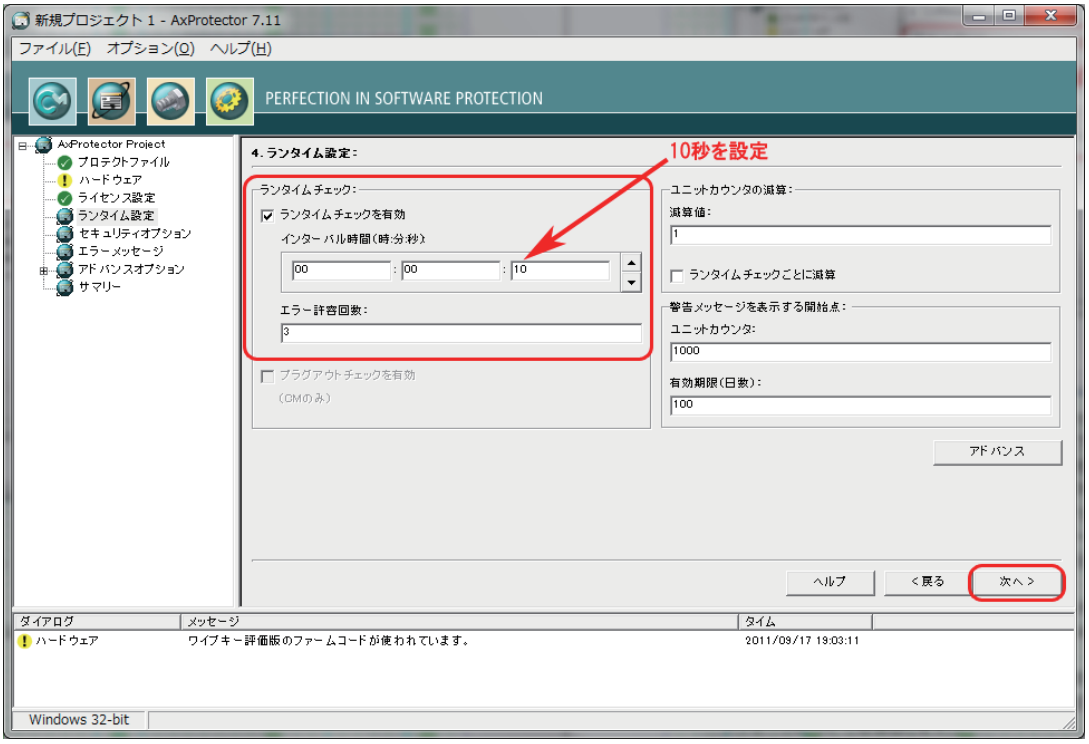
「次へ」ボタンをクリックして先に進みます。

[NOTE]
ライセンスオプションの詳細につきましては、「Chapter 5 自動暗号化ツール AxProtectorについて / 5-4. AxProtectorの各入力画面の説明 / 3. ライセンス設定」をご参照ください。

4. ランタイム設定

「4. ランタイム設定」画面で、一定時間ごとにワイブキーをチェックするランタイムチェックを設定します。このランタイムチェック機能を使うことにより、プロテクトされたプログラムが動作するには必ずワイブキーをPCに装着し続ける必要があり、１個のワイブキーを使ってプログラムを同時に複数のPC上で使用するライセンス違反を防ぐことができます。

設定できるインターバル時間は、時、分、秒単位で設定可能です。デフォルトでは30秒ですが、貴社のセキュリティポリシーに応じてインターバル時間を調整してください。ここでは、テスト的に10秒を設定します。"ランタイムチェックを有効"にチェックを入れ、秒欄（最右部）に10を入力します。



"エラー許容回数"は、ランタイムチェック時に、何らかの原因でチェックエラーになっても、ここで指定した回数だけはアプリケーションの続行を許可するオプション機能です。この機能は、特にプリンターポート版WK25PやCOMポート版WK25STを使用する場合に役に立ちます。たとえば、印刷出力などでプリンターポートを占有していたり、データ通信などでCOMポートを占有している最中にワイブキーチェックが入ると、ポートが物理的にブロックされ、チェックエラーになります。その際、ただちにアプリケーションを終了させずに、指定した回数だけチェックエラーを無視してアプリケーションを続行させる機能です。ワイブキーの先に、出力デバイスを接続する場合に有効なオプション機能です。

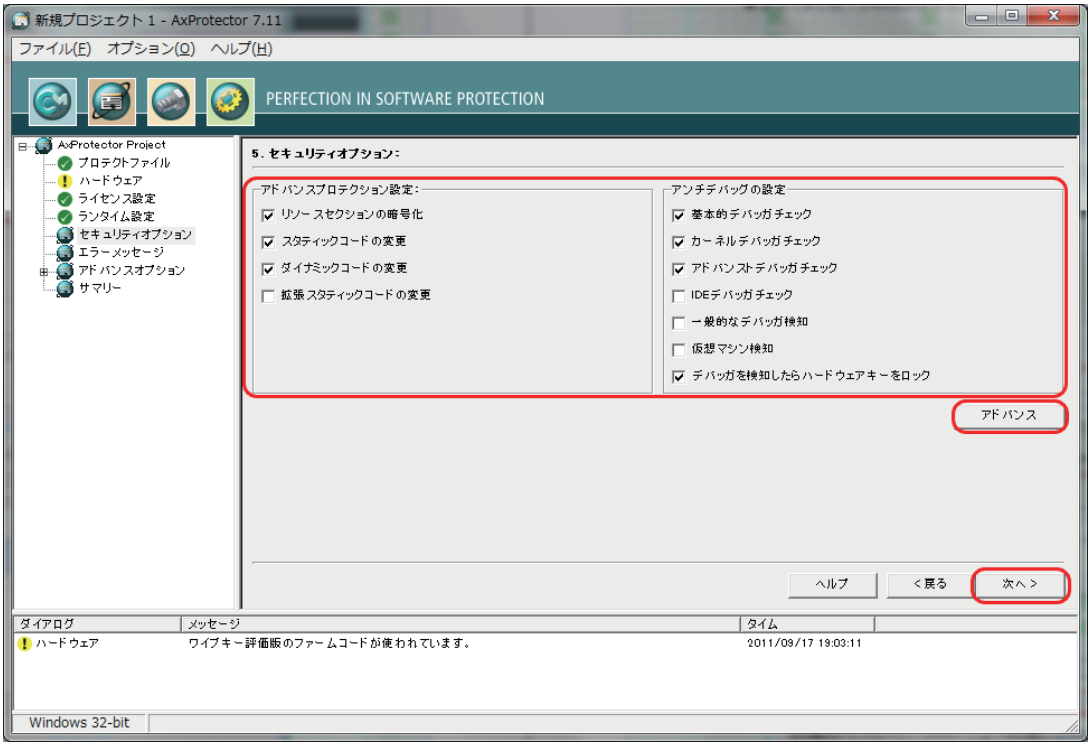
USB版（WKUSB、WKUSB/R）の場合は、ワイブキーの先に別のデバイスを接続することがないため、ポートを占有されることはありません。ここでは、念のためデフォルトの3を設定します。

また、この"ランタイム設定"で、プログラムの起動回数を制限するユニットカウンター(Unit Counter)の減算値や警告メッセージを表示するタイミングを設定することもできます。（ユニットカウンターや警告メッセージについては後述します。）

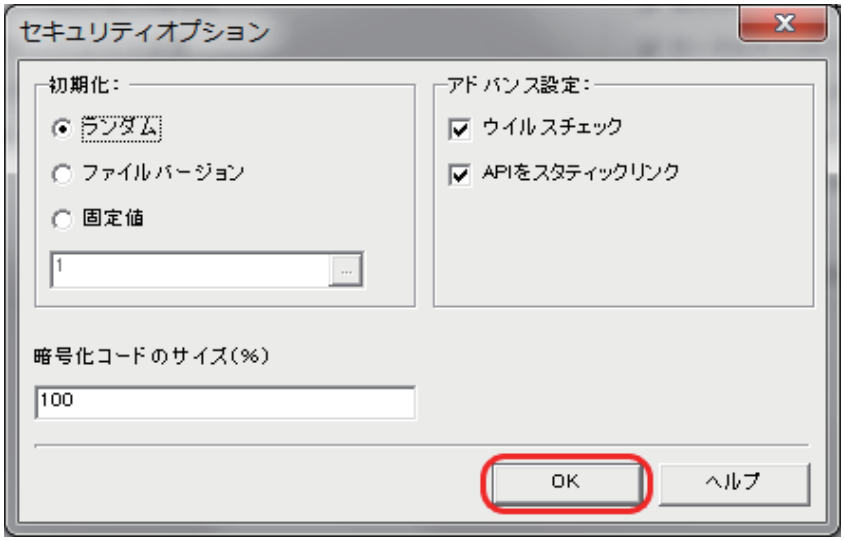
設定が終了したら、「次へ」ボタンをクリックして次に進みます。

5. セキュリティオプション

「5. セキュリティオプション」画面で、セキュリティ強度の設定を行います。暗号化の方法や、デバッガ解析に対するセキュリティポリシーを決めます。AxProtectorは、デフォルトの設定で十分強力な暗号化を実現できるため、ここではデフォルトの状態にします。



また、中央部右側の"アドバンス"ボタンをクリックすると、ランダムデータ生成やウイルスチェック機能追加、暗号化するサイズ（%指定）などのアドバンス機能を追加できます。ここでは、デフォルトのままOKボタンをクリックし、アドバンスト設定"セキュリティオプション"画面を閉じます。



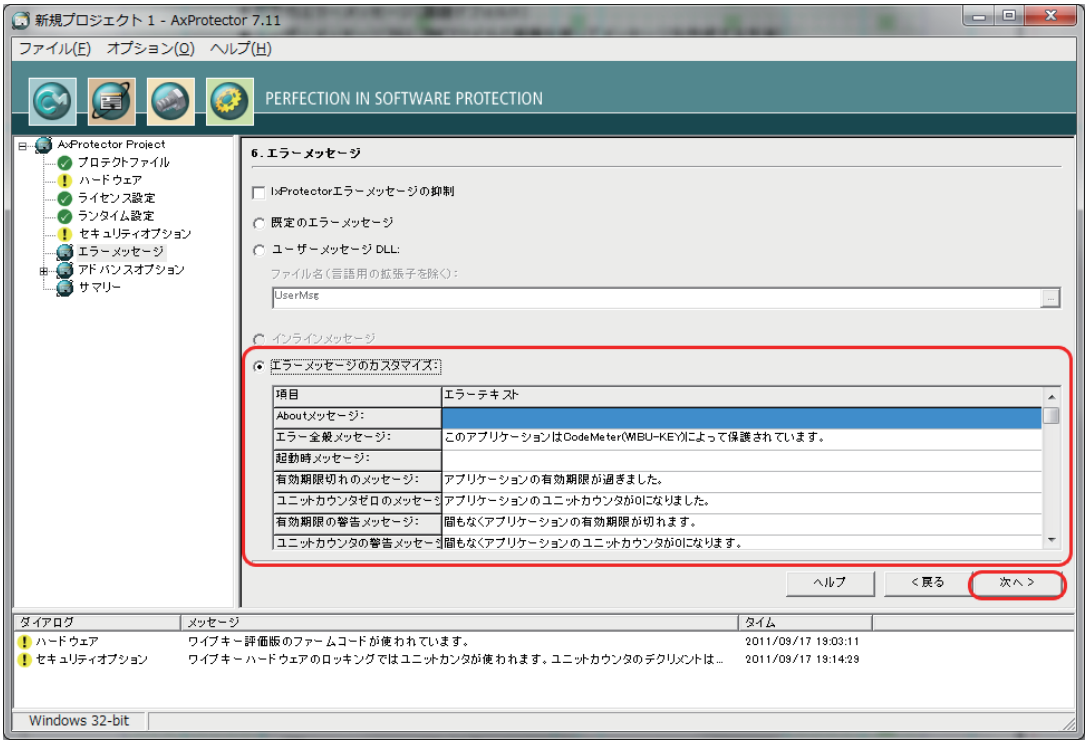
「5. セキュリティオプション」画面に戻り、「次へ」ボタンをクリックして次に進みます。

6. エラーメッセージ

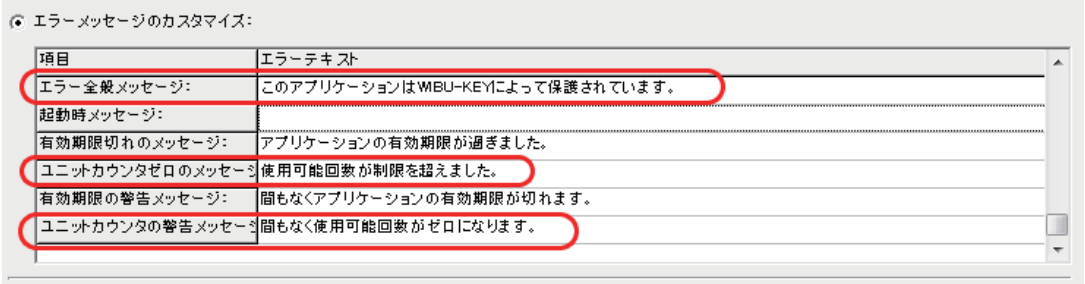
「6. エラーメッセージ」画面で、エラーメッセージを作成します。エラーメッセージの作成には 4 通りの方法があります。

- **既定のエラーメッセージ（英語デフォルト）**
- **ユーザーメッセージ DLL（INIファイルと画像を使ってメッセージを作成する方法）**
- **インラインメッセージ（.NETアセンブリ用）**
- **エラーメッセージのカスタマイズ（フォームからメッセージを直接入力する方法）**

ここでは、「エラーメッセージのカスタマイズ」にチェックを入れ、「エラー全般メッセージ：」項目のエラーテキスト欄をクリックし、"CodeMeter(WIBU-KEY)"部分を"WIBU-KEY"に修正します。このメッセージは、ワイブキーが見つからなかった時に表示されるメッセージです。



また、「ユニットカウンタゼロのメッセージ」を「使用可能回数が制限を超えました。」に、「ユニットカウンタの警告メッセージ」を「間もなく使用可能回数がゼロになります。」と修正し、「次へ」ボタンをクリックして次に進みます。



7. アドバンスオプション

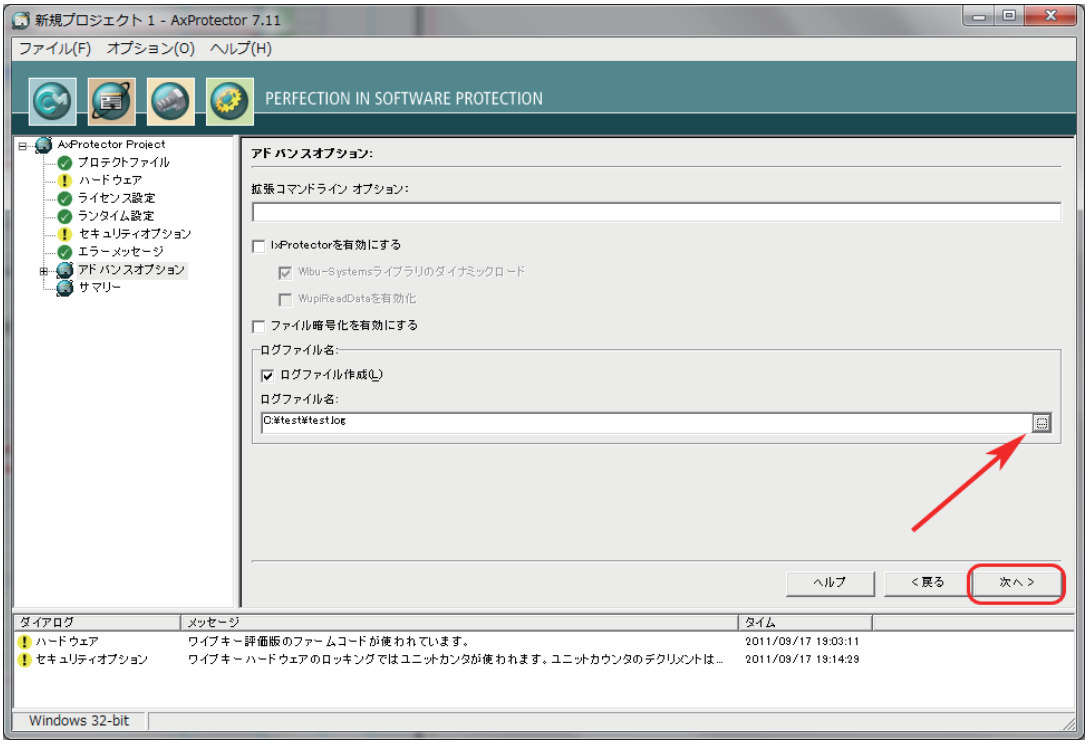
アドバンスオプション画面で、暗号化の拡張機能を追加します。

「拡張コマンドラインオプション」では、AxProtector画面で用意されていない新しい拡張機能を追加する場合に使用します。

「IxProtectorを有効にする」は、IxProtectorセキュリティ機能を使用する場合にチェックを入れます。IxProtectorは、メモリー上で展開されるコードを常に暗号化しておき、必要な時に必要なモジュールを復号し、実行したあとは再び暗号化しておくという、メモリー上での「オンデマンド復号」を実現する機能です。AxProtectorで暗号化されたコードが、メモリー上でも常に暗号化されているため、クラッキングに対して非常に強力なセキュリティを実現できます。IxProtectorを使用するには、ソースコードにWUPI (Wibu Universal Protection Interface)ファンクションを組み込む必要があります。ここでは、IxProtectorは使用しませんのでチェックをはずしておきます。

「ファイル暗号化を有効にする」は、アプリケーションから読み書きするデータファイルを暗号化する機能です。ここでは使用しませんのでチェックをはずしておきます。

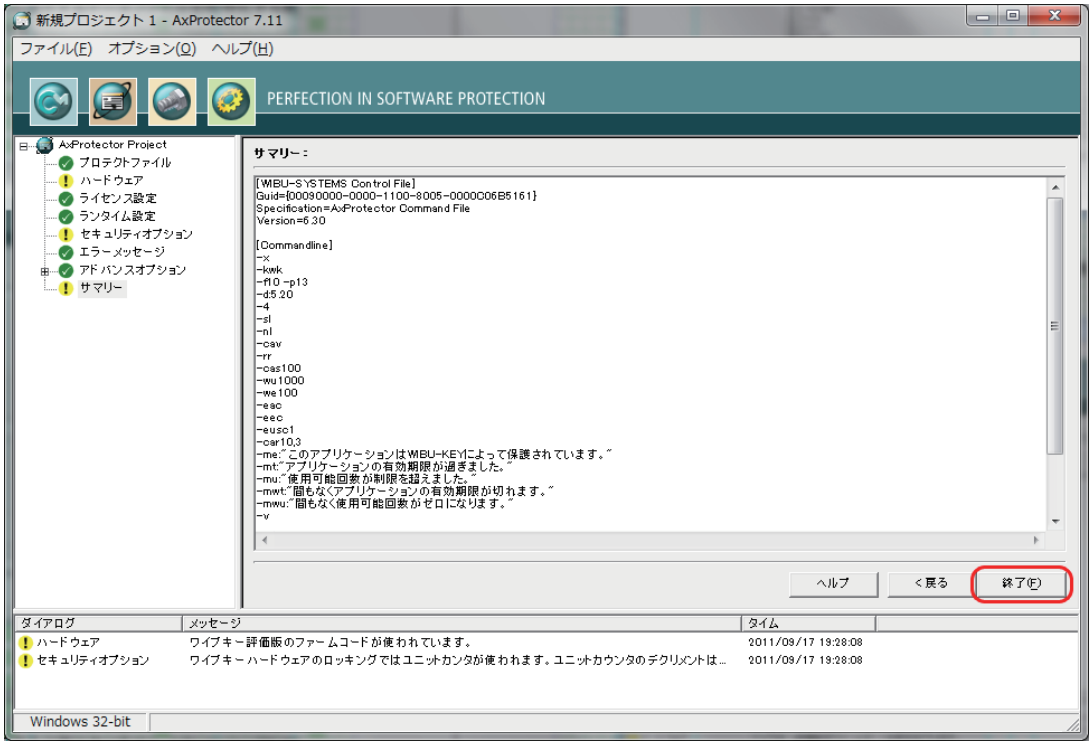
「ログファイル作成」は、AxProtectorで暗号化処理を行った結果の内容が反映されます。ログファイルを作成する場合は、ログファイルを指定してチェックを入れます。ここでは、¥testフォルダにtest.logを入力してログファイルを作成します。



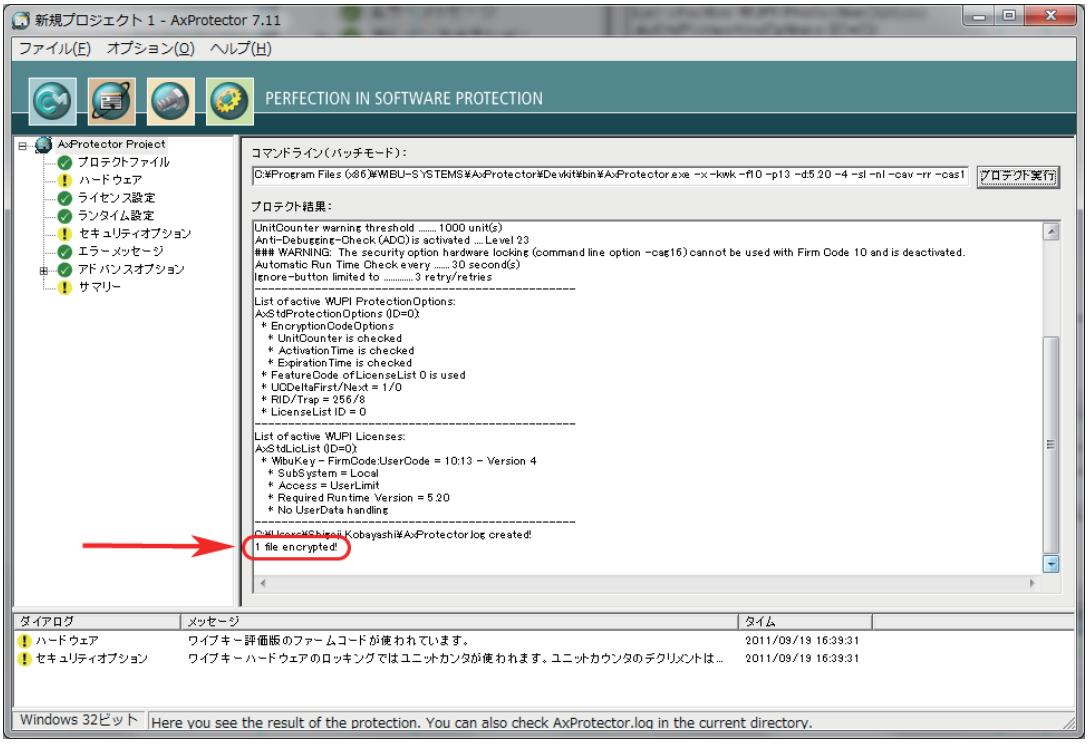
[NOTE]
C#やVB.NET等で作成したMicrosoft .NET Framework対応プログラム（マネージコード）の場合は、IxProtector/WUPIを使用しなくても、AxProtector(.NETアセンブリ)で暗号化することで自動的に「オンデマンド復号」機能が組み込まれます。ソースコードを修正する必要がありません。IxProtector/WUPIをソースコードに組み込む必要があるのは、VC++やVB6などのネイティブコード（アンマネージコード）の場合です。

8. プロテクト内容を確認する（サマリー :）

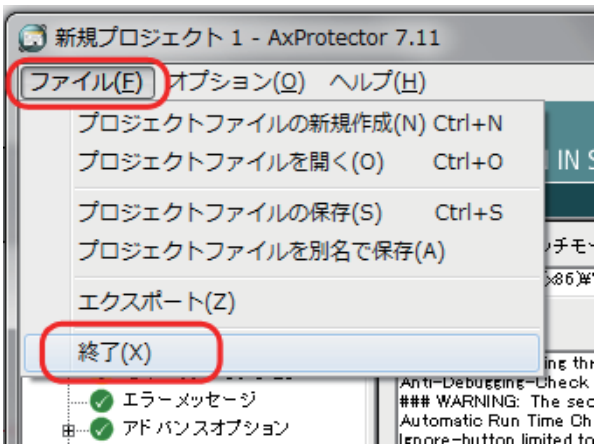
サマリー画面に、いままでに設定したプロテクト内容が表示されます。



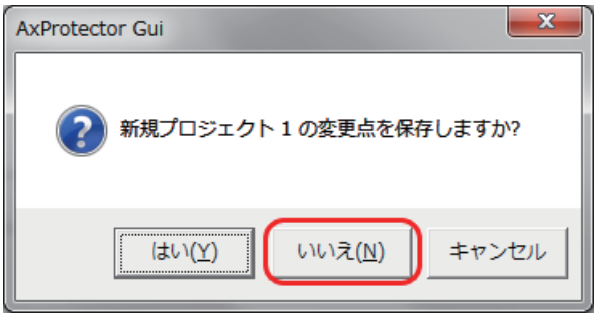
「終了」ボタンをクリックし、暗号化処理を開始します。
暗号化処理が正常に行われると、以下の画面に"1 file encrypted!"の表示がされます。
（右のスクロールバーで最終行までスクロールしてください。）



これで、sample.exeの暗号化処理は終了です。AxProtectorを終了し、プロテクトしたsample.exeの動作を確認します。メニューバーの「ファイル(F)」-「終了(X)」、または画面右上の × ボタンで終了してください。



最後に、AxProtector を終了する時に右のダイアログBoxが表示されます。いままでのプロテクト内容をファイルに保存するかどうかの選択です。ここでは、「いいえ(N)」をクリックして終了します。



エクスプローラ等を使って、test¥protectedフォルダに、暗号化されたsample.exeが作成されていることを確認してください。

3-3. ワイブキー (ハードウェア) にコードを登録する

次に、ワイブキー（ハードウェア）にファームコード(Firm Code)とユーザーコード(User Code)を登録します。ワイブキー（ハードウェア）にコードを登録するには、貴社のFSB(Firm Security Box)とコード登録ツール「Wklist32.exe」を使用します。評価版の場合はFSBは不要です。FSBが無くても評価用コード(Firm Code = 10, User Code = 0 ~ 16777215)を登録することができます。スケルトン版は、評価用コード(Firm Code=250010, User Code = 0 ~ 16777215)を登録できます。

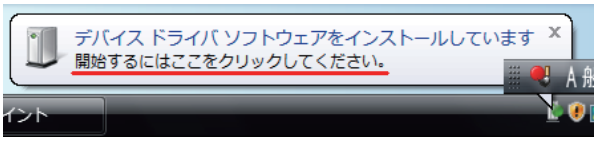
ここでは、プロフェッショナル版である、USBポート版ワイブキーWKUSBにコードを登録します。その他のLPTポート版ワイブキーWK25P、COMポート版ワイブキーWK25ST、スケルトン版WKUSB/Rの場合も操作は同じです。

1. FSB (Firm Security Box) を PC に装着する

まず、FSBをUSBポートに装着します。評価版の場合は、この作業は不要ですので次の「2. ワイブキーWKUSBをPCに装着する」に進んでください。FSBがLPTタイプ（プリンターポートタイプ）の場合は、PCのLPTポート（プリンターポート）に装着します。FSBの形状は、一般のワイブキーと同じですが、表面に貴社固有のFirmCode（プロフェッショナル版は整数値4桁、スケルトン版は整数値6桁）が表示されていますので、一般のワイブキーと混同しないようにご注意ください。

※　FSB は貴社固有のものです。プロテクト作業終了後は厳重に管理してください。

初めてワイブキーUSBタイプ（FSB、WKUSB、WKUSB/R）を装着すると、新しいハードウェアとして認識され、デバイスドライバのインストール作業が行なわれます。



2. ワイブキー WKUSB を PC に装着する

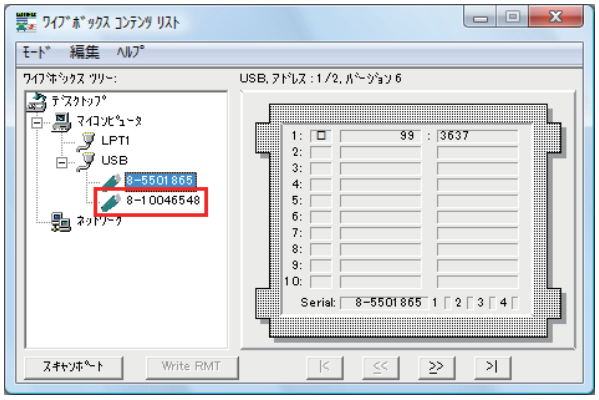
次に、ワイブキーWKUSBをFSBとは別のUSBポートに装着します。PC本体のUSBポートが足りない場合は、USBハブへ装着することも可能です。

[NOTE]
FSBとワイブキーが共にUSB版の場合は、USBポートを2つ使用することになります。FSBとワイブキーが共にプリンタポート版(WK25P)の場合は、2つを連結してPCのLPTポート（パラレルポート）に装着します。COMポート版ワイブキーWK25STの場合は、WK25STをCOMポートに装着し、FSBをUSBポート（FSBがUSBタイプの場合）またはLPTポート（FSBがLPTポートの場合）に装着します。

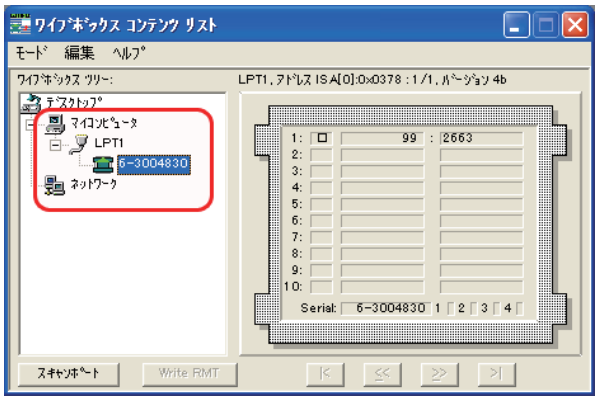
3. コード登録ツール "WKLIST32.EXE" を起動する

FSBとワイブキーWKUSBを装着した状態で、ワイブキーコード登録ツール"WKLIST32.EXE"を起動します。【すべてのプログラム(P)】→【WibuKey】→【Development-Kit】→【WkList32 - WibuKey Programming】から起動します。なお、"WKLIST32.EXE" は ¥Program Files¥WibuKey¥DevKit¥Binフォルダの中にありますので、直接ダブルクリックして"WKLIST32.EXE"を起動することも可能です。

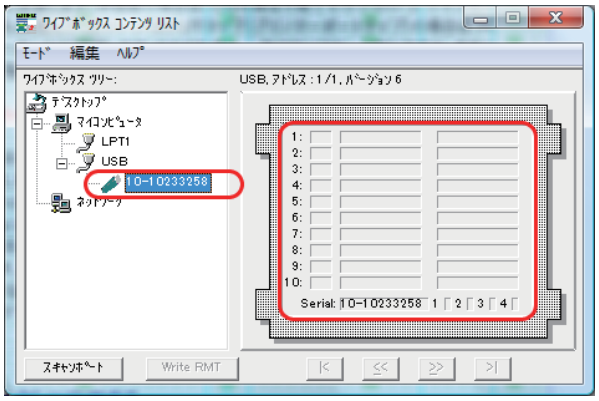
画面ワイブボックスツリー部に、現在装着されているワイブキーのシリアル番号が表示され、画面右部には、選択されたワイブキーの内容（エントリ1～エントリ10）が表示されます。FSBの場合は、エントリ１の左側に"99"、右側にファームコード(FirmCode)が登録されています。



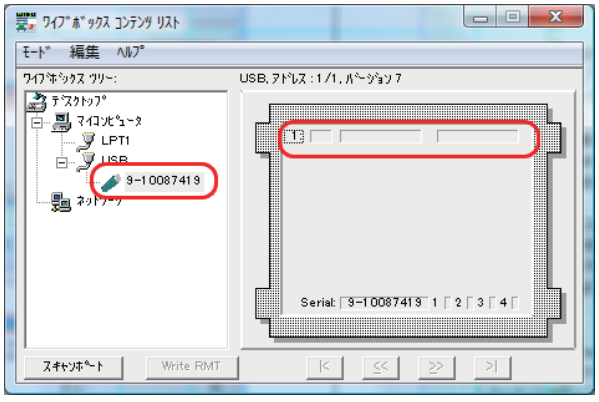
FSBがLPTタイプ（プリンタポートタイプ）の場合は、ツリー部のLPTにFSBのシリアル番号が表示されます。



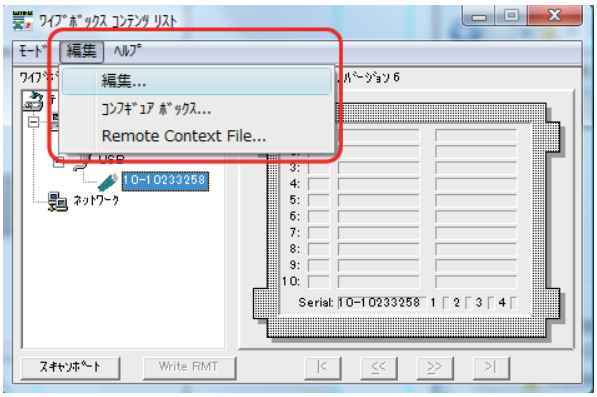
コード登録をするワイブキーWKUSBのシリアル番号を選択し、右側にそのワイブキー内容を表示させます。（出荷時点のワイブキーには、何も情報が書かれていません。）



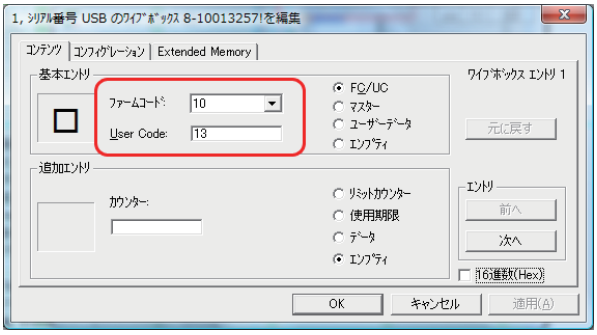
スケルトン版WKUSB/Rの場合も同じように右側にワイブキー内容が表示されます。スケルトン版WKUSB/Rの場合は、エントリーが１つだけ表示されます。



「編集」メニューから「編集…」を選択します。

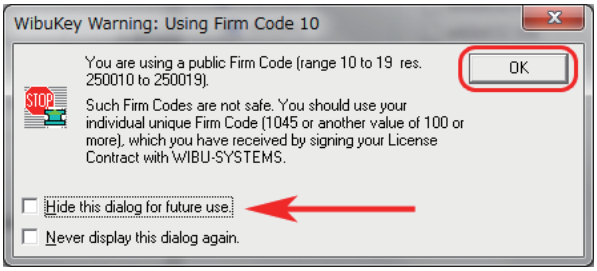


編集画面が開きます。
FC/UCを選択し、「ファームコード」欄の右側の▼ボタンをクリックし、ドロップダウンリストから貴社のファームコードを選択します。その下の「UserCode」欄には、あらかじめ貴社で設定した任意の整数値を入力します。User Codeは、0-16777215の範囲の整数値が使用できます。



ここでは、ファームコードを10、User Codeを13に設定してワイブキーWKUSBに登録することにします。入力後、下のOKボタンをクリックすると、WKUSBにコードが登録されます。

はじめて、ファームコード=10または250010を登録すると、右の警告メッセージが表示されます。これは、登録しようとするファームコードが評価用コードのためセキュリティが脆弱になることを警告するメッセージです。今回は問題ありませんので、「OK」ボタンをクリックして閉じます。また、次回以降表示させたくない場合は、"Hide this dialog for future use"にチェックを入れます。



[注意]
貴社専用のライセンスファイル"WkFirm.wbc"が正しくインポートされていないと、「ファームコード」欄に貴社のファームコードが表示されませんのでご注意ください。(ファームコード=10のみ表示される)
ライセンスファイル"WkFirm.wbc"のインポートにつきましては、「Chapter 2 ワイブキー開発ツールをインストールする / 2-1. ワイブキー開発ツール(SDK)をインストールする / 3. ライセンスCDから貴社固有のライセンスファイルWkFirm.wbcをコピーする」をご参照ください。

[NOTE]
ユーザーコード(User Code)は、商品ごと、ユーザーごとに数値を使い分けると便利です。
ここで登録した「ファームコード/ユーザーコード」と、プログラムを暗号化する際に組み込まれる「ファームコード/ユーザーコード」が一致した時に、暗号化されたプログラムが起動します。
なお、ユーザーコードの15728640~16777215の範囲は、ヒュージライセンスマネジメント(HLM)のために通常は使用しません。(ヒュージライセンスマネジメントについては、後述の「Chapter 7 ネットワーク機能について」を参照してください。)

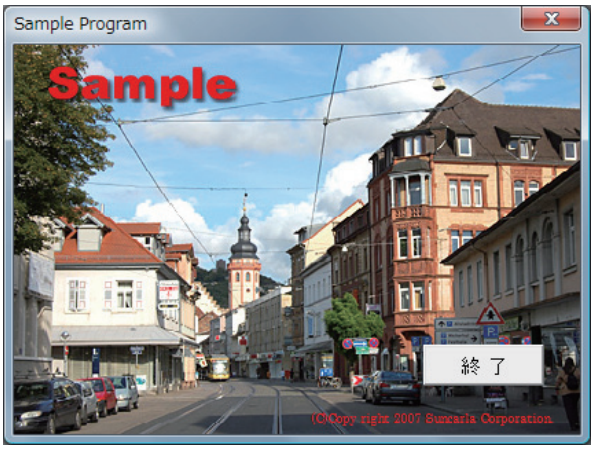
［NOTE］
プロフェッショナル版「評価版」の場合は、FirmCode=10 が登録できます。スケルトン版「評価版」は、Firm Code = 250010が登録できます。（User Codeは0-16777215の範囲で登録可能。）

[NOTE]
一度登録したコードは、何度でも修正したり消去することができます。「エンプティ」を選択しOKまたは適用ボタンをクリックすると編集中のエントリ内容が消去されます。

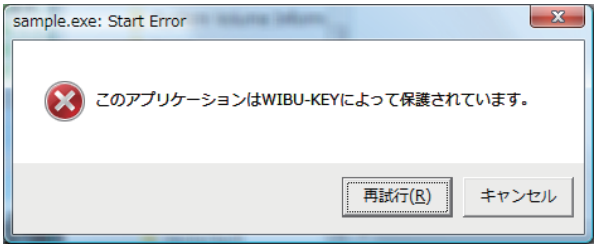
3-4. 暗号化された sample.exe の動作を確認する

暗号化されたsample.exeの動作確認をします。
正しいコードが登録されたワイブキーをPCに装着し、暗号化されたsample.exeが起動することを確認します。次に、ワイブキーをPCから取りはずして暗号化されたsample.exeを起動します。エラーメッセージが表示されsample.exeが起動しないことが確認できます。

正しく起動すると右の画面が表示されます。
（終了ボタンをクリックするとプログラムを終了します。）

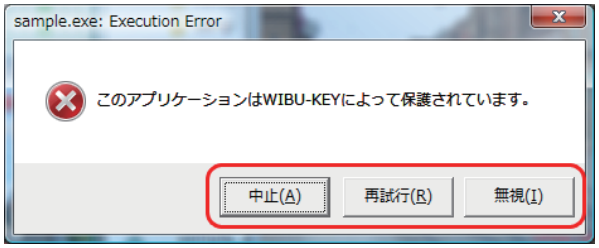


エラーの場合は、右のメッセージが表示されます。
正しいワイブキーが装着されていないとプロテクトしたsample.exeが起動しません。

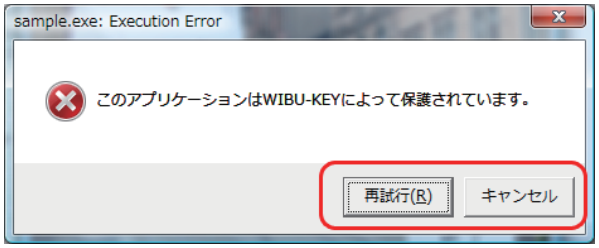


ランタイムチェックを確認する

sample.exeが起動された状態で、ワイブキーをPCから取り外します。ランタイムチェックのインターバル時間を10秒に設定してあるので、10秒後に右のメッセージが表示されます。



ワイブキーをPCに装着して「再試行(R)」ボタンをクリックすると、sample.exeに制御が移ります。「中止(A)」ボタンをクリックすると、sample.exeは正常に終了します。「無視(I)」ボタンをクリックすると、AxProtectorの「4.ランタイム設定」の「エラー許容回数」で設定した回数分だけsample.exeを続行することができます。



「エラー許容回数」で設定した回数を超えると、「無視(I)」ボタンが表示されなくなり、ワイブキーを装着して再試行するか、「キャンセル」ボタンをクリックしてsample.exeを終了させるかの２つの選択だけになります。

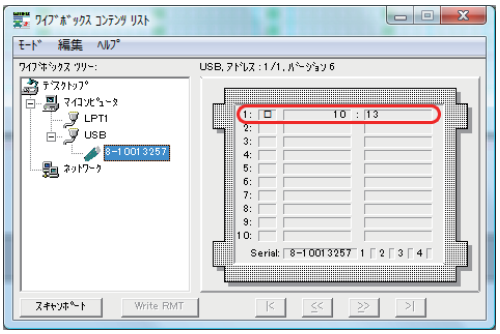
今回は、エラー許容回数を３回（デフォルト）で設定しましたので、「無視(I)」ボタンが３回まで表示されます。貴社のセキュリティポリシーに応じて自由にエラー許容回数を設定してください。

[NOTE]
１個のワイブキーを使って複数のPC上で同時にプログラムを使用するというライセンス違反を防ぐためにも、このランタイムチェック機能のご使用をお勧めいたします。ユーザーは、プログラムを使用するために、常にワイブキーをPCに装着することが強要されます。このランタイムチェックを使用しない場合は、AxProtectorの「4.ランタイム設定」画面で、「ランタイムチェックを有効」のチェックをはずします。この場合、ワイブキーチェックはプログラムの起動時だけ行われ、プログラム起動後はチェックが行われなくなります。

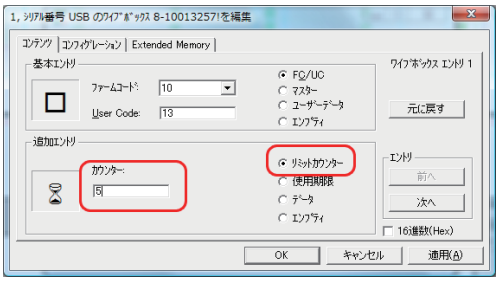
3-5. 起動回数を設定したプロテクトを行う

sample.exeに対し、起動回数（リミットカウンター)を設定したプロテクトを行ってみます。起動回数を設定するには、ワイブキー側の追加エントリーのリミットカウンターに指定回数を登録します。

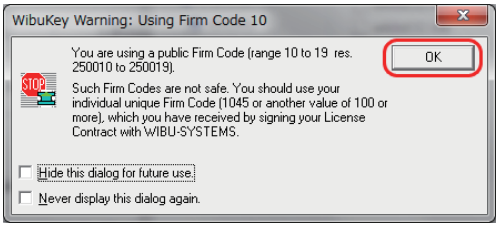
1．先ほど、作成したWKUSBをPCに装着し、コード設定ツール"Wklist32.exe"を起動します。赤枠部分をクリックして、エントリ編集画面を開きます。



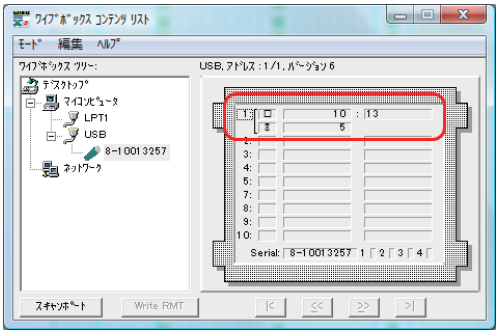
2．追加エントリ部分で、リミットカウンターにチェックを入れ、カウンター欄に指定起動回数を入力します。ここで指定した回数がプログラムの起動回数になります。ここでは、5を入力し、下のOKボタンをクリックします。



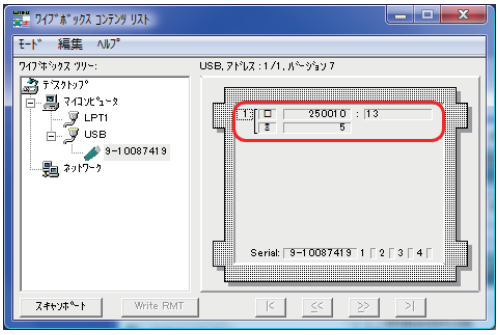
3．警告メッセージが表示されますが、そのままOKボタンをクリックして進めます。このメッセージは、ファームコード = 10または250010でプロテクト作業を行う場合に、セキュリティ警告として表示されるメッセージです。実際の正式ファームコードで作業する場合は表示されません。



4．エントリ 1 のファームコード = 10、ユーザーコード = 13の下にリミットカウンター値=5が登録されました。

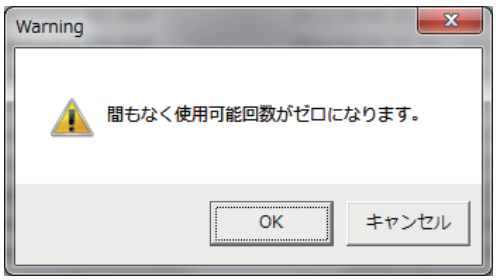


スケルトン版(WKUSB/R)の場合も、エントリ 1 のファームコード=250010、ユーザーコード=13の下に、追加エントリとしてリミットカウンター値=5が登録されます。



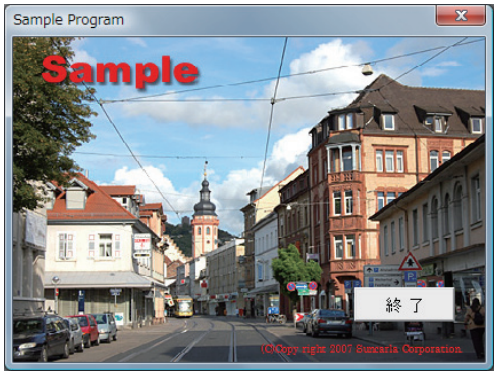
5. 先ほど暗号化したsample.exeを起動してみます。ワイブキーWKUSBはPCに装着したままにしてください。

途中、警告メッセージが表示されますが、これはリミットカウンターの残り回数が少ない場合に表示される警告メッセージです。

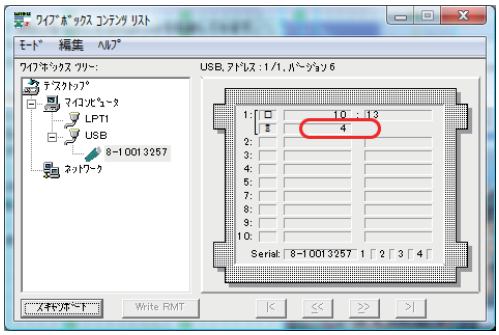


OKボタンをクリックすると、sample.exeが正常に起動します。

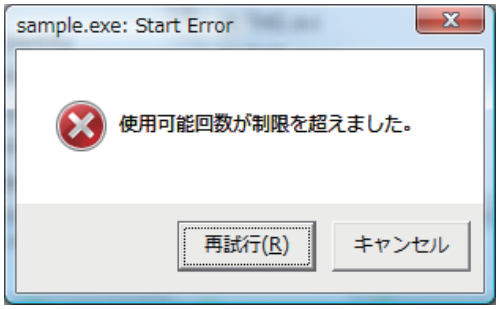
終了ボタンをクリックしてsample.exeを閉じます。



６. 再び、コード設定ツール"Wklist32.exe"を起動します。リミットカウンター値が１つ減って、４になっているのがわかります。



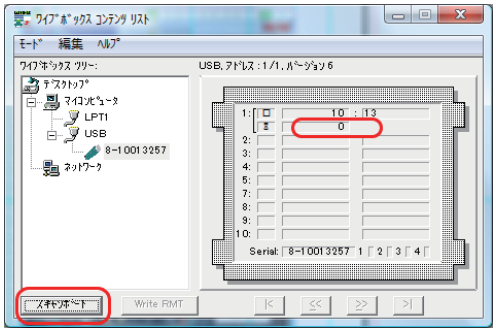
７. sample.exeをあと４回続けて起動してください。４回目に右のエラーメッセージが表示され、sample.exeが起動できなくなります。これは、許可された起動回数を超え、プログラムが起動できなくなったためです。



８. コード設定ツール"Wklist32.exe"画面で、"スキャンポート"をクリックします。最新のリミットカウンター値が表示され、0になっているのがわかります。

リミットカウンター値が0になると、プログラムは起動できなくなります。

これが、ワイブキーの起動回数制限機能（リミットカウンター機能）です。



[NOTE]
リミットカウンター値が0になったワイブキーは、FSBを使って、エントリ編集画面から再度カウンター値を更新することができます。評価用コード(ファームコード = 10または250010)の場合は、FSBがなくても、カウンター値を更新できますが、正式なファームコードでのカウンター値の更新には、必ずそのファームコードが割り当てられたFSBが必要になります。ユーザーが自由に書き換えることができません。

カウンター値は、FSBがあれば貴社にて何度でも書き換えることが可能ですので、貴社のアプリケーションの評価用(使用回数制限版)に使用すると非常に便利です。

また、ユーザー先にあるワイブキーのカウンター値を、ファイルのやりとりだけで更新できるリモートアップデート機能も用意されています。この機能を使えば、ワイブキーを直接送ったり送り返したりする必要がなく、メール添付などの遠隔操作で更新作業が行えます。
(「Chapter 6 リモートアップデート機能について」参照)

エラーメッセージなどの日本語化については、「Chapter 5 自動暗号化ツールAxProtectorについて / 5-4. AxProtectorの各入力画面の説明/ 6. エラーメッセージ」を参照してください。

リミットカウンター機能を使用しない場合は、追加エントリで"エンプティ"にチェックをいれ「OK」または「適用」ボタンをクリックしてリミットカウンターをはずします。

[注意]
ワイブキーを装着した状態で、リミットカウンター値の更新を繰り返しながら、起動回数のテストを繰り返すと、セキュリティ上の理由から、ワイブキーがロックされることがあります。これは、第三者によるワイブキーの解析を防ぐためのハードウェアロックです。ロックされた場合は、PCを一度再起動してから再実行してください。

3-6. プロテクトされたプログラムを起動する場合の注意点

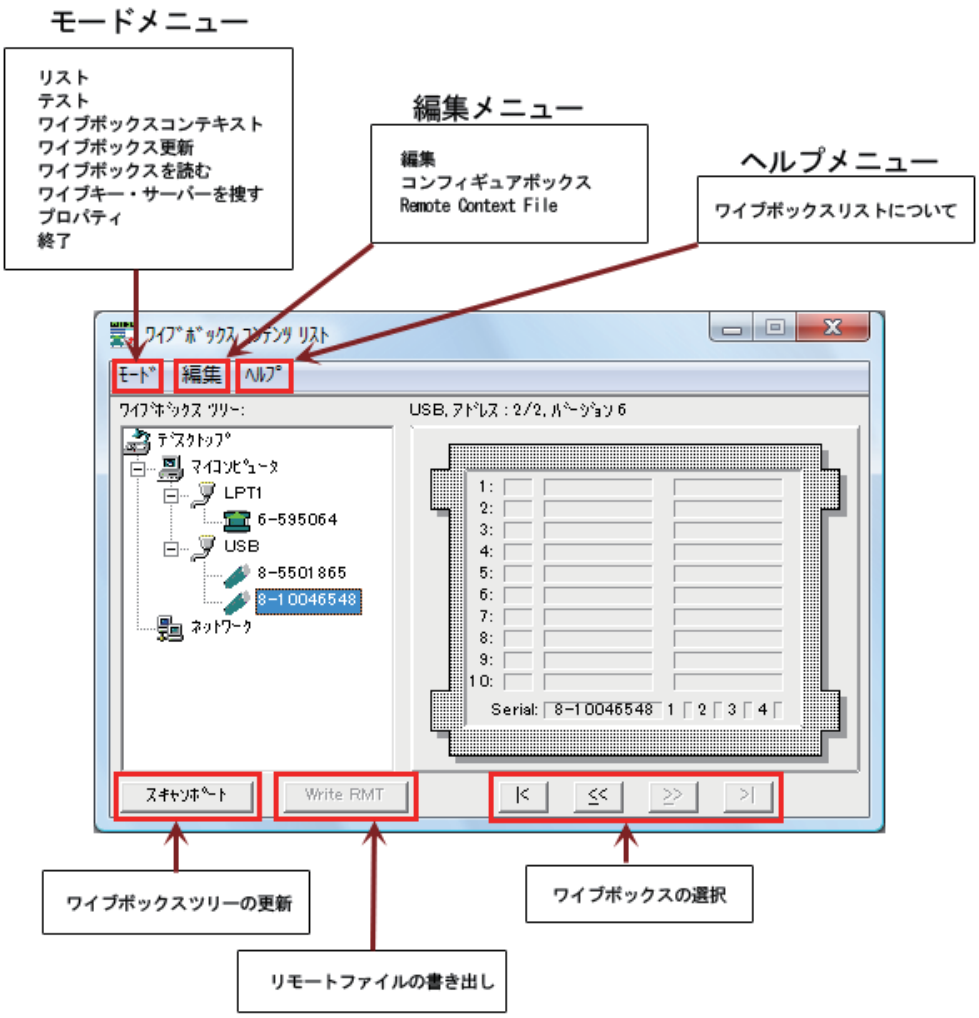
ワイブキーでプロテクトされたプログラムが動作するためには、ワイブキーランタイムキットがPCにインストールされている必要があります。今回プロテクト作業を行ったPCには開発キットをインストールした時点でワイブキーランタイムキットもインストールされたため、特に単独でランタイムキットをインストールする必要がありませんでした。プロテクトされたプログラムを別のPC上で起動するためには、あらかじめワイブキーランタイムキットをインストールする必要があります。ワイブキーランタイムキットのインストール方法につきましては、「Chapter 13 その他 / 13-1. ワイブキーランタイムキットのインストールについて」を参照してください。

Chapter 4

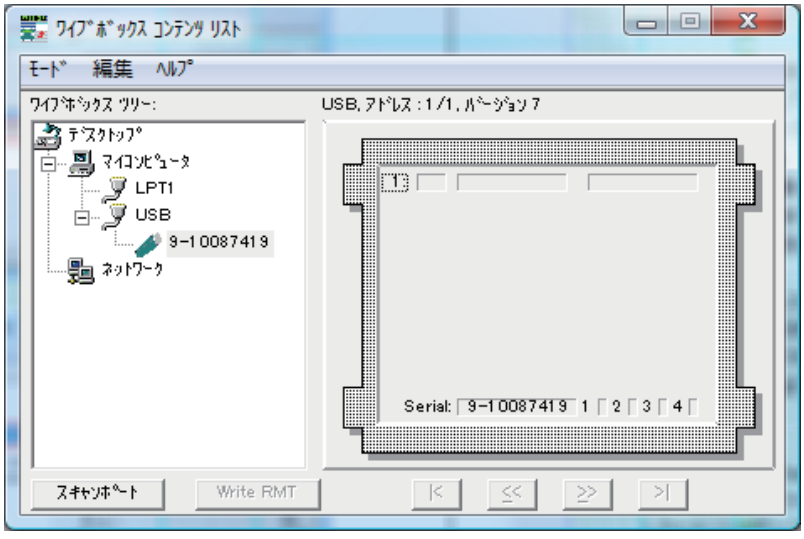
コード設定ツール Wklist32.exe について

4-1. コード設定ツール「Wklist32.exe」の基本画面

プロフェッショナル版（WKUSB、WK25P、WK25ST）



スケルトン版（WKUSB/R）



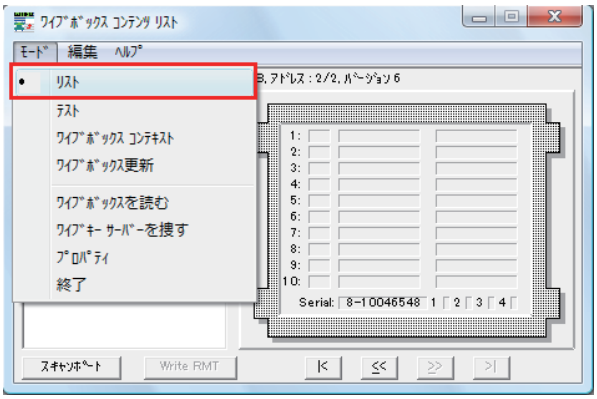
「モード」メニュー
・リスト ・・・「ワイブボックスコンテンツリスト」ウィンドウを表示します。(起動直後はこの画面)
・テスト ・・・・「ワイブボックス テスト」ウィンドウを表示します。
・ワイブボックスコンテキスト ・・・・「ワイブボックス コンテキスト」ウィンドウを表示します。
・ワイブボックス更新 ・・・・「ワイブボックス 更新」ウィンドウを表示します。
・ワイブボックスを読む ・・・・ ローカル装着のワイブキーを検索します。
・ワイブキーサーバーを捜す　・・・・ LAN装着のワイブキー(ワイブキーサーバー)を検索します。
・プロパティ ・・・・ ワイブキーのプロパティを表示します(実際には編集メニューの編集と同じ)。
・終了　・・・・・・・ "Wklist32.exe" を終了します。

「編集」メニュー
・編集 ・・・・ワイブキーの編集(=登録)を行ないます。
　　「ワイブボックスコンテンツリスト」ウィンドウが表示されている場合にのみ有効です。
・コンフィギュアボックス ・・・・ ワイブキーのコンフィグレーションを編集します。
・Remote Context File ・・・・ リモートコンテキストファイル(RTCファイル)を読み込みます。

「ヘルプ」メニュー
・ワイブボックスリストについて ・・・・ “Wklist32.exe” のバージョン情報を表示します。

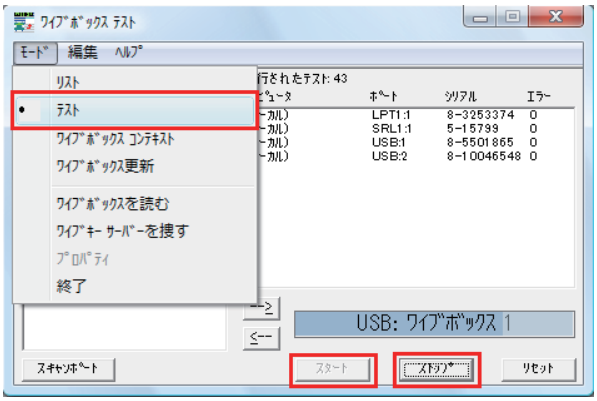
1.「モード」メニュー

[リスト]
現在装着されているワイブキーを全て表示します。"Wklist32.exe"を起動した直後はこの画面が表示されます。



[テスト]
装着されているワイブキーのテストを行ないます。[スタート]ボタンをクリックするとテストが開始されます。

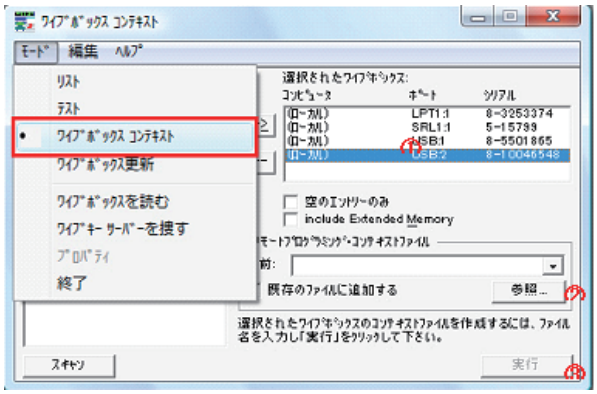
テストが正常に終了した回数がカウントアップ表に表示されます。エラーが発生した場合はエラー回数がカウントアップ表示されます。[ストップ] ボタンをクリックするとテストが終了します。



[ワイブボックスコンテキスト]

装着されているワイブキーのリモートプログラミング・コンテキストファイル（RTCファイル）を作成します。これはコントロールパネル「ワイブキー」ツールの、「ワイブボックスコンテキスト」と同じものです。

(①)にリストアップされているワイブキーのリモートプログラミング・コンテキストファイル（RTCファイル）が作成されます。



※　複数のワイブキーがリストアップされている場合、すべてのワイブキーについて情報を取得して１つのファイルに書き出されます。従い、RTCファイルを作成したいワイブキーを、必ず［<--］ボタンでリストから外すか、PCから取り外し再読込みを行なってから作成してください。

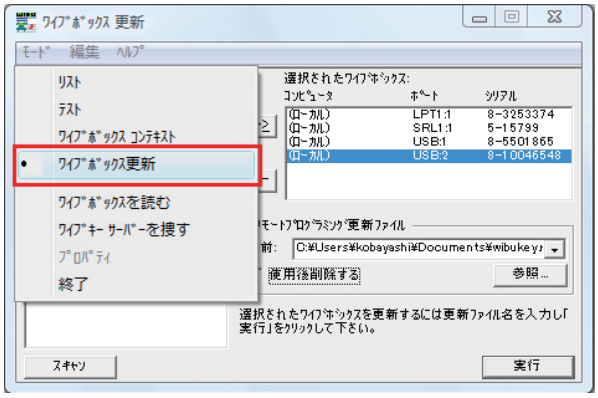
［参照］ボタン（②）でリモートプログラミング・コンテキストファイルを保存するフォルダとファイル名を指定します。［実行］ボタン（③）をクリックするとリモートプログラミング・コンテキストファイルが作成されます。

通常は、「空のエントリのみ」と「include Extended Memory」はチェックを入れないでご利用ください。「空のエントリのみ」をチェックすると、プログラミング済みのエントリが非表示の状態で保存されます。「include Extended Memory」は、拡張メモリ付きのワイブキーのみで有効です。チェックしない場合でも、User Typeメモリの最初の１ページ（32バイト）が保存されます。

[ワイブボックス更新]

装着されているワイブキーの内容をリモートプログラミング・コンテキストファイル（RTUファイル）を使って更新します。

これはコントロールパネルのWIBU-KEYアプレットの「ワイブボックス更新」と同じものです。
リストアップされているワイブキーを、リモートプログラミング・コンテキストファイルを使って更新します。［参照］ボタンで、更新に使用するRTUファイルを選択します。［実行］ボタンをクリックするとワイブキーの更新が行なわれます。



[ワイブボックスを読む]

Localに装着されているワイブキーを検索して情報を更新します。これは［スキャン］ボタンでマイコンピュータをスキャンしたのと同じ動作になります。

[ワイブキーサーバーを捜す]

LAN上のワイブキーを検索して情報を更新します。これは［スキャン］ボタンでネットワークをスキャンしたのと同じ動作になります。

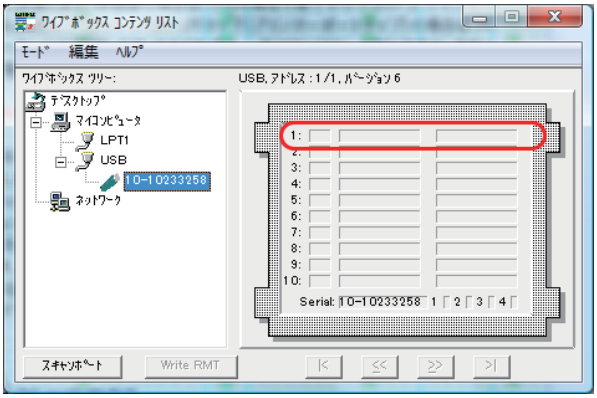
2.「編集」メニュー

[編集]

ワイブキーの内容を編集（プログラミング）します。ワイブボックスコンテンツリストで編集したいエントリ（赤枠部分）をダブルクリックすると編集画面が表示されます。

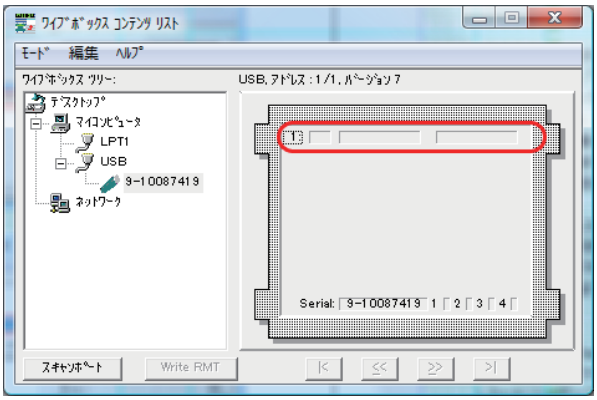
プロフェッショナル版の場合

エントリ１からエントリ10までの10個のエントリを編集します。



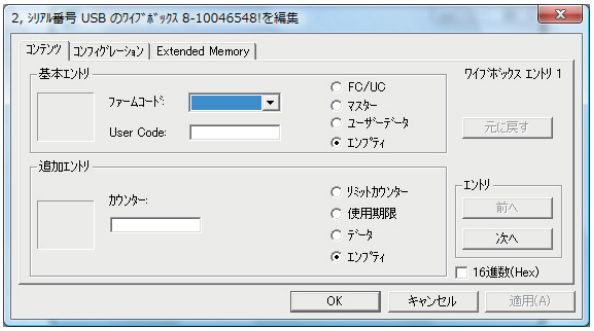
スケルトン版の場合

エントリ１の１個のエントリを編集します。



[コンテンツ]

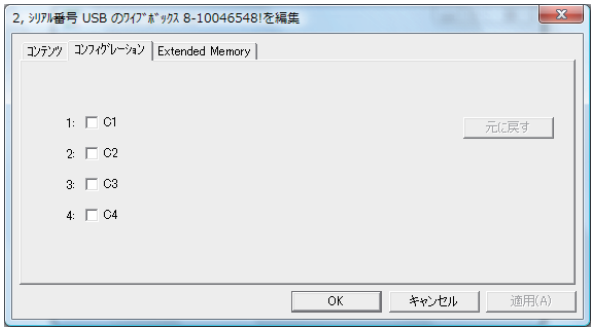
右の編集画面で、ワイブキーのセキュリティ内容を登録します。



[コンフィグレーション]

ワイブキーのコンフィグレーションを設定します（セキュリティデータには影響を与えません）。通常はデフォルトのままで使用します。

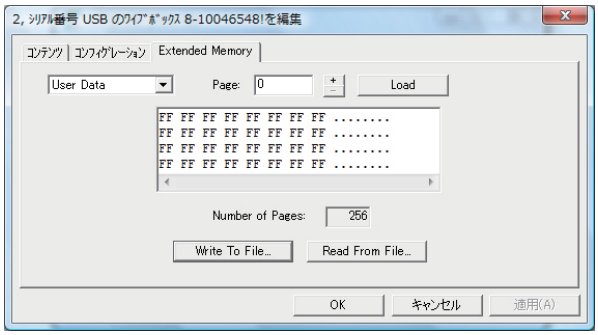
装着されているワイブキーの種類によって表示される画面が多少異なります。（LPT用ワイブキーWK25Pの場合は、[UD]と[BD]が表示されます。）



UD･･･単方向モード　　BD･･･双方向モード　　C2～C4･･･未使用

[Extended Memory]

ワイブキーの拡張メモリに書き込まれたデータを参照、編集することができます。(拡張メモリの具体的な使い方は、後述の「 Chapter 8 拡張メモリー(ExtMem)について」を参照してください。)



[Remote Context File]

[編集]-[Remote Context File]を開き、保存されているRTCファイルを開いてください。
編集の対象としてリモートプログラミング・コンテキストファイルを読み込み、読み込んだ内容を通常のワイブボックスと同様に編集し、結果を[Write RMT]ボタンで RTUファイルとして書き出すことができます。(具体的な使い方は、後述の「 Chapter 6 リモートアップデート機能について」をご参照ください。)

3.「ヘルプ」メニュー

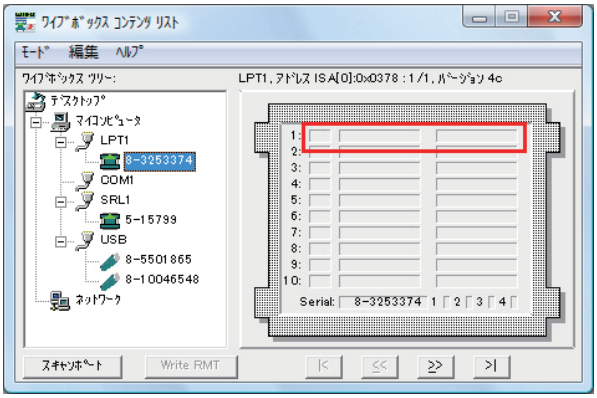
[ワイブボックスリストについて]

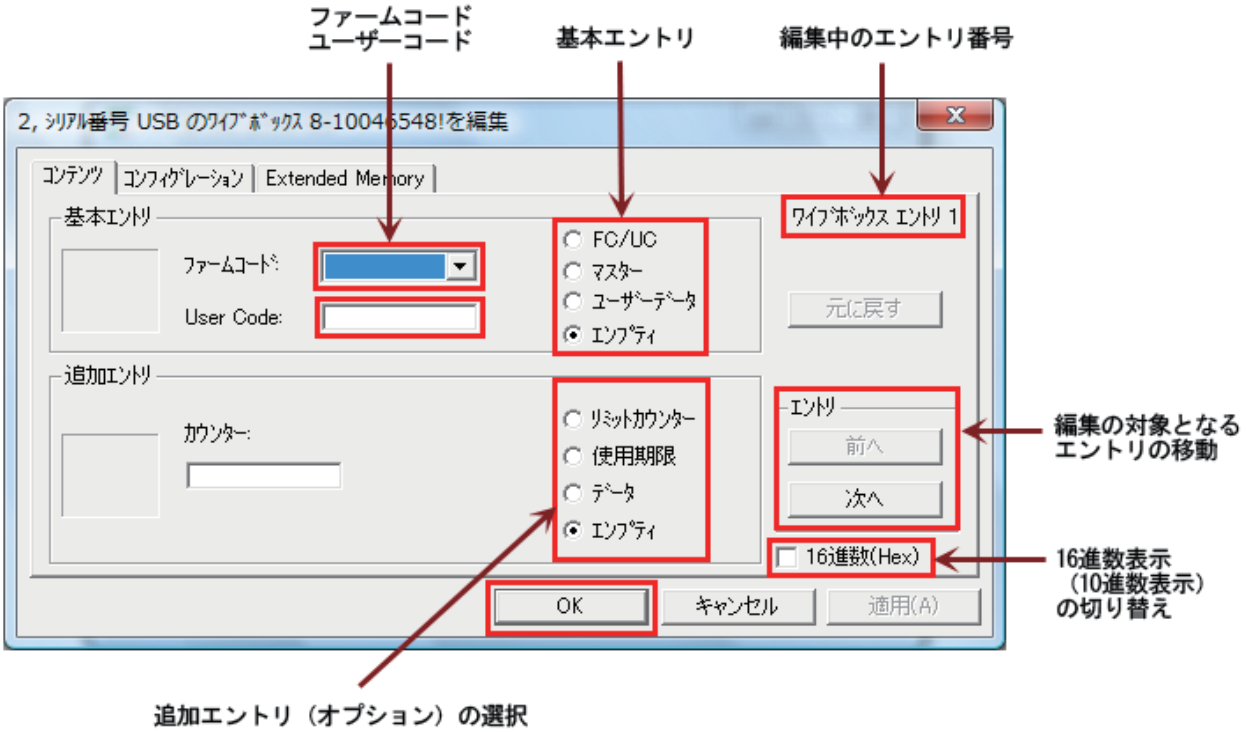
"Wklist32.exe" のバージョン情報を表示します。

4-2. ワイブキー（ハードウェア）の編集

ワイブボックスコンテンツリスト画面で編集するエントリをダブルクリック（ または「クリック」→【編集】メニュー→【編集...】をクリック）します。
（例）　エントリ１を編集する場合、赤枠部分をダブルクリックします。

以下の編集ウィンドウが表示されます。



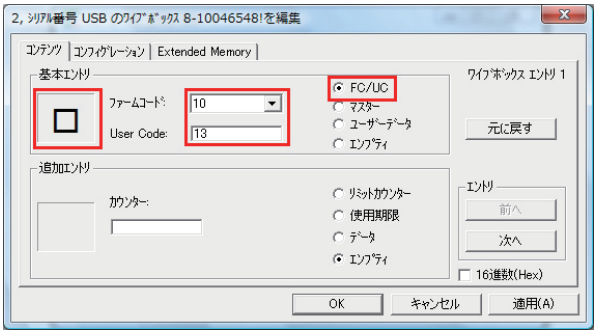


次に各項目の詳細について説明します。

1. 基本エントリの設定

[FC/UC]

FirmCode／UserCodeを設定します。リミットカウンター、使用期限、ユーザーデータの追加エントリ（オプション）を使用する場合も、基本エントリは FC/UC になります。FC/UCを使用すると、基本エントリに FC/UCのシンボルマークが表示されます。

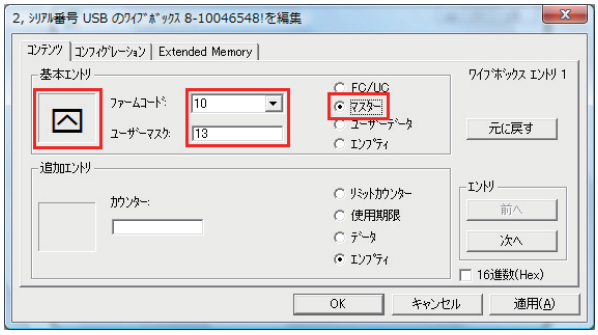


ファームコード

貴社のFirmCode（ファームコード）を設定します。FirmCodeは貴社FSBに記載されている専用コードです。

UserCode
ユーザーコード（24ビット整数）を設定します。入力値は0～16777215 の範囲の整数値です。

[マスター]（プロフェッショナル版のみ）

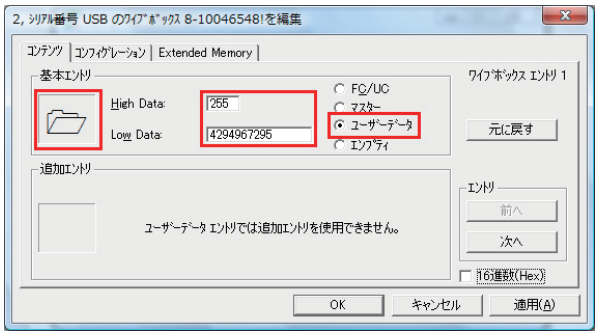


ファームコードとユーザーマスクを設定します。ユーザーマスクは複数のユーザーコードを 1 つにまとめたもので、1 つのユーザーマスクで最大20個までのユーザーコード（User Code）をまとめることができます。マスター機能を使用すると、基本エントリにマスタのシンボルマークが表示されます。
* マスター機能はスケルトン版にはありません。

ファームコード
貴社のFirmCode（ファームコード）を設定します。FirmCodeは貴社FSBに記載されている専用コードです。

ユーザーマスク
複数のユーザーコード(User Code)のOR演算値を入力します。（詳細については「4-3. マスター機能について（プロフェッショナル版のみ）」をご参照ください。）

[ユーザーデータ]



「ユーザーデータ」とは、各エントリに格納する整数値のデータです。

「ユーザーデータ」の設定範囲は以下のとおりです。
「High Data」: 0～255までの整数値。（1バイト）
「Low Data」: 0～4294967295までの整数値。（4バイト）

基本エントリの「ユーザーデータ」はコントロールパネルのWIBU-KEYアプレットから編集が可能な（つまりエンドユーザーによって変更が可能）データです。（追加エントリの「ユーザーデータ」の場合は、WIBU-KEYアプレットからの編集は不可能です。）

「ユーザーデータ」を使用すると、「基本エントリ」に「ユーザーデータ」のシンボルマークが表示されます。なお、「ユーザーデータ」には「追加エントリ」を設定することができません。

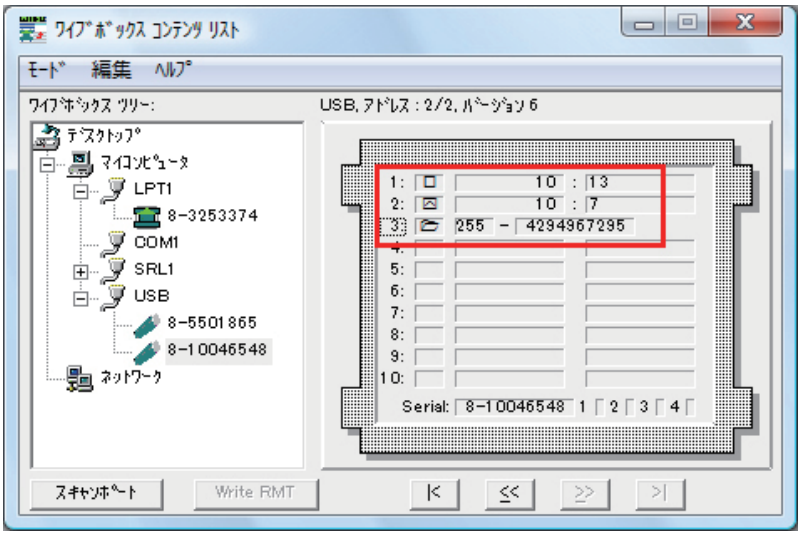
[エンプティ]

エンプティはエントリに登録された内容を消去するオプションです。エンプティを選択して下の「OK」ボタンまたは「適用」ボタンをクリックすると指定したエントリの内容が消去されます。エンプティを選択するとシンボルマークは表示されません。

以下はエントリ 1 からエントリ3に

エントリ1: FC/UC (Firm Code = 10, User Code = 13)
エントリ2: マスター (Firm Code = 10, Master Code = 7)
エントリ3: ユーザーデータ (High Data = 255, Low Data = 4294967295)

を設定し、ワイブボックスコンテンツリストで表示した例です。エントリ番号の直後に表示されるシンボルマークにより、基本エントリがどのようなタイプで設定されているのかがわかります。



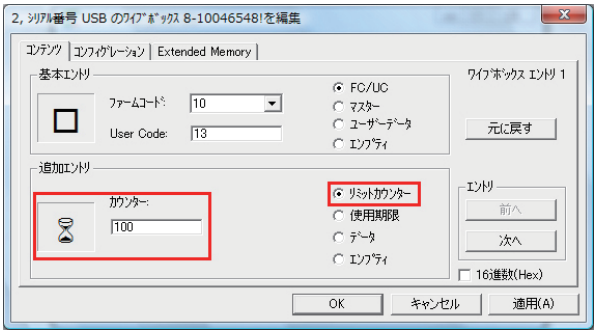
2. 追加エントリの設定

追加エントリは、すでに設定済みの基本エントリに対して機能を追加する形で設定します。
追加エントリを使用すると、基本エントリと合わせて 2 つのエントリを使用することになります。
利用できる追加エントリは以下の4種類です。

1. 「リミットカウンター」: プログラム起動回数を設定します。
2. 「使用期限」: プログラム使用有効期限を設定します。
3. 「データ」: ユーザーデータを設定します。
4. 「エンプティ」: 追加エントリを削除します。

[リミットカウンター]

リミットカウンターはプログラムの起動回数を制御するオプション機能です。ここで指定したカウンター値が実際のプログラム起動回数になります。基本エントリで指定したFirm CodeとUser Codeで暗号化されたプログラムが起動するたびに、カウンター値が 1 つずつ減ります。カウンター値が0になった時点でプログラムが起動できなくなります。

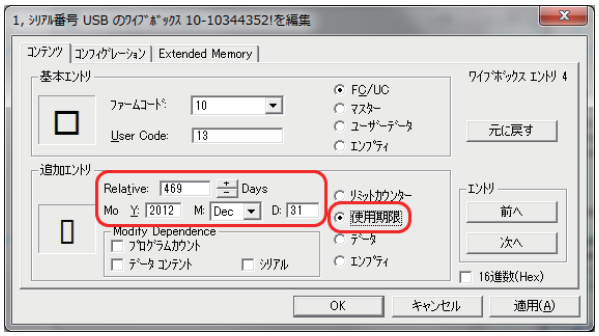


リミットカウンターオプションを選択し、カウンター欄に整数値（使用回数）を 1 ～ 1048575 の範囲内で入力します。

[使用期限]

「使用期限」はプログラムの使用有効期限を設定するオプション機能です。ここで指定した日を過ぎると基本エントリで指定したFirm CodeとUser Codeを持つ暗号化プログラムが起動できなくなります。

使用期限オプションを選択し、日付を設定します。年月日を手動で設定するか、Relative欄に日数を入力します。



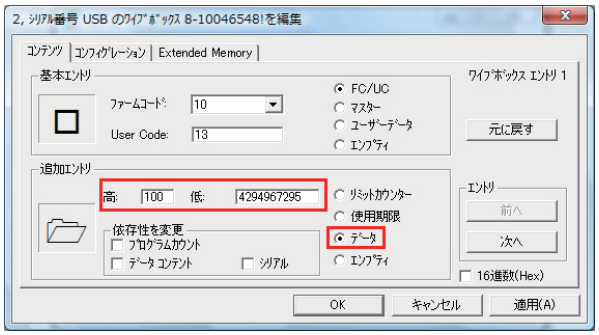
[注意]

ワイブキーの「使用有効期限」機能は、ワイブキーが装着されているPCのシステム時計だけをチェックしています。従い、有効期限が過ぎても、システム時計を過去に戻すことで「使用有効期限」が無効になる点をご了承ください。なお、別商品の「コードメータ」の場合は、内部クロック機能がありますので、システム時計を戻しても有効期限は無効化しません。

[データ]

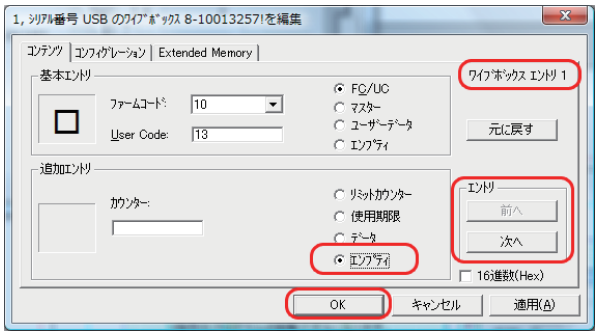
基本エントリの「ユーザーデータ」と基本的には同じです。異なる点は、基本エントリの「ユーザデータ」はコントロールパネルのWIBU-KEYアプレットから編集ができますが、追加エントリの「ユーザーデータ」は編集ができない点です。従い、ユーザーのシリアル情報や商品の製品番号など、ユーザーが勝手に変更できないような内容を保存するのに便利です。

「データ」の設定範囲は以下のとおりです。
「高」: 0～255までの整数。(1バイト)
「低」: 0～4294967295までの整数。(4バイト)



[エンプティ]

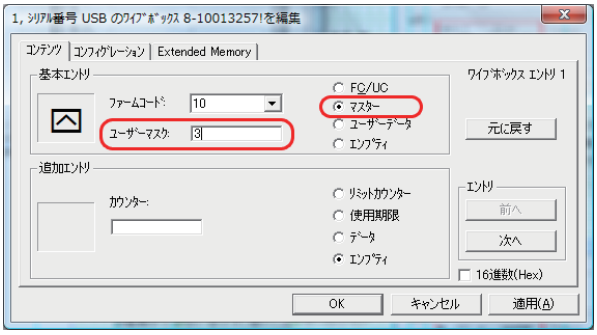
エンプティは追加エントリに登録された内容を消去するオプションです。エンプティを選択して下の「OK」ボタンまたは「適用(A)」ボタンをクリックすると指定した追加エントリの内容が消去されます。



4-3. マスター機能について（プロフェッショナル版のみ）

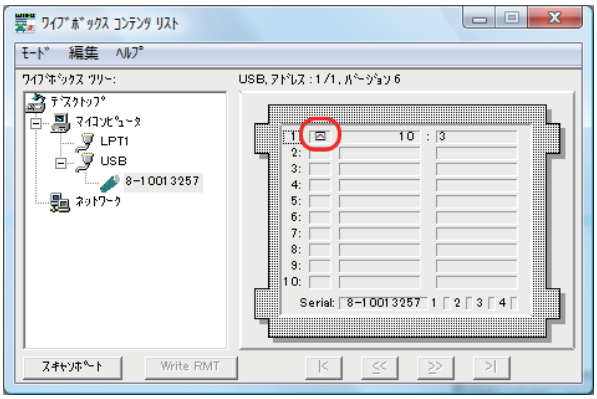
マスター機能とは、1 つの整数値で複数のユーザーコードを兼ねるものです。このマスター機能はプロフェッショナル版のみ対応しており、スケルトン版は対応しておりません。

例えば、マスターコードを 3 に設定したとします。
（ユーザーマスク欄に 3を入力する。）
3という数字は、2進数では 0011ですので、0011のマスクは"0001、0010、0011"になります。



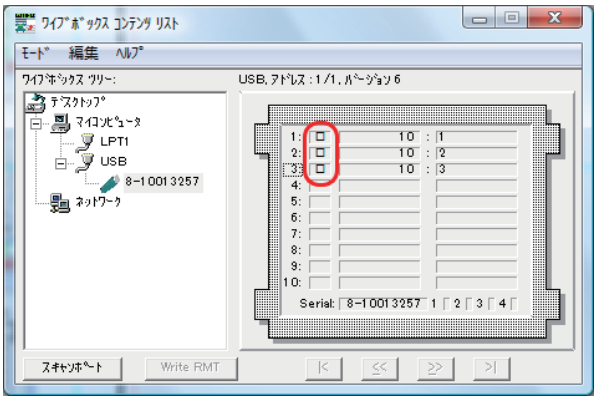
0011のマスクは、

0001
0010
0011



これらのマスク（２進数）を10進数に直すと、

0001 ->1
0010 ->2
0011 ->3



になります。すなわち、0011は、0001=1(10進数)、0010=2(10進数)、0011=3(10進数)のマスクを持ちます。この0011をマスターコードとしてユーザーマスク欄に登録することにより、

ユーザーコード(User Code)=1
ユーザーコード(User Code)=2
ユーザーコード(User Code)=3

をそれぞれのエントリに登録したことと同じになります。（マスターコードとユーザーコードではエントリマークが異なる。赤枠部分。）

マスターコードを使用するメリットは、1 つのエントリで複数のユーザーコードを兼用できる点です。

例えば　15（10進）を指定した場合は、2進では 1111 となり　0001、0010、0011、0100、0101、0110、0111、1000、1001、1010、1011、1100、1101、1110、1111　の15種類をユーザーコードとして認識することができます。つまり、15個のユーザーコードを登録するのに、マスターコードを利用すればエントリの消費が 1 つですむということになります。

このマスター機能を使うことで、1 個のワイブキーの中に最大190種類のユーザーコードを登録することが可能になります。

以下にマスターエントリを使った販売計画の例を示します。
例えば、数種類のアプリケーションモジュール（以下、単にモジュール）を任意の組み合わせて購入できるようなプロダクトがあるとします。例では、モジュール数は 1、2、3、4 と 4種類あり、ユーザーはこれを任意の組み合わせで購入できるとします。開発元は、販売モジュールをそれぞれ以下のUserCodeでプロテクトして1枚のCD-ROMに収めて（またはインターネットのサイトにアップロードして）販売することにします。

モジュール-1　を　UserCode =　1 でプロテクト。
モジュール-2　を　UserCode =　2 でプロテクト。
モジュール-3　を　UserCode =　4 でプロテクト。
モジュール-4　を　UserCode =　8 でプロテクト。

ここで、ユーザーから　モジュール-1　と モジュール-2 を購入したいという希望があったとします。
この場合のマスターエントリコードの計算は、

モジュール-1	UserCode= 1	0001 (2進)
モジュール-2	UserCode= 2	0010 (2進)
		0011 (2進)・・・UserCodeのOR演算値です。

となり、マスターエントリコードが 3（２進で 0011 ）のワイブキーを作成してユーザーに発送（またはリモートプログラミングでエンドユーザーのワイブキーをプログラミング）すれば良いことになります。

同様に、モジュール-3　と モジュール-4 の購入希望があった場合は

モジュール-3	UserCode= 4	0100 (2進)
モジュール-4	UserCode= 8	1000 (2進)
		1100 (2進)・・・ UserCodeのOR演算値です。

となり、マスターエントリコードが 12（２進で 1100 ）のワイブキーを作成します。

また、全てのプロダクトを購入する場合は、

モジュール-1	UserCode= 1	0001 (2進)
モジュール-2	UserCode= 2	0010 (2進)
モジュール-3	UserCode= 4	0100 (2進)
モジュール-4	UserCode= 8	1000 (2進)
		1111 (2進)・・・ UserCodeのOR演算値です。

となり、マスターエントリコードが 15（２進で 1111 ）のワイブキーを作成します。

※マスターエントリコードが 15 の場合、理論上は15種類のUserCodeを認識できる訳ですが、それぞれのアプリケーションモジュールを独立して管理する場合は、他のコードと重複するビットを持つ数値は使用できません。

例えば、UserCode=3（2進で 0011 ）でプロテクトしたモジュールを計画に入れると、そのアプリケーションを購入したユーザーは無条件で モジュール-1（=0001）とモジュール-2（=0010） を使用できるという不都合が生じます。

上記の例のように販売するモジュール数が少ない場合は、マスターエントリでは無く、ワイブキーの標準エントリを使用して同様の効果が得られます。マスターエントリが最も威力を発揮するのは10種類を超えるような多くのモジュールを管理する場合です。

UserCodeは 0 x 000000 ～ 0 x FFFFFFの範囲で設定することができます。
従って、アプリケーションモジュールを独立して管理する場合、1 つのエントリだけでは最大24個（24ビット）のUserCodeしか利用できないことになります。
それ以上のUserCodeを管理する場合は、複数のエントリを使って、それぞれ重複しないビットをプログ

ラミングする必要があります。

以下に、6つのエントリを使って、最大120個のモジュールをそれぞれ独立してプロテクト管理する場合のビット構成テーブル表のサンプルを記載します。テーブル表はあくまで参考例ですので、貴社独自で自由に作り変えることが可能ですが、計算が複雑になりますので、できれば後述のテーブル表を使用されることをお勧め致します。

Module nn はプロテクトをかけて販売する貴社アプリケーションを意味します。
最大で120種類のアプリケーションがあるものとして、1 つのエントリ（テーブル）に20種類のUserCodeを割り当て、それぞれのエントリ（テーブル）が互いに干渉しないようにビット配列を計算してあります。
貴社アプリケーションモジールを任意の Module nn に対応させて配置し、そのモジュールに割り当てられたUserCodeでプロテクトをかけて販売します。
ユーザーから複数のアプリケーションモジュールの購入依頼があったら、それらのUserCodeをOR演算した値をマスターエントリのUserCodeとしてワイブキーに登録し、ユーザーに発送（またはリモートプログラミング）します。

（ 最初に、全モジュールのUserCode をOR演算したマスターエントリをワイブキーに登録しておいて、購入依頼があったアプリケーションモジュールをあとから送るという使い方はできません。 この場合、ワイブキーはすでに全モジュールに対応しているため、未購入のモジュールでもコピーするだけで不正に使用することができます。）

[NOTE]
マスター機能は、1 個のワイブキーで11種類以上のアプリケーションまたはモジュールをプロテクト管理する場合に有効です。プロテクト管理するアプリケーションまたはモジュールが10種類以内の場合は、FC/UC（ファームコード/ユーザーコード）による管理の方が使いやすく便利です。

また、11種類以上のプロテクト管理を行う場合は、別商品「コードメータ」のご利用をお勧め致します。「コードメータ」は、6,000種類までのプロテクト管理が可能で、FC/UCと同じ感覚で使用できとても便利です。OR演算のような計算が必要ありません。

【表の見方（使い方）】

便宜上、エントリ5～10 を使用するものとして説明しています。従ってエントリ番号は仮のもので実際には 1~10のどのエントリに割り当てても問題ありません。1 つのテーブルは 1 つのエントリに対応します。

エントリ5用テーブル表：

To be combined in Entry 5:	ビット（2 進）表記	16 進表記	10 進表記
Module 1:	0011 0000 0000 0000 0000 0001	= 300001 (Hex)	= 3145729 (Dec) = UserCode;
Module 2:	0011 0000 0000 0000 0000 0010	= 300002 (Hex)	= 3145730 (Dec) = UserCode;
	・		
	（中 略）		
	・		
Module 19:	0011 0100 0000 0000 0000 0000	= 340000 (Hex)	= 3407872 (Dec) = UserCode;
Module 20:	0011 1000 0000 0000 0000 0000	= 380000 (Hex)	= 3670016 (Dec) = UserCode;

数値は、2進表記、16進表記、10進表記とも同じものです。
アプリケーションモジュールをプロテクトする時のUserCodeを意味します。

Module nn はプロテクトをかけて販売する貴社アプリケーションを意味します。
最大で120種類のアプリケーションがあるものとして、1 つのエントリ（テーブル）に20種類のUserCodeを割り当て、それぞれのエントリ（テーブル）が互いに干渉しないようにビット配列を計算してあります。
貴社アプリケーションモジールを任意の Module nn に対応させて配置し、そのモジュールに割り当てられたUserCodeでプロテクトをかけて販売します。
ユーザーから複数のアプリケーションモジュールの購入依頼があったら、それらのUserCodeをOR演算した値をマスターエントリのUserCodeとしてワイブキーに登録し、ユーザーに発送（またはリモートプログラミング）します。

（ 最初に、全モジュールのUserCode をOR演算したマスターエントリをワイブキーに登録しておいて、購入依頼があったアプリケーションモジュールをあとから送るという使い方はできません。 この場合、ワイブキーはすでに全モジュールに対応しているため、未購入のモジュールでもコピーするだけで不正に使用することができます。）

以下に、1 エントリあたり20個のUserCodeを配置したテーブル表のサンプルを6つ紹介します。

エントリ 5 用テーブル表：
To be combined in Entry 5:

Module 1: 0011 0000 0000 0000 0000 0001 = 300001 (Hex) = 3145729 (Dec) = UserCode;
Module 2: 0011 0000 0000 0000 0000 0010 = 300002 (Hex) = 3145730 (Dec) = UserCode;
Module 3: 0011 0000 0000 0000 0000 0100 = 300004 (Hex) = 3145732 (Dec) = UserCode;
Module 4: 0011 0000 0000 0000 0000 1000 = 300008 (Hex) = 3145736 (Dec) = UserCode;
Module 5: 0011 0000 0000 0000 0001 0000 = 300010 (Hex) = 3145744 (Dec) = UserCode;
Module 6: 0011 0000 0000 0000 0010 0000 = 300020 (Hex) = 3145760 (Dec) = UserCode;
Module 7: 0011 0000 0000 0000 0100 0000 = 300040 (Hex) = 3145792 (Dec) = UserCode;
Module 8: 0011 0000 0000 0000 1000 0000 = 300080 (Hex) = 3145856 (Dec) = UserCode;
Module 9: 0011 0000 0000 0001 0000 0000 = 300100 (Hex) = 3145984 (Dec) = UserCode;
Module 10: 0011 0000 0000 0010 0000 0000 = 300200 (Hex) = 3146240 (Dec) = UserCode;
Module 11: 0011 0000 0000 0100 0000 0000 = 300400 (Hex) = 3146752 (Dec) = UserCode;
Module 12: 0011 0000 0000 1000 0000 0000 = 300800 (Hex) = 3147776 (Dec) = UserCode;
Module 13: 0011 0000 0001 0000 0000 0000 = 301000 (Hex) = 3149824 (Dec) = UserCode;
Module 14: 0011 0000 0010 0000 0000 0000 = 302000 (Hex) = 3153920 (Dec) = UserCode;
Module 15: 0011 0000 0100 0000 0000 0000 = 304000 (Hex) = 3162112 (Dec) = UserCode;
Module 16: 0011 0000 1000 0000 0000 0000 = 308000 (Hex) = 3178496 (Dec) = UserCode;
Module 17: 0011 0001 0000 0000 0000 0000 = 310000 (Hex) = 3211264 (Dec) = UserCode;
Module 18: 0011 0010 0000 0000 0000 0000 = 320000 (Hex) = 3276800 (Dec) = UserCode;
Module 19: 0011 0100 0000 0000 0000 0000 = 340000 (Hex) = 3407872 (Dec) = UserCode;
Module 20: 0011 1000 0000 0000 0000 0000 = 380000 (Hex) = 3670016 (Dec) = UserCode;

エントリ 6 用テーブル表：
To be combined in Entry 6:

Module 21: 0101 0000 0000 0000 0000 0001 = 500001 (Hex) = 5242881 (Dec);
Module 22: 0101 0000 0000 0000 0000 0010 = 500002 (Hex) = 5242882 (Dec);
Module 23: 0101 0000 0000 0000 0000 0100 = 500004 (Hex) = 5242884 (Dec);
Module 24: 0101 0000 0000 0000 0000 1000 = 500008 (Hex) = 5242888 (Dec);
Module 25: 0101 0000 0000 0000 0001 0000 = 500010 (Hex) = 5242896 (Dec);
Module 26: 0101 0000 0000 0000 0010 0000 = 500020 (Hex) = 5242912 (Dec);
Module 27: 0101 0000 0000 0000 0100 0000 = 500040 (Hex) = 5242944 (Dec);
Module 28: 0101 0000 0000 0000 1000 0000 = 500080 (Hex) = 5243008 (Dec);
Module 29: 0101 0000 0000 0001 0000 0000 = 500100 (Hex) = 5243136 (Dec);
Module 30: 0101 0000 0000 0010 0000 0000 = 500200 (Hex) = 5243392 (Dec);
Module 31: 0101 0000 0000 0100 0000 0000 = 500400 (Hex) = 5243904 (Dec);
Module 32: 0101 0000 0000 1000 0000 0000 = 500800 (Hex) = 5244928 (Dec);
Module 33: 0101 0000 0001 0000 0000 0000 = 501000 (Hex) = 5246976 (Dec);
Module 34: 0101 0000 0010 0000 0000 0000 = 502000 (Hex) = 5251072 (Dec);
Module 35: 0101 0000 0100 0000 0000 0000 = 504000 (Hex) = 5259264 (Dec);
Module 36: 0101 0000 1000 0000 0000 0000 = 508000 (Hex) = 5275648 (Dec);
Module 37: 0101 0001 0000 0000 0000 0000 = 510000 (Hex) = 5308416 (Dec);
Module 38: 0101 0010 0000 0000 0000 0000 = 520000 (Hex) = 5373952 (Dec);
Module 39: 0101 0100 0000 0000 0000 0000 = 540000 (Hex) = 5505024 (Dec);
Module 40: 0101 1000 0000 0000 0000 0000 = 580000 (Hex) = 5767168 (Dec);

エントリ 7 用テーブル表：
To be combined in Entry 7:

Module 41: 0110 0000 0000 0000 0000 0001 = 600001 (Hex) = 6291457 (Dec);
Module 42: 0110 0000 0000 0000 0000 0010 = 600002 (Hex) = 6291458 (Dec);
Module 43: 0110 0000 0000 0000 0000 0100 = 600004 (Hex) = 6291460 (Dec);
Module 44: 0110 0000 0000 0000 0000 1000 = 600008 (Hex) = 6291464 (Dec);
Module 45: 0110 0000 0000 0000 0001 0000 = 600010 (Hex) = 6291472 (Dec);
Module 46: 0110 0000 0000 0000 0010 0000 = 600020 (Hex) = 6291488 (Dec);
Module 47: 0110 0000 0000 0000 0100 0000 = 600040 (Hex) = 6291520 (Dec);
Module 48: 0110 0000 0000 0000 1000 0000 = 600080 (Hex) = 6291584 (Dec);
Module 49: 0110 0000 0000 0001 0000 0000 = 600100 (Hex) = 6291712 (Dec);
Module 50: 0110 0000 0000 0010 0000 0000 = 600200 (Hex) = 6291968 (Dec);
Module 51: 0110 0000 0000 0100 0000 0000 = 600400 (Hex) = 6292480 (Dec);
Module 52: 0110 0000 0000 1000 0000 0000 = 600800 (Hex) = 6293504 (Dec);
Module 53: 0110 0000 0001 0000 0000 0000 = 601000 (Hex) = 6295552 (Dec);
Module 54: 0110 0000 0010 0000 0000 0000 = 602000 (Hex) = 6299648 (Dec);
Module 55: 0110 0000 0100 0000 0000 0000 = 604000 (Hex) = 6307840 (Dec);
Module 56: 0110 0000 1000 0000 0000 0000 = 608000 (Hex) = 6324224 (Dec);
Module 57: 0110 0001 0000 0000 0000 0000 = 610000 (Hex) = 6356992 (Dec);
Module 58: 0110 0010 0000 0000 0000 0000 = 620000 (Hex) = 6422528 (Dec);
Module 59: 0110 0100 0000 0000 0000 0000 = 640000 (Hex) = 6553600 (Dec);
Module 60: 0110 1000 0000 0000 0000 0000 = 680000 (Hex) = 6815744 (Dec);

エントリ 8 用テーブル表：
To be combined in Entry 8:

Module 61: 1001 0000 0000 0000 0000 0001 = 900001 (Hex) = 9437185 (Dec);
Module 62: 1001 0000 0000 0000 0000 0010 = 900002 (Hex) = 9437186 (Dec);
Module 63: 1001 0000 0000 0000 0000 0100 = 900004 (Hex) = 9437188 (Dec);
Module 64: 1001 0000 0000 0000 0000 1000 = 900008 (Hex) = 9437192 (Dec);
Module 65: 1001 0000 0000 0000 0001 0000 = 900010 (Hex) = 9437200 (Dec);
Module 66: 1001 0000 0000 0000 0010 0000 = 900020 (Hex) = 9437216 (Dec);
Module 67: 1001 0000 0000 0000 0100 0000 = 900040 (Hex) = 9437248 (Dec);
Module 68: 1001 0000 0000 0000 1000 0000 = 900080 (Hex) = 9437312 (Dec);
Module 69: 1001 0000 0000 0001 0000 0000 = 900100 (Hex) = 9437440 (Dec);
Module 70: 1001 0000 0000 0010 0000 0000 = 900200 (Hex) = 9437696 (Dec);
Module 71: 1001 0000 0000 0100 0000 0000 = 900400 (Hex) = 9438208 (Dec);
Module 72: 1001 0000 0000 1000 0000 0000 = 900800 (Hex) = 9439232 (Dec);
Module 73: 1001 0000 0001 0000 0000 0000 = 901000 (Hex) = 9441280 (Dec);
Module 74: 1001 0000 0010 0000 0000 0000 = 902000 (Hex) = 9445376 (Dec);
Module 75: 1001 0000 0100 0000 0000 0000 = 904000 (Hex) = 9453568 (Dec);
Module 76: 1001 0000 1000 0000 0000 0000 = 908000 (Hex) = 9469952 (Dec);
Module 77: 1001 0001 0000 0000 0000 0000 = 910000 (Hex) = 9502720 (Dec);
Module 78: 1001 0010 0000 0000 0000 0000 = 920000 (Hex) = 9568256 (Dec);
Module 79: 1001 0100 0000 0000 0000 0000 = 940000 (Hex) = 9699328 (Dec);
Module 80: 1001 1000 0000 0000 0000 0000 = 980000 (Hex) = 9961472 (Dec);

エントリ９用テーブル表：
To be combined in Entry 9:

Module 81: 1010 0000 0000 0000 0000 0001 = A00001 (Hex) = 10485761 (Dec);
Module 82: 1010 0000 0000 0000 0000 0010 = A00002 (Hex) = 10485762 (Dec);
Module 83: 1010 0000 0000 0000 0000 0100 = A00004 (Hex) = 10485764 (Dec);
Module 84: 1010 0000 0000 0000 0000 1000 = A00008 (Hex) = 10485768 (Dec);
Module 85: 1010 0000 0000 0000 0001 0000 = A00010 (Hex) = 10485776 (Dec);
Module 86: 1010 0000 0000 0000 0010 0000 = A00020 (Hex) = 10485792 (Dec);
Module 87: 1010 0000 0000 0000 0100 0000 = A00040 (Hex) = 10485824 (Dec);
Module 88: 1010 0000 0000 0000 1000 0000 = A00080 (Hex) = 10485888 (Dec);
Module 89: 1010 0000 0000 0001 0000 0000 = A00100 (Hex) = 10486016 (Dec);
Module 90: 1010 0000 0000 0010 0000 0000 = A00200 (Hex) = 10486272 (Dec);
Module 91: 1010 0000 0000 0100 0000 0000 = A00400 (Hex) = 10486784 (Dec);
Module 92: 1010 0000 0000 1000 0000 0000 = A00800 (Hex) = 10487808 (Dec);
Module 93: 1010 0000 0001 0000 0000 0000 = A01000 (Hex) = 10489856 (Dec);
Module 94: 1010 0000 0010 0000 0000 0000 = A02000 (Hex) = 10493952 (Dec);
Module 95: 1010 0000 0100 0000 0000 0000 = A04000 (Hex) = 10502144 (Dec);
Module 96: 1010 0000 1000 0000 0000 0000 = A08000 (Hex) = 10518528 (Dec);
Module 97: 1010 0001 0000 0000 0000 0000 = A10000 (Hex) = 10551296 (Dec);
Module 98: 1010 0010 0000 0000 0000 0000 = A20000 (Hex) = 10616832 (Dec);
Module 99: 1010 0100 0000 0000 0000 0000 = A40000 (Hex) = 10747904 (Dec);
Module 100: 1010 1000 0000 0000 0000 0000 = A80000 (Hex) = 11010048 (Dec);

エントリ１０用テーブル表：
To be combined in Entry 10:

Module 101: 1100 0000 0000 0000 0000 0001 = C00001 (Hex) = 12582913 (Dec);
Module 102: 1100 0000 0000 0000 0000 0010 = C00002 (Hex) = 12582914 (Dec);
Module 103: 1100 0000 0000 0000 0000 0100 = C00004 (Hex) = 12582916 (Dec);
Module 104: 1100 0000 0000 0000 0000 1000 = C00008 (Hex) = 12582920 (Dec);
Module 105: 1100 0000 0000 0000 0001 0000 = C00010 (Hex) = 12582928 (Dec);
Module 106: 1100 0000 0000 0000 0010 0000 = C00020 (Hex) = 12582944 (Dec);
Module 107: 1100 0000 0000 0000 0100 0000 = C00040 (Hex) = 12582976 (Dec);
Module 108: 1100 0000 0000 0000 1000 0000 = C00080 (Hex) = 12583040 (Dec);
Module 109: 1100 0000 0000 0001 0000 0000 = C00100 (Hex) = 12583168 (Dec);
Module 110: 1100 0000 0000 0010 0000 0000 = C00200 (Hex) = 12583424 (Dec);
Module 111: 1100 0000 0000 0100 0000 0000 = C00400 (Hex) = 12583936 (Dec);
Module 112: 1100 0000 0000 1000 0000 0000 = C00800 (Hex) = 12584960 (Dec);
Module 113: 1100 0000 0001 0000 0000 0000 = C01000 (Hex) = 12587008 (Dec);
Module 114: 1100 0000 0010 0000 0000 0000 = C02000 (Hex) = 12591104 (Dec);
Module 115: 1100 0000 0100 0000 0000 0000 = C04000 (Hex) = 12599296 (Dec);
Module 116: 1100 0000 1000 0000 0000 0000 = C08000 (Hex) = 12615680 (Dec);
Module 117: 1100 0001 0000 0000 0000 0000 = C10000 (Hex) = 12648448 (Dec);
Module 118: 1100 0010 0000 0000 0000 0000 = C20000 (Hex) = 12713984 (Dec);
Module 119: 1100 0100 0000 0000 0000 0000 = C40000 (Hex) = 12845056 (Dec);
Module 120: 1100 1000 0000 0000 0000 0000 = C80000 (Hex) = 13107200 (Dec);

Chapter 5

自動暗号化ツール AxProtector について

5-1. 自動暗号化ツール AxProtector について

ワイブキー(WIBU-KEY)には、ソースコードを変更せずに、貴社のEXEやDLLなどの実行形式プログラムを強力に暗号化する自動暗号化ツール「AxProtector」が用意されています。貴社のFirm Code/User Codeを取り込みながらアプリケーションプログラムを強力に自動暗号化します。

また、メモリー上で展開されるコードを常に暗号化し、必要な時に必要なモジュールを実行する「メモリー上のオンデマンド復号機能」を使用することができます。メモリー上でもコードが暗号化されているため、ハッキングに対する強力なセキュリティを実現することが可能になります。

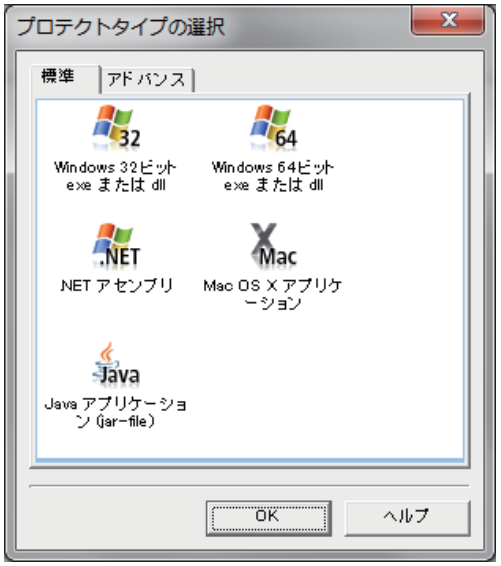
AxProtectorの対象になるファイルは以下のとおりです。

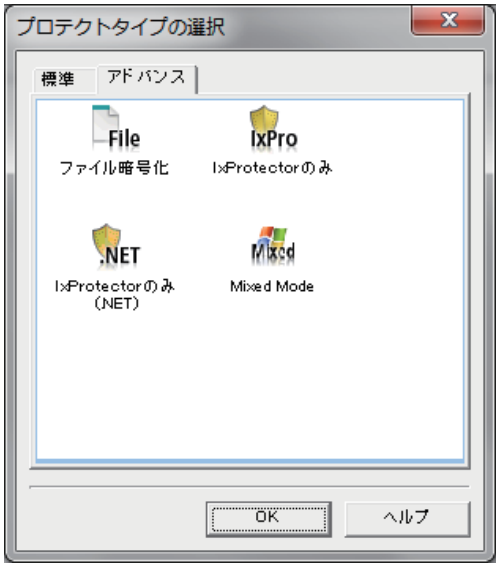
1. Windows32bit 実行形式プログラム (EXE, DLL)
2. Windows64bit 実行形式プログラム (EXE, DLL)
3. .NET アセンブリ実行形式プログラム
4. Mac OS X 実行形式プログラム
5. Javaプログラム

また、アドバンス機能として、

6. ファイル暗号化(データファイル暗号化)
7. IxProtectorのみ使用
8. IxProtectorのみ使用(.NET)
9. Mixed Mode

があります。



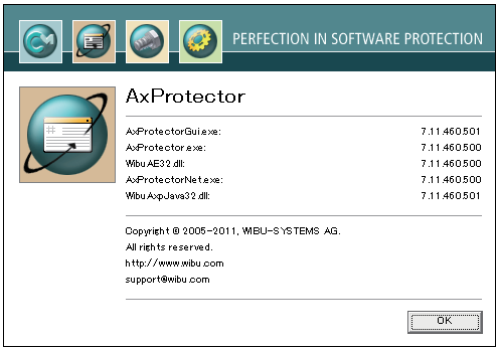


AxProtectorGui.exeは、¥Program Files¥WIBU-SYSTEMS¥AxProtector¥DevKit¥binフォルダに存在します。

【スタート】→【すべてのプログラム(P)】→【AxProtector】→【AxProtector】から起動できます。

AxProtector 7.11 バージョン

AxProtectorGui.exe 7.11.460.501
AxProtector.exe 7.11.460.500
WibuAE32.dll 7.11.460.500
AxProtectorNet.exe 7.11.460.500
WibuAxpJava32.dll 7.11.460.501



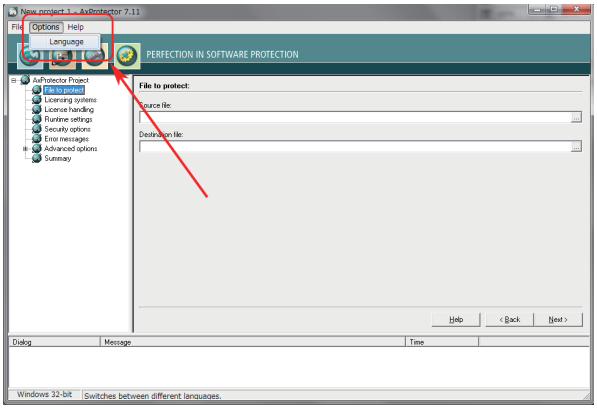
Hotfix について：

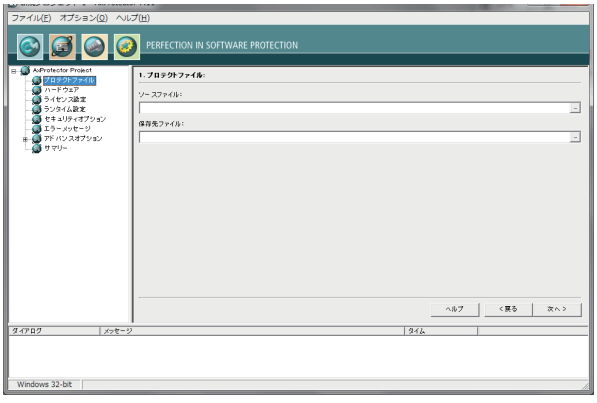
.NET プログラムをAxProtectorで暗号化する場合、Microsoft Windows Updateを行っていないPCでは、下記HotfixをOS環境に合わせてインストールする必要があります。インストールする必要があるのは、AxProtectorで暗号化作業を行うPCのみで、暗号化されたプログラムを実行するPC（ユーザー先）には必要ありません。これは、.NET Framework 2.0 Service Pack1における不具合によるもので、詳細につきましてはマイクロソフトサイトKB950986 (**http://support.microsoft.com/kb/950986/ja**) をご参照ください。なお、Microsoft Windows Updateを定期的に行っているPCではインストールする必要がありません。また、Windows 7 (32bit/64bit)以降は必要ありません。

Hotfix 401752 (KB950986) Windows 2000/XP 32-bit用 NDP20SP1-KB950986-x86.exe
Hotfix 401752 (KB950986) Windows XP 64-bit用 NDP20SP1-KB950986-x64.exe
Hotfix 401752 (KB950986) Windows Vista 32-bit用 Windows6.0-KB950986-x86.msu
Hotfix 401752 (KB950986) Windows Vista 64-bit用 Windows6.0-KB950986-x64.msu

5-2. 画面を日本語モードにする

[Options]-[Language]を選択し、「言語選択」画面で"Japanese"を選択し、OKをクリックします。AxProtectorが日本語モードに変換されます。





5-3. AxProtector のメニュー画面

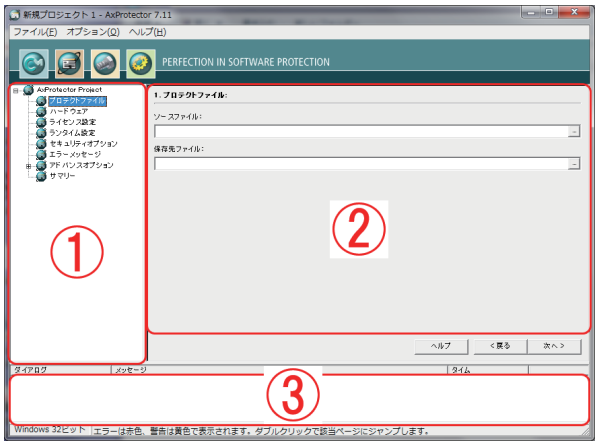
①のペインにはAxProtectorの操作メニュー項目が表示されます。各メニュー項目を上から順に実行すれば自動的に全てのパラメータを設定できるようになっています。ここで選択したメニュー項目に対するパラメータの値は②のペインで入力します。

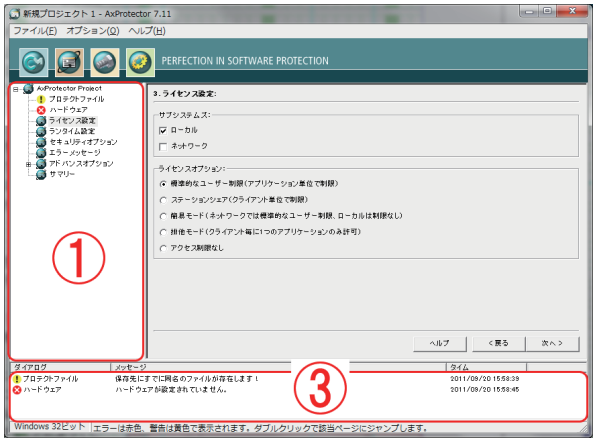
②のペインで実際のパラメータを入力、設定します。入力が終了したら「次へ」ボタン をクリックすると、次のメニュー項目の設定画面に切り替わります。また、任意のメニュー項目を直接クリックして設定画面を表示することもできます。

③のペインにはエラーまたは警告が発生した場合に、その内容がリアルタイムで表示されます。

AxProtectorを操作した結果は、①と③のペインにアイコンを伴ってリアルタイムに表示されます。

なお、項目を選択した時の状況によっては、①と③のペインに、エラーや警告のアイコンが表示されることがありますが、そのまま先に進めて構いません。その後の操作で適切な値を設定することによってこれらのアイコンは表示されなくなります。最終的に、ファイルの暗号化を実行する時点でエラーや警告のアイコンが表示されなければ問題ありません。





アイコンの説明

・・・ エラーが無く操作が行なわれたことをあらわします。

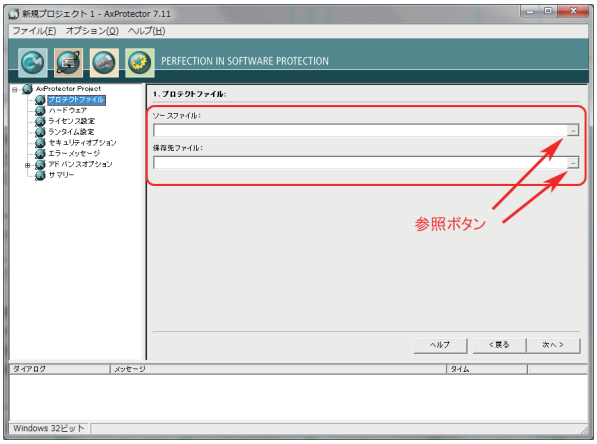
・・・ 警告を表します。警告の内容によっては無視しても構わないケースがありますが、できるだけ警告を解消する必要があります。

・・・ エラーを表します。このアイコンが表示された項目に対して正しい設定をし直す必要があります。このアイコンが表示されている間はプロテクト操作が正常に行なわれません。

5-4. AxProtector の各入力画面の説明

1. プロテクトファイル

暗号化する前のオリジナルファイル名と、暗号化生成されるファイル名を指定します。AxProtectorを起動した直後はこの画面が表示されます。



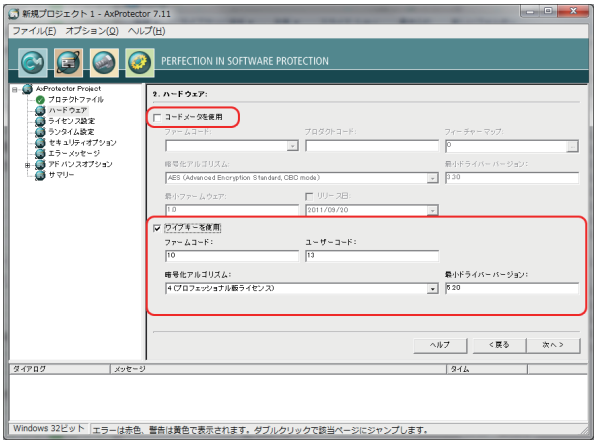
ソースファイル:
暗号化する前のオリジナルファイル名を指定します。右部の参照ボタンからファイルを指定することもできます。

保存先ファイル:
暗号化生成されるファイル名とその保存先のフォルダ名を指定します。右部の参照ボタンからファイルを指定することもできます。

ソースファイルと保存先ファイルを同一にすると、ソースファイルが暗号化生成されるファイルに上書きされますのでご注意ください。ファイル名が同じ場合は別フォルダに保存するか、同じフォルダに保存する場合は、ファイル名を変更するようにしてください。なお、ソースファイルを指定すると、同一フォルダにprotectedフォルダが自動的に作成されます。

2. プロテクトハードウェア

プロテクトに使用するハードウェアキーを指定します。AxProtectorはハードウェアキーとして、ワイブキー の他に「コードメータ」を使用することができます。



☐ コードメータを使用
コードメータを利用した暗号化処理を行います。

☐ ワイブキーを使用
ワイブキーを利用した暗号化処理を行います。

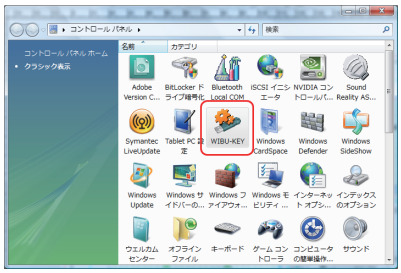
[NOTE]
ワイブキーだけを使用する場合、必ず「ワイブキーを使用」をチェックし、「コードメータを使用」のチェックを外します。

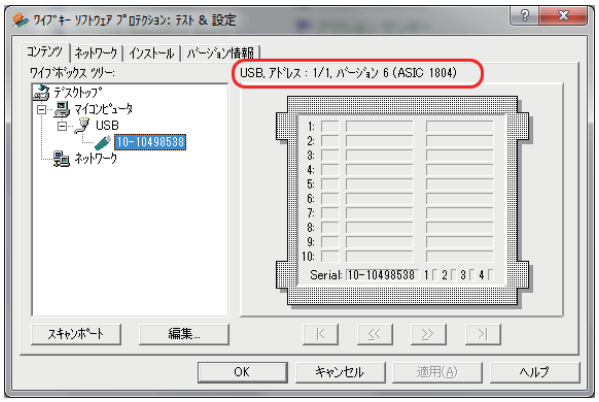
ファームコード:
貴社のファームコードを入力します。
貴社のファームコードは、FSBの表面に記載されています。
プロフェッショナル版の場合は４桁の整数値、スケルトン版の場合は6桁の整数値。

ユーザーコード:
使用するユーザーコードを入力します。
ユーザーコードは、0 ～16777215の範囲の整数値が可能。

暗号化アルゴリズム：
プロフェッショナル版は"4 (プロフェッショナル版ライセンス)"、スケルトン版は"5 (スケルトン版ライセンス)"を指定します。プロフェッショナル版ワイブキーで、ハードウェアバージョン4b以前のワイブキーを使用する場合は、暗号化アルゴリズムを"2 (プロフェッショナル版ライセンス/2004年3月以前のハードウェアキー対象)、スケルトン版ワイブキーで、ハードウェアバージョン5以前のワイブキーを使用する場合は、暗号化アルゴリズムを"3 (スケルトン版ライセンス/2004年3月以前のハードウェアキー対象)を指定してください。

各ワイブキーのハードウェアバージョンは、コントロールパネル上にあるWibuke32.CPLを実行して調べることができます。



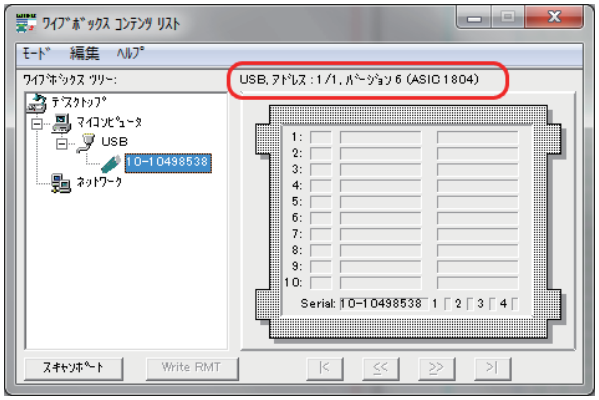


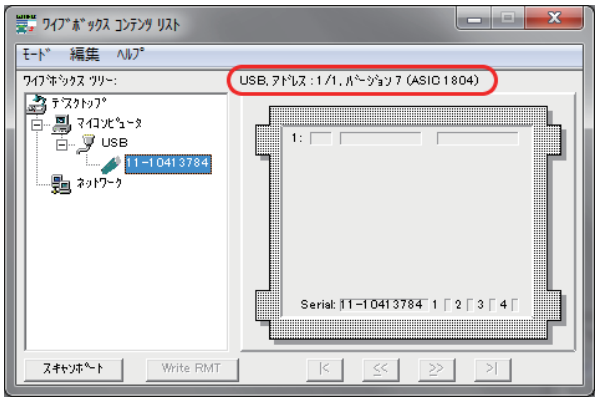
また、コード設定ツールWklist32.exeを起動して調べることもできます。

なお、現時点での最新のハードウェアバージョンは、

プロフェッショナル版の場合："6"
スケルトン版の場合："7"

になります。

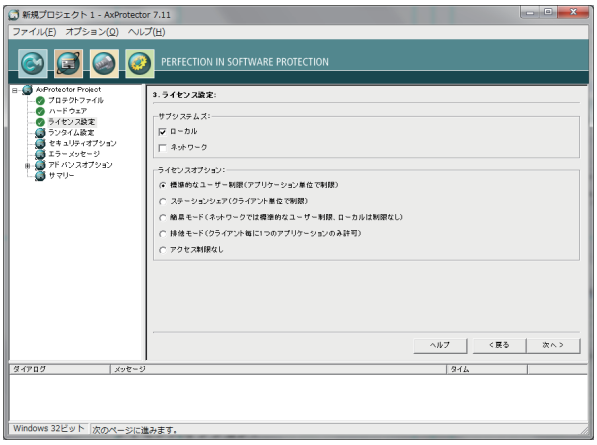




最低限必要なドライバーバージョン：
使用するワイブキー・ドライバーの必要最低バージョンを指定します。

3. ライセンス設定

ローカルPC上またはネットワーク上に装着されたワイブキーを自動検索して認識する設定を行ないます。



サブシステムズ:

□ ローカル
ローカルPCに装着されているワイブキーを検索します。

□ ネットワーク
ネットワーク(LAN)上のワイブキーサーバーに装着されているワイブキー検索します。

両方にチェックを入れた場合は、最初にローカルPCをチェックし、ワイブキーが見つからなければネットワーク上のワイブキーサーバーを検索します。

ライセンスオプション:

○ 標準的なユーザー制限(アプリケーション単位で制限)(Normal User Limit)
実行するアプリケーションごとに 1 つのライセンスを割り当てます。例えば、同じアプリケーションを同時に2回起動する場合は 2つのライセンスが必要になります。この原則はワイブキーがローカルPCにある場合もネットワーク上にある場合も同じように適用されます。
(1アプリケーション = 1ライセンス)

○ ステーションシェア(クライアント単位で制限)(Station Share)
1台のPCで同一のアプリケーションを同時に複数回起動した場合でも 1 ライセンスとして扱われます。
(1PC = 1ライセンス)

○ 簡易モード(ネットワークでは標準的なユーザー制限、ローカルは制限なし)
ネットワーク上のキーに対しては「標準的なユーザー制限」として動作しますが、ローカルPC上のキーに対しては制限がありません。(スケルトン版は使用不可)

○ 排他モード(クライアントごとに 1 つのアプリケーションのみ許可)(Exclusive Mode)
同一クライアント上でのアプリケーションの重複起動を防止します。

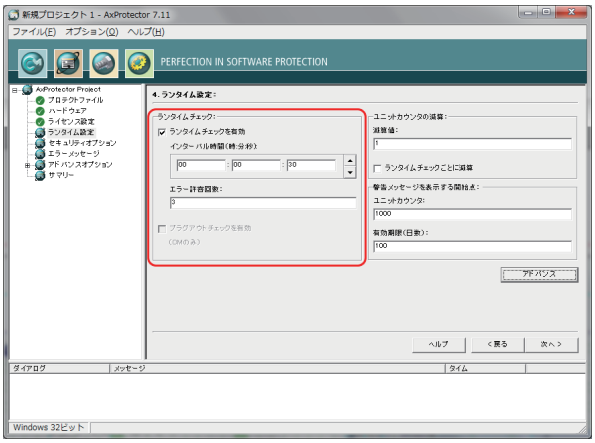
○ アクセス制限なし(ユーザー数無制限)(No User Limit)
起動に必要なワイブキーがネットワーク上で見つかれば、アプリケーションが起動します。ライセンス数の制約を受けません。(スケルトン版は使用不可)

4. ランタイム設定

使用制限機能に関する動作を設定します。

ランタイムチェック：

ランタイムチェックのインターバル時間を設定します。



□ ランタイムチェックを有効
ランタイムチェックを有効にします。

□時間 (時:分:秒) ：
ランタイムチェックにおいてワイブキーのチェックが行なわれてから次のチェックが行なわれるまでのインターバルを時分秒で設定します。デフォルトは30秒（00:00:30）が設定されています。

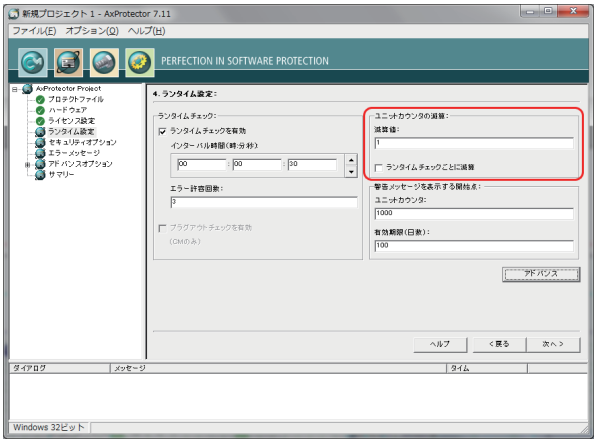
□ エラー許容回数：
この機能は、ランタイムチェックに失敗した時（チェックエラーの場合）、ただちにアプリケーションを終了させず、許可した回数だけエラーを無視してアプリケーションを続行させる機能です。この機能は、WK25P(LPTポート版）やWK25ST（COMポート版）を使用する場合に便利です。プリンタへデータ出力中の場合や、シリアルポートへデータ転送中の場合、パラレルポートやシリアルポートが占有されているとワイブキーデバイスへのアクセスができずにチェックエラーになります。その際、すぐにアプリケーションを終了させるのでなく、指定した回数だけエラーを無視してアプリケーションを続行させる機能です。

例えば、ここで３を設定すると、チェックエラーになっても3回まではアプリケーションを続行させることが可能です。しかし、４回目でエラーになるとアプリケーションを終了させます。その際、自動的にアプリケーションを終了させるのでなく、「中止」か「再試行」（ワイブキーを装着すること）のメッセージを出してから終了させます。この機能を使わず、１回目のランタイムチェックエラーでアプリケーションを中止させたい場合は０を設定します。

□ プラグアウトチェックを有効（CMのみ）
キーをPCから取り外すとすぐにエラーを表示させます。この機能は「コードメータ」のみ使用可能です。

ユニットカウンタの減算:

ワイブキーの使用回数制限（リミットカウンタ＝ユニットカウンタ）の動作を設定します。ユニットカウンタを使用すると、ワイブキーチェックが行なわれるたびにリミットカウンタのチェックが行なわれ、「減算値」で設定した数値が減算されます（ワイブキーでは減算値を1以外に設定できません）。ユニットカウンタが0(ゼロ)になるとアプリケーションの起動ができなくなります。ワイブキーエントリの「追加エントリ」で「リミットカウンタ」オプションが設定されていない場合は、ここで設定したユニットカウンタは無視され減算は行われません。アプリケーションは無制限に起動します。（起動回数は無制限になります。）



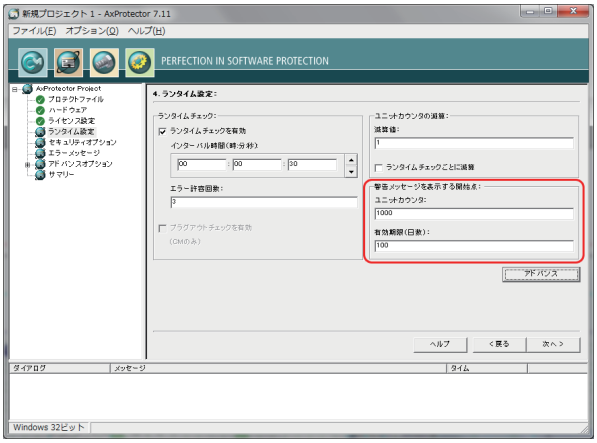
☐ **減算値:**
起動時にワイブキーチェックが行なわれた時に減算する値を設定します。デフォルトは1が設定されています。

☐ **ランタイムチェックのたびに減算:**
ランタイムチェックが行われるごとにリミットカウンタの減算を行います。ランタイムチェック時に減算しない場合はチェックをはずします。この機能は、ランタイムチェックのインターバル時間と組み合わせることで、アプリケーションプログラムの使用可能時間を設定することができます。

警告メッセージを表示する開始点 :

リミットカウンタ(ユニットカウンタ)や有効期限(日数)の残りが一定の値(閾値)を下回ると、警告メッセージを表示させるオプションです。

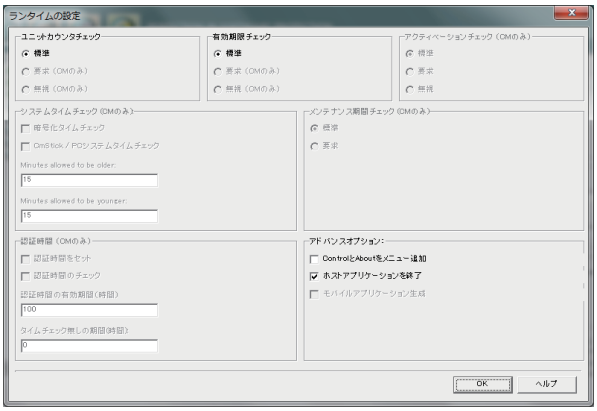
☐ **ユニットカウンタ:**
リミットカウンタ(ユニットカウンタ)残数の閾値を設定します。デフォルトは1000が設定されています。ワイブキーに設定されたリミットカウンタ数が閾値以下になると、プログラムが起動するたびに警告メッセージを出します。



☐ **有効期限(日数):**
プログラムの使用期限の残日数の閾値を設定します。デフォルトは100が設定されています。ワイブキーに設定された使用期限が閾値以下になると、プログラムが起動するたびに警告メッセージを出します。

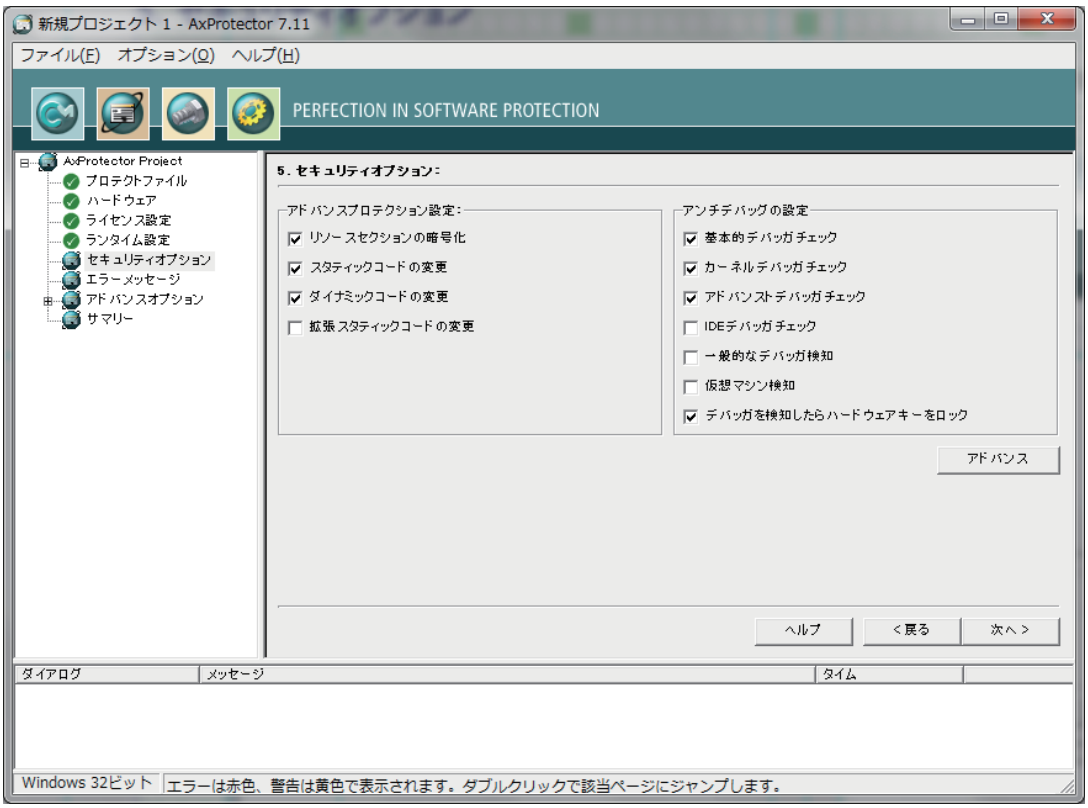
アドバンス :

「アドバンス」ボタンをクリックすると別画面が開きますが、別商品の「コードメータ」でのみ設定できるオプションです。ワイブキーには使用しませんので、ここでは説明を省略します。



5. セキュリティオプション

このセキュリティオプションは、プロテクト強度に関連する設定です。自動暗号化に対する解析強度を強化する場合は、適宜設定してください。



アドバンストプロテクション設定：

高度なプロテクション方法を指定します。解析強度を重視する場合はすべてにチェックを入れてください。（ただし、「スタティックコードの変更」と「拡張スタティックコードの変更」は同時に指定できません）

□リソースセクションの暗号化
リソースを暗号化します。

□スタティックコードの変更
デバッグ、ダンプ、リバースエンジニアリングを不可能にするために、正常にコンパイルされたコードを独自な方法で変更し直します。

□ダイナミックコードの変更
アプリケーションが実行されている間、コードを変更します。

□拡張スタティックコードの変更
「スタティックコードの変更」の拡張版です。「スタティックコードの変更」と同時には指定できません。

アンチデバッグの設定

アンチデバッギングの設定です。解析強度を重視する場合は可能な限りチェックを入れてご利用ください。アプリケーションによって動作に差し障りが生じる場合は、以下を参考にチェックをはずしてみてください。

□基本的なデバッガーチェック
一般的なデバッグプログラムを検知します。デバッグプログラムが見つかった場合、アプリケーションは起動しません。

□カーネルデバッガーチェック
「SoftICE」のようなカーネルデバッグプログラムを検知します。カーネルデバッグプログラムが見つかった場合、アプリケーションは起動しません。

□アドバンストデバッガーチェック
デバッグプログラムの検知をより強化します。デバッグプログラムが見つかった場合、アプリケーションは起動しません。もし、アプリケーション起動中にデバッグプログラムが検知された場合は、アプリケーションを終了させます。

□IDE デバッガーチェック
Visual StudioやDelphiなどの統合開発環境(IDE)のデバッガを禁止します。もし見つかった場合は、アプリケーションが起動しません。

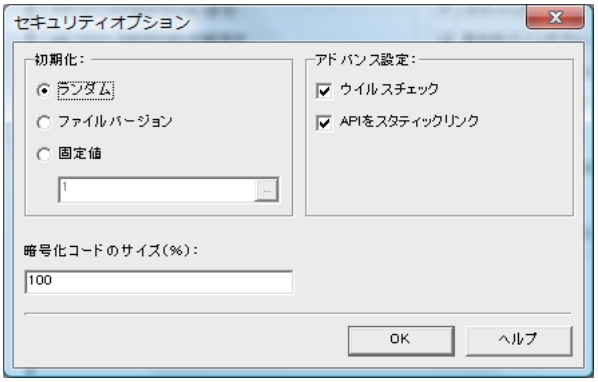
□一般的なデバッガ検知
デバッガがアプリケーションにアタッチされないようなメカニズムを追加します。

□仮想マシン検知
アプリケーションがバーチャルマシン上で起動できないようにします。

□デバッガを検知したらハードウェアキーをロック
デバッカの動作を検知した場合に、ワイブキー（ハードウェア）自身をロックし、アプリケーションが起動できないようにします。ワイブキー（ハードウェア）のロックを解除するには、ライセンサーによるリモートプログラミング（更新ファイル）が必要になります。

アドバンスセキュリティオプション

「アドバンス」ボタンをクリックするとアドバンスセキュリティオプション画面が開きます。



初期化：

ファイルを暗号化する暗号キーの生成方法を設定します。

○ ランダム

暗号キーをランダムに生成します。(デフォルト)

○ ファイルバージョン

暗号キーをファイルバージョン情報に基づき生成します。

○ 固定値

暗号キーを固定化します。固定値を設定して暗号化処理を行うと、常に同じ暗号化結果が得られます。(同一のアプリケーションに対して、暗号化処理を繰り返しても同じバイナリデータになる。File Compareで比較すると一致する。)

[NOTE]

基本的に、暗号キーはファームコードとユーザーコードを要素に生成されます。ここでは、ファームコードとユーザーコードを要素に、暗号キーをどのように生成するか、例えば、ランダムに生成するか、ファイルバージョンを基に生成するか、固定値で生成するかを決定します。特に必要がなければ、「ランダム」で生成することをお勧め致します。

アドバンス設定：

□ ウイルスチェック

実行ファイル感染型のウイルスに対するチェック機能を追加します。プログラムがウイルス感染した場合、または感染したと疑われる場合に警告メッセージを出します。

□ APIをスタティックリンク

APIをスタティックリンクします。ファイルサイズは大きくなりますが、セキュリティ強度は増します。

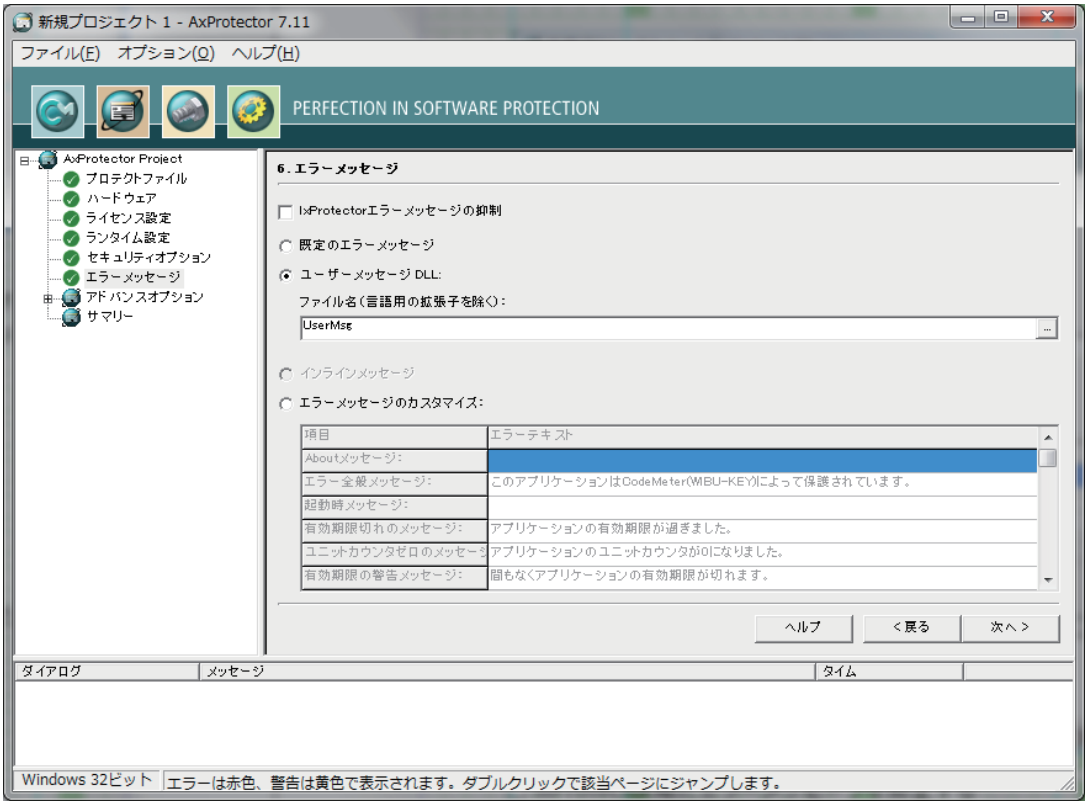
暗号化コードのサイズ(%):

暗号化するコードサイズを%にて設定します。

6. エラーメッセージ

エラーメッセージを作成します。エラーメッセージの作成には４通りの方法があります。

- 既定のエラーメッセージ（英語デフォルト）
- ユーザーメッセージDLL（UserMsgUs.dllとUserMsgJa.iniファイルを使用）
- インラインメッセージ（.NET アセンブリ用）
- エラーメッセージのカスタマイズ（フォームから直接メッセージを入力）



□ IxProtector用にエラーメッセージを抑止する

IxProtector/WUPIを使用した場合、IxProtector/WUPI側で何かエラーが発生するとエラーメッセージが表示されます。このオプションを選択すると、IxProtector/WUPIによるエラーメッセージを表示させないようにできます。

○ 既定のエラーメッセージ

ワイブキーがあらかじめ用意しているエラーメッセージです。メッセージは英語で表示されます。

○ ユーザーメッセージ DLL :

BMP画像を含めたメッセージをカスタマイズすることができます。カスタマイズ作業は“UserMsgJa.ini” ファイルの内容を直接編集します。このオプションを選択して暗号化処理を行うと、暗号化されたプログラムと同じフォルダに"UserMsgJa.ini"が作成されます。この"UserMsgJa.ini"をメモ帳などのエディタで編集してください。編集した“UserMsgJa.ini” やBMP画像ファイルは暗号化されたプログラムと同じフォルダに保存する必要があります。該当するファイルが見つからなかった場合は、ワイブキーのデフォルトメッセージ（英語）が表示されます。

【UserMsgJa.iniファイルの説明】

[Main]

BuyUrl：WebサイトのURLを設定します。

Logo：左部のメッセージ画面に表示されるBMP画像ファイルを指定します。(BMPのみ有効)

Caption：メッセージ画面のタイトルを設定します。

MainText：エラーメッセージ本文を入力します。改行は￥nで行います。

BuyText：購入サポート窓口情報などを入力します。

HeadLine：エラーメッセージのヘッドラインを設定します。

Okbutton：メッセージ画面の［OK］ボタンの名前を設定します。

Cancelbutton：メッセージ画面の［キャンセル］ボタンの名前を設定します。

Retrybutton：メッセージ画面の［再試行］ボタンの名前を設定します。

Ignorebutton：メッセージ画面の［無視］ボタンの名前を設定します。

BuyNowbutton：メッセージ画面の［購入］（HPへのリンク）ボタンの名前を設定します。

BuyHint：

BuyHint=on　BuyTextメッセージを表示

Buyhint=off　BuyTextメッセージを非表示

UnitCounterMax = プログレスバーに表示するユニットカウンタ最大数（例：1000）

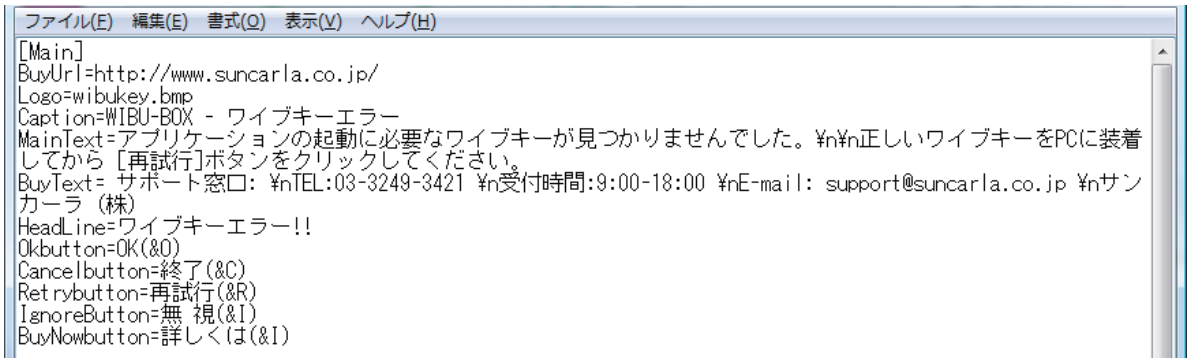
DaysMax = プログレスバーに表示する有効日数最大数（例：100）

UnitCounterText = ユニットカウンタのタイトル（例：回数:）

ExpirationDateText = 有効期限のタイトル（例：日数:）

【操作例】

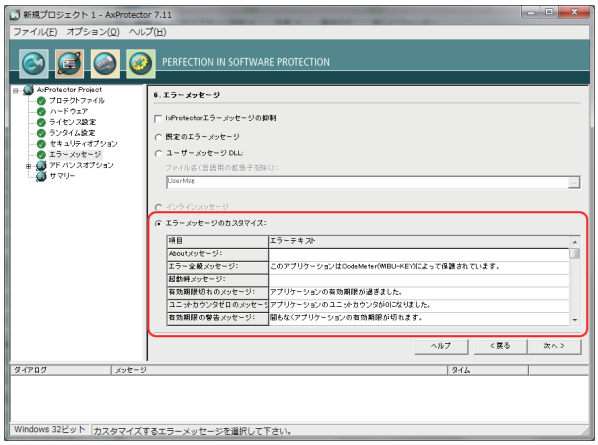
1.「UserMsgJa.ini」ファイルをテキストエディタで開き、必要に応じてメッセージを変更します。



2. 暗号化されたプログラムと同じフォルダに、"UserMsgUs.dll" と "UserMsgJa.ini"および使用した画像ファイル(BMP)をおき、ワイブキーを装着しない状態でプログラムを実行するとエラーメッセージが表示されます。

3. 暗号化されたプログラムを配布する際は、必ず"UserMsgUs.dll" と "UserMsgJa.ini"も一緒に配布し、暗号化されたプログラムと同じフォルダに保存するようにしてください。もし、"UserMsgUs.dll" と "UserMsgJa.ini"が存在しない場合は、ワイブキーのデフォルトメッセージ(英語)が表示されます。

○ エラーメッセージのカスタマイズ：
ワイブキーにデフォルトで用意されているメッセージボックスのメッセージ部分を直接編集して、貴社専用のエラーメッセージにカスタマイズします。



About メッセージ
画面には表示されません。

エラー全般メッセージ
アプリケーションの起動や、ランタイムチェックの際、ワイブキーが見つからない場合に表示されるメッセージです。

起動時メッセージ
アプリケーション起動時に表示されるメッセージです。

有効期限切れのメッセージ
ワイブキーに設定された有効期限(Expiration Time)が過ぎた時に表示するメッセージです。

ユニットカウンタゼロのメッセージ
ワイブキーに設定されたリミットカウンタ（=ユニットカウンタ）が0の時に表示するメッセージです。

有効期限の警告メッセージ
ワイブキーに設定された有効期限(Expiration Time)が「警告メッセージを表示する開始点」（「４. ランタイム設定」画面で設定する）に達した時に表示する警告するメッセージです。アプリケーションの起動時に表示されます。この警告メッセージは、有効期限（Expiration Time)が過ぎるまでアプリケーション起動時に常に表示されます。

ユニットカウンタの警告メッセージ
ワイブキーに設定されたリミットカウンタ（=ユニットカウンタ）が「警告メッセージを表示する開始点」（「４. ランタイム設定」画面で設定する）に達した時に表示する警告するメッセージです。アプリケーションの起動時に表示されます。この警告メッセージは、リミットカウンタ（=ユニットカウンタ）が0になるまでアプリケーション起動時に常に表示されます。

7. アドバンスオプション

□ 拡張コマンドラインオプション

AxProtector操作画面から選択できない拡張機能をコマンドラインでサポートします。

□ IxProtectorを有効にする

IxProtectorを使用する場合は、チェックをいれます。

IxProtectorは、メモリー上で展開されるコードを常に暗号化しておき、必要な時に必要なモジュールを復号し、実行したあとは再び暗号化しておく、メモリー上での「オンデマンド復号」を実現する機能です。AxProtectorで暗号化されたコードが、メモリー上でも常に暗号化されているため、クラッキングに対して非常に強力なセキュリティを実現できます。

[NOTE]
IxProtectorは、WUPI(Wibu Universal Protection Interface)ファンクションと一緒に使用します。
C#やVB.NETなどで作成された.NETアセンブリやJavaアプリケーションの場合、IxProtector/WUPIを使用しなくても、AxProtectorのプロテクトタイプ「.NET アセンブリ」または「Java アプリケーション」で暗号化するだけで自動的に「オンデマンド復号」機能が付加されます。

□ ファイル暗号化を有効にする

データファイルを暗号化します。暗号化されたデータファイルは、メモリー上で自動的に復号されます。

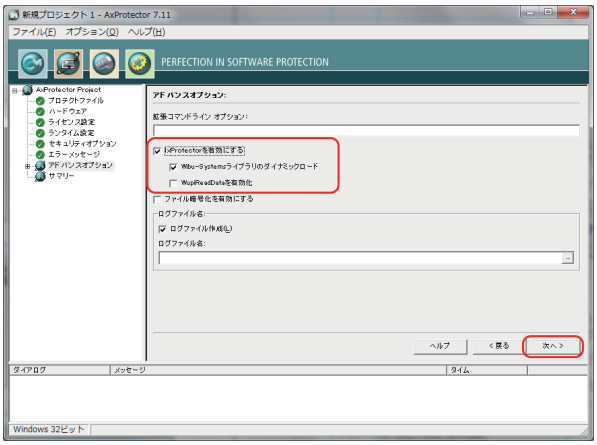
□ ログファイル作成

暗号化処理のログファイルを作成します。

7-1. IxProtectorの使い方

① IxProtector を有効にする

アドバンスオプション画面で、「IxProtectorを有効にする」にチェックを入れ、「次へ」をクリックする。



☐ Wibu-Systemsライブラリのダイナミックロード

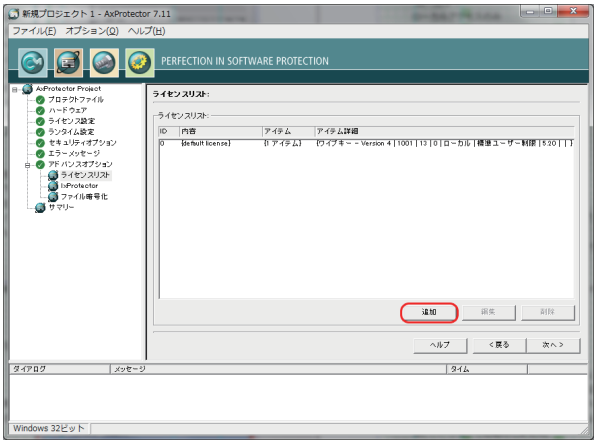
VB6でコンパイルしたアプリケーションの場合、またはWibu-Systemsライブラリをダイナミックでロードする場合は、このオプションにチェックを入れます。

☐ WupiReadDataを有効化

ワイブキーから保存データを読み込む場合は、このオプションにチェックを入れます。

② ライセンスリストを作成する

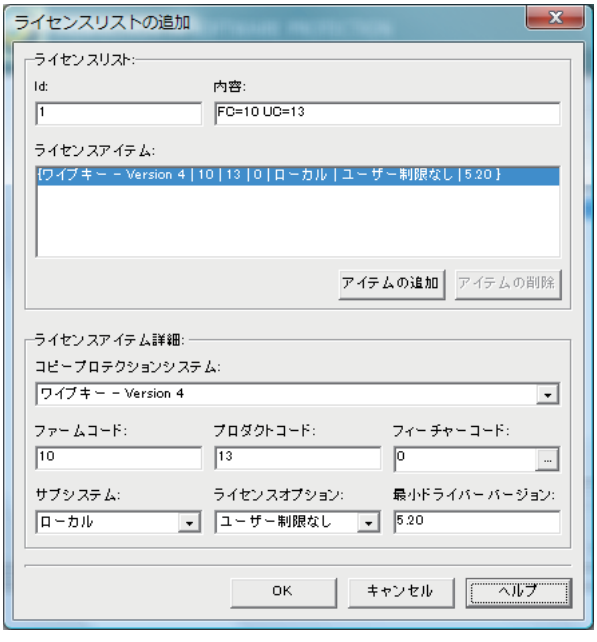
ライセンスリスト画面が表示されますので、「追加」ボタンをクリックします。ライセンスリストとは、使用するハードウェアキー、ファームコード、ユーザーコード（プロダクトコード）、サブシステム、ローカルオプションなどの情報を 1 つのライセンスとしてId登録し、そのライセンスの集合体を指します。"IxProtector"および"ファイル暗号化"（後述）の場合、ライセンスリストに登録されているライセンスを使用し、それらのライセンスはIdによって呼び出します。



暗号化されているモジュールをメモリー上で復号化する前に、指定したライセンスIdでハードウェアキーのチェックを行い、キーチェックに成功した場合のみモジュールの復号化を行います。キーチェックに失敗するとモジュールの復号化は行われません。（その結果、モジュールは実行されません）

③ ライセンス内容を登録する

「ライセンスの追加」画面でライセンス内容を登録します。



Id:

ライセンスIDを設定する。ライセンスIDは連番で登録します。

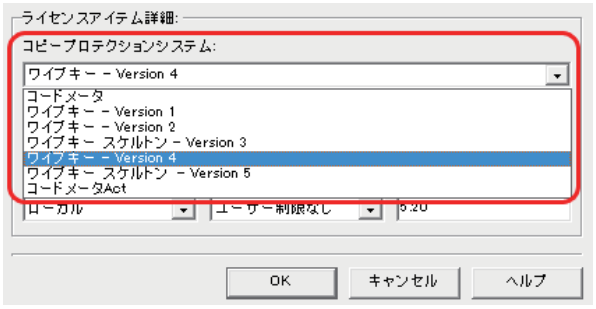
内容:

ライセンス内容を記載する。この内容は、実際の暗号化処理には反映されませんので、セキュリティ内容や商品名など、自由に記載してください。

コピープロテクションシステム:

使用するハードウェアキーを指定する。プロフェッシ

ョナル版ワイブキーは "ワイブキー-Version 4"を、スケルトン版ワイブキーは "ワイブキー スケルトン -Version 5"を選択します。

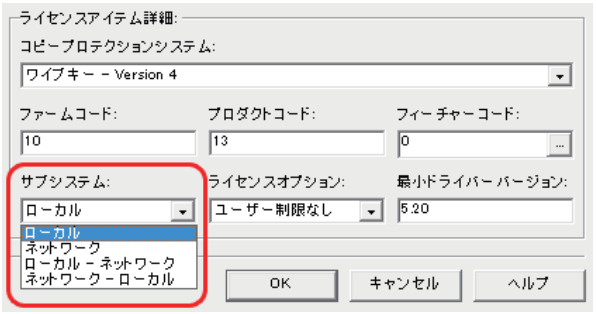


ファームコード:
ライセンスIDに割り当てるファームコードを設定。

プロダクトコード:(ユーザーコード)
ライセンスIDに割り当てるユーザーコードを設定。(CodeMeterの場合はプロダクトコード)

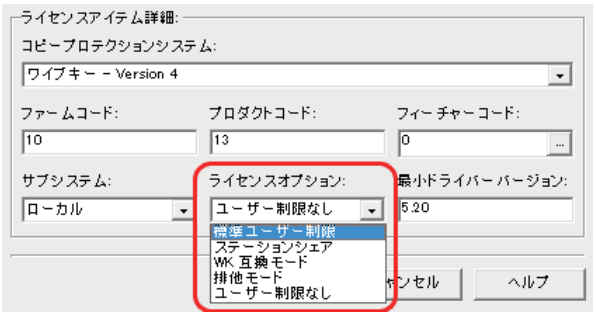
フィーチャーコード:
CodeMeterのみ(ワイブキーでは使用しない)

サブシステム:
ワイブキーのサブシステムを割り当てます。



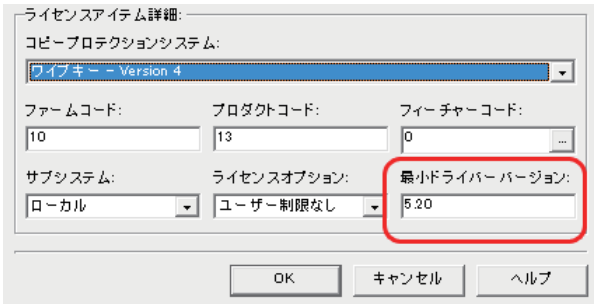
「ローカル」:
ローカルアクセスのみ。ローカルポートだけを検索します。
「ネットワーク」:
ネットワークアクセスのみ。ワイブキーサーバー上のワイブキーだけを検索します。
「ローカル - ネットワーク」:
最初にローカルアクセスを行い、次にネットワークアクセスを行います。
「ネットワーク - ローカル」:
最初にネットワークアクセスを行い、次にローカルアクセスを行います。

ライセンスオプション:
ライセンス形態を割り当てます。

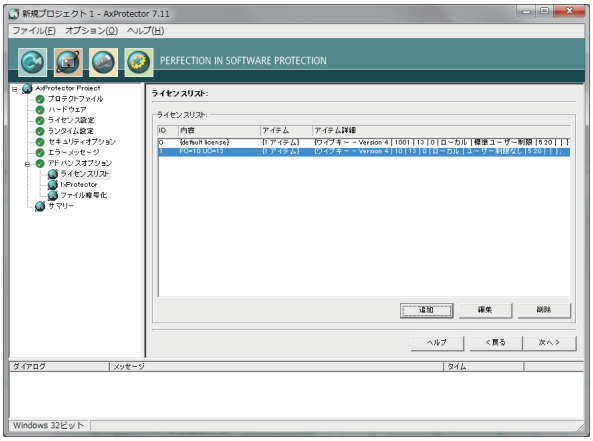


「標準ユーザー制限」:
アプリケーション単位で制限(標準制限)
「ステーションシェア」:
クライアント単位で制限(ステーションシェア)
「WK 互換モード」:
簡易モード(ネットワークでは標準制限、ローカルは制限なし)
「排他モード」:
クライアントごとに 1 つのアプリケーションのみ許可。
「ユーザー制限なし」:
アクセス制限なし

最小ドライバーバージョン:
ワイブキーを動作させるための最低バージョンを指定します。暗号化したプログラムは、ここで指定したワイブキーランタイムキットバージョン（ドライバーバージョン）以降で動作し、指定したバージョンより古いバージョンでは動作しません。

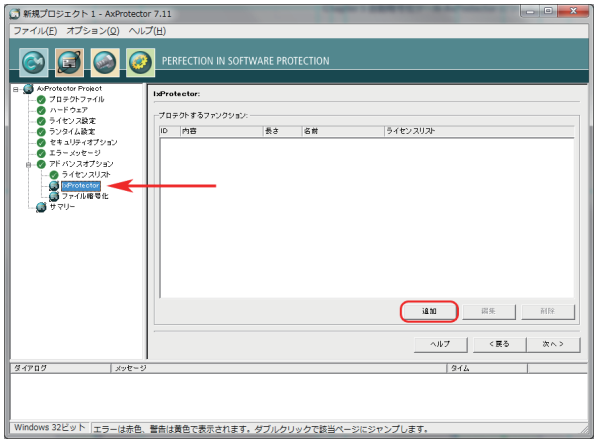


「OK」ボタンをクリックするとライセンス内容がライセンスリストに追加登録されます。



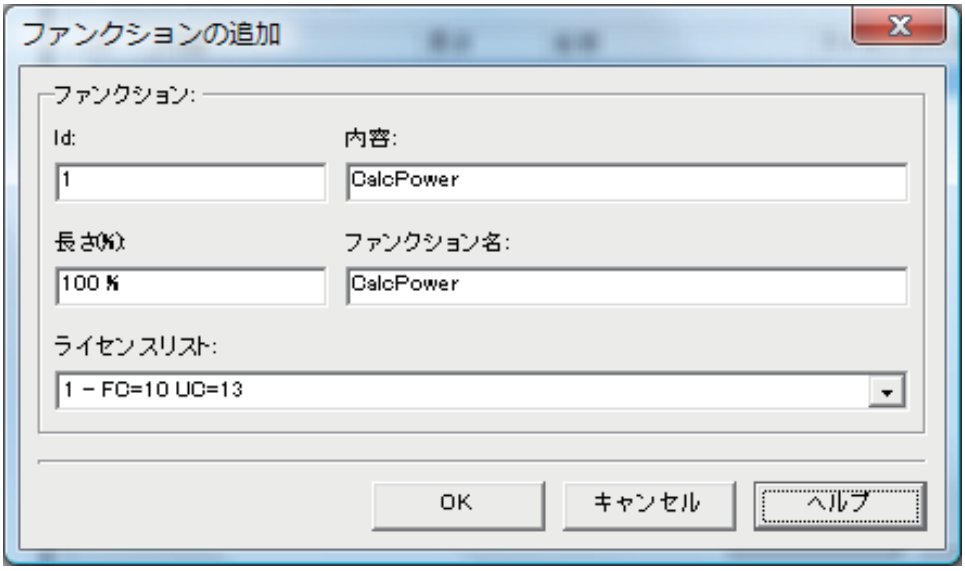
④ 暗号化するファンクションを登録する

次に、暗号化するファンクション（モジュール）を登録します。「IxProtector」画面で、「追加」ボタンをクリックして、暗号化するファンクション名を登録します。



⑤ ファンクション名を登録する

「ファンクションの追加」画面で暗号化するファンクションを登録します。



Id:
ファンクションIDを登録します。ファンクションIDは自動的に連番登録されます。

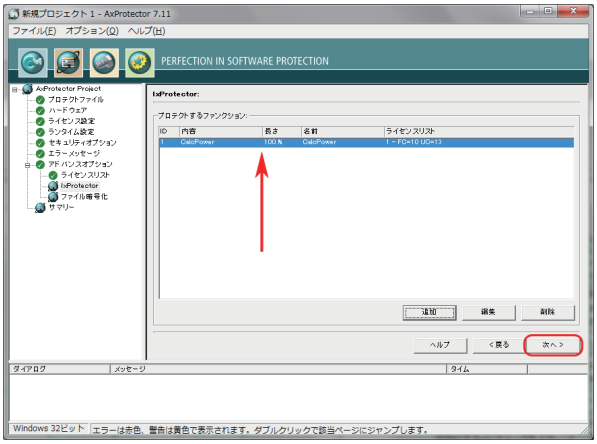
内容:
ファンクションの内容を登録します。この内容情報は実際の暗号化処理には適用されませんので、自由に記述してください。

長さ (%):
暗号化するファンクションモジュール範囲をパーセンテージ(%)で指定します。%を使用しないで、直接数値を設定すると、設定したバイト数分が暗号化されます。

ファンクション名:
ソースコードの中で実際に使われているファンクション名を正確に入力します。
(例)
CWupiCalulatorDlg::CalcSimpleOperation
OnButtonCalcMemClear
CWupiCalculatorDlg::CalcAngle

ライセンスリスト:
使用するライセンスをライセンスリストから指定します。

登録後、「OK」ボタンをクリックすると、IxProtectorに暗号化するファンクションが登録されます。「次へ」ボタンをクリックして次に進めます。

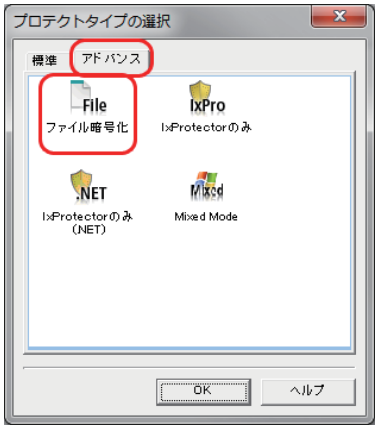


7-2. ファイル暗号化の使い方

この「ファイル暗号化を有効にする」オプションは、プログラムが使用（または制御）するデータファイルをメモリー上で復号化・暗号化する機能です。復号化・暗号化はあくまでメモリー上で行われ、ディスク上のデータファイルは常に暗号化されているため、データファイル自身のコピープロテクトが可能です。

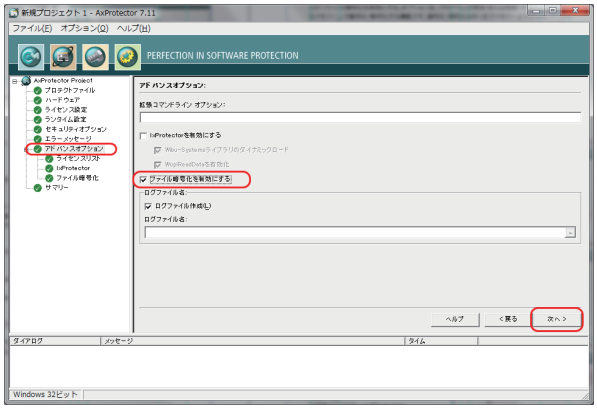
なお、データファイル自身はAxProtectorの"ファイル暗号化"で最初に暗号化しておきます。

この「ファイル暗号化を有効にする」オプションは、ここで暗号化するプログラムに「データファイルの復号化・暗号化を行う機能」を持たせるかどうかの選択をするオプションです。このオプションを有効にしない場合、データファイルの復号化・暗号化は行われません。



① ファイル暗号化を有効にする

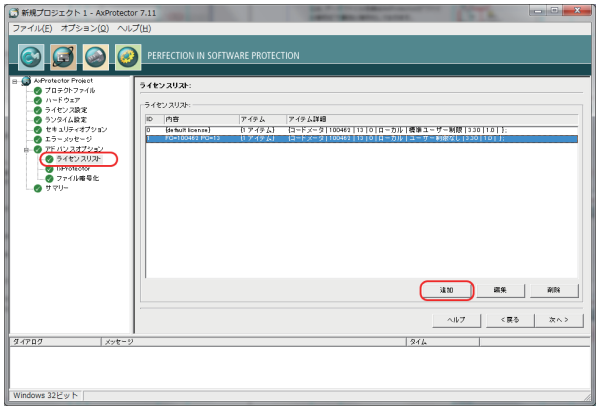
「アドバンスオプション」画面で、「ファイル暗号化を有効にする」にチェックを入れ、「次へ」をクリックする。



② ライセンスリストを作成する

「ライセンスリスト」画面が表示されますので、「追加」ボタンをクリックします。すでに、ライセンスリストにライセンスが登録されている場合は不要です。

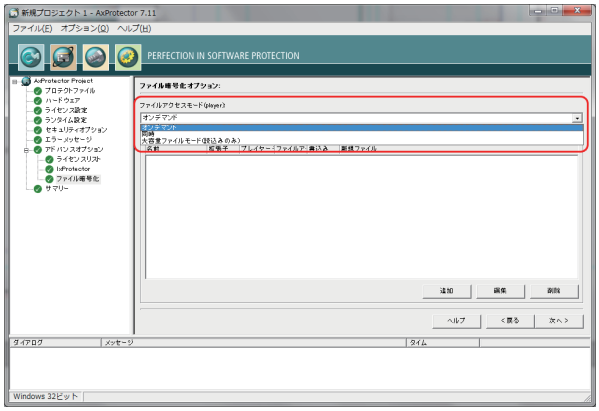
ライセンスリストに登録されているライセンスは、IxProtectorとファイル暗号化の両方で使用可能です。IxProtectorを使用しない場合は、IxProtector画面がグレーアウトしますので、「次へ」ボタンをクリックして次に進めます。



③ ファイルアクセスモードを指定する

「ファイル暗号化オプション」画面で、ファイルアクセスモード（Player)を指定します。ファイルアクセスモードには次の３種類があります。

オンデマンド
同時（一括）
大容量ファイルモード（読込みのみ）

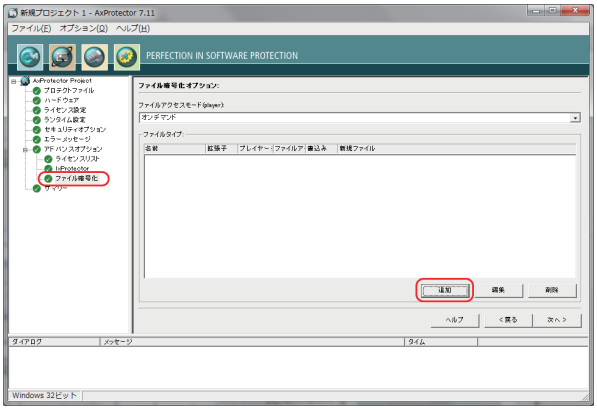


オンデマンド:
「オンデマンド」モードでは、プログラム（Player)は、ファイル全体のメモリーを確保しますが、読み込むのは必要な部分だけです。読み込みと復号化は4kバイトブロック単位で行い、必要な時に必要なブロックをロードし復号化します。一度復号化されたブロックは、メモリー上に残るため、2回目以降はメモリー上のブロックが使用されます。この「オンデマンド」モードはロードされるファイル全体のメモリーが必要になりますが、一度復号化されたデータを再利用できるため、パフォーマンスに優れます。このモードは、データファイルを読み書き(Read/Write)する場合に適します。

同時（一括）:
「同時（一括）」モードでは、プログラム（Player)は、ファイル全体のメモリーを確保し、ファイル全体をメモリーに読み込んでから一度に復号化します。復号化されたデータはメモリー上に残り、必要な都度メモリー上から読み出されます。ファイル全体のメモリーが必要になりますが、復号化済のデータを利用できるためパフォーマンスに優れます。「オンデマンド」モードと異なる点は、最初のアクセス時にファイル全体を復号化するための時間が必要となることです。このモードは、「オンデマンド」モード同様、データファイルを読み書き（Read/Write)する場合に適します。

大容量ファイルモード（読込みのみ）:
この「大容量ファイルモード（読込みのみ）」モードでは、ファイル全体のメモリーを確保せず、必要な部分を読んで復号化します。復号化されたデータはメモリーに残らないため、メモリーの消費量を抑えることができます。このモードは、リードオンリー（読込みのみ）のデータファイルに適します。

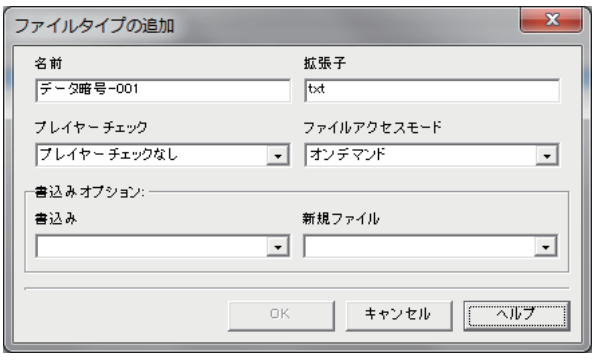
④ ファイルタイプの定義を行う



「ファイル暗号化オプション」画面で、「追加」ボタンをクリックし、ファイルタイプの追加を行います。

名前:
ファイルタイプ名を定義します。この名前は暗号化処理には影響しません。単なる参照名です。

拡張子:
作成するファイルの拡張子を指定します。ここで指定する拡張子を持つデータファイルだけが復号化・暗号化の対象になります。
（例）txt



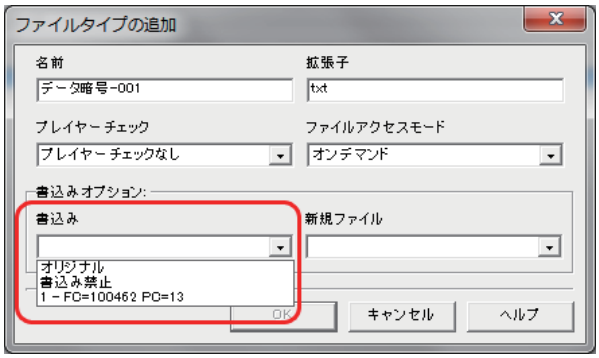
プレイヤーチェック:
プログラム（プレイヤー）自身のチェックを指定します。特にプレイヤーのチェックを行わない場合は、「プレイヤーチェックなし」を選択します。

ファイルアクセスモード
「オンデマンド」、「同時（一括）」、「大容量ファイルモード（読込みのみ）」から選択します。

書込みオプション:

書込み:

暗号化されていたファイルをプレイヤーで開いた後に編集した場合、どのようなセキュリティ属性で保存するかを指定できます。セキュリティ属性には、「オリジナル」、「書込み禁止」、「(ライセンスリスト)」の３通りがあります。



オリジナル
あらかじめ設定されていたライセンス仕様(ファームコード、ユーザーコードなど)にて保存する。

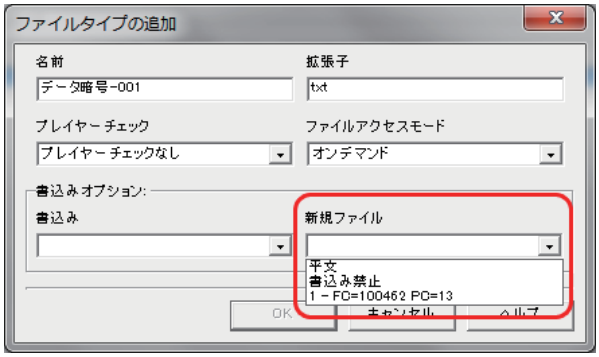
書込み禁止
ファイルへの書込みを禁止する。
(編集保存禁止)

(ライセンスリスト)
指定したライセンス仕様で保存する。ライセンスリストで追加されたライセンスが表示されます。

新規ファイル:

プレイヤーで新しく作成する新規ファイルのセキュリティ属性を指定します。

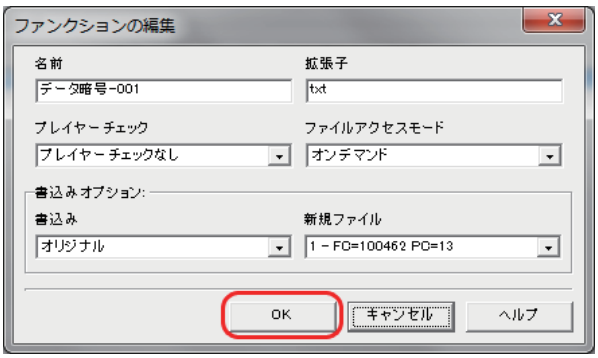
平文
新規ファイルは平文(暗号化されない)として保存する。



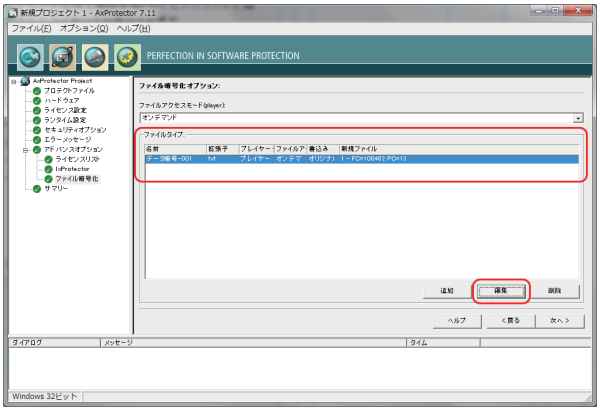
書込み禁止
新規ファイルは作成しない。新規保存できない。

(ライセンスリスト)
指定したライセンス仕様で作成保存する。ライセンスリストで追加されたライセンスが表示されます。

ファイルタイプの追加が終了したら「ＯＫ」ボタンをクリックして、「ファイル暗号化オプション」画面に戻ります。追加したファイルタイプが登録されているのを確認してください。ここで登録されたデータファイルがプログラム(プレイヤー)によってセキュリティ制御(復号化・暗号化)されます。

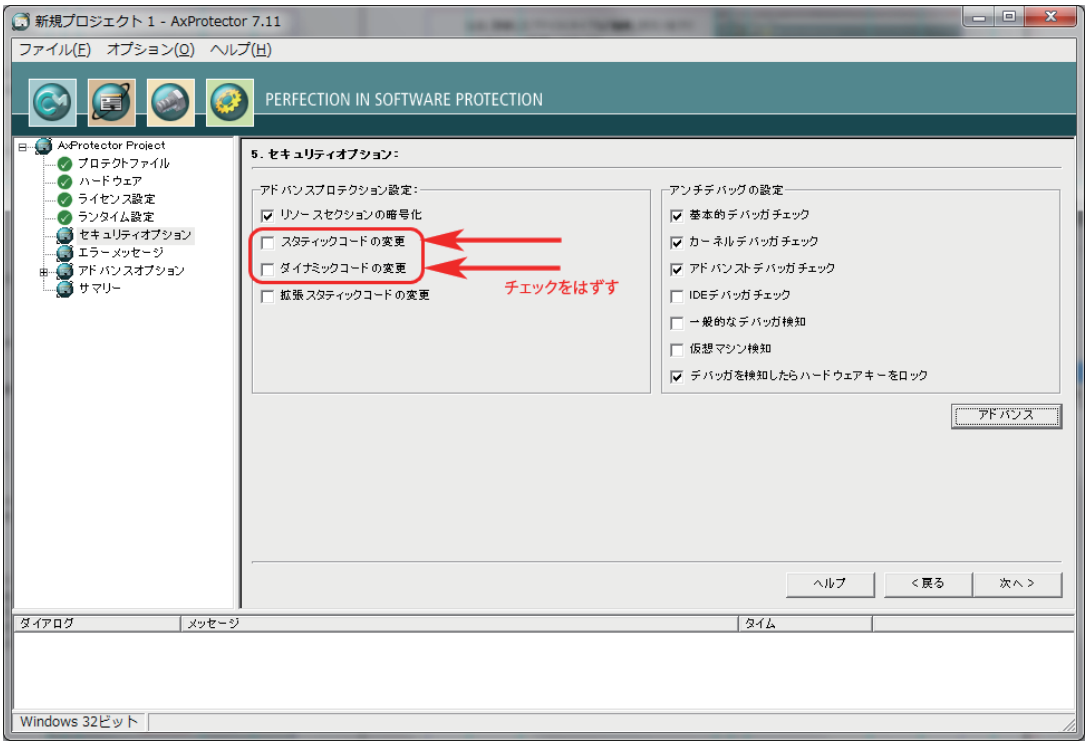


なお、登録したファイルタイプは「編集」ボタンをクリックすることで編集が可能です。



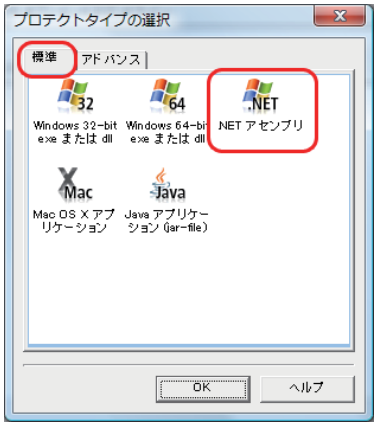
[注意]

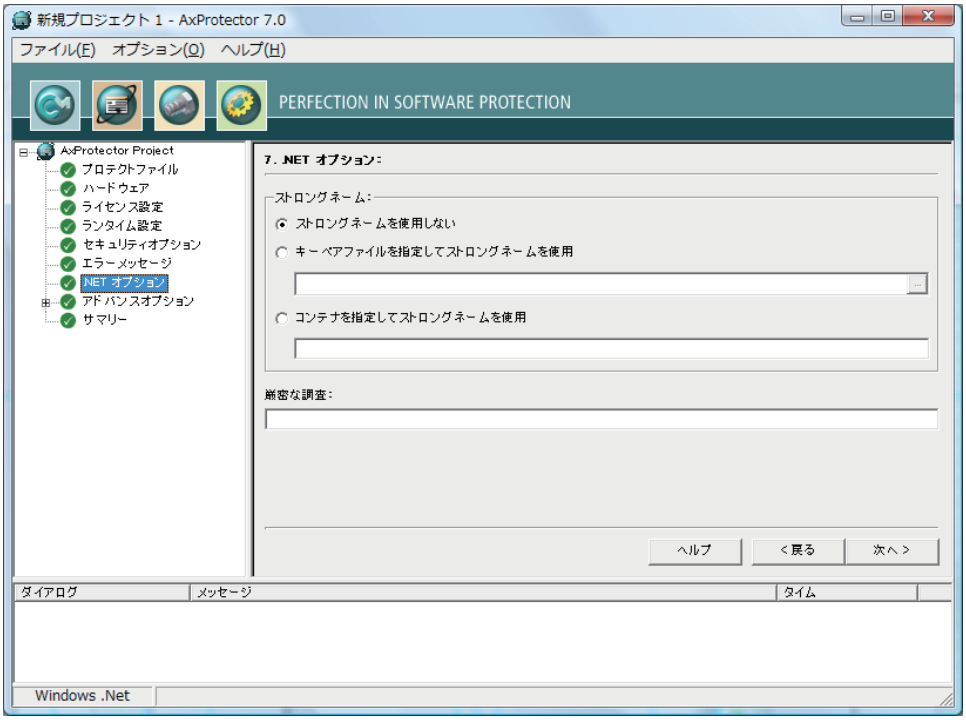
制御するプログラムをAxProtectorで暗号化する際、「５．セキュリティオプション」で、「スタティックコードの変更」と「ダイナミックコードの変更」のオプションをはずす必要があります。デフォルトではチェックが入っています。チェックを入れたまま暗号化処理を行うと処理の途中でエラーが発生しますのでご注意ください。



7a. .NET オプション：（.NET アセンブリの場合）

AxProtector起動時に、「プロテクトタイプの選択」画面で「.NETアセンブリ」を選択した場合、この「.NETオプション」画面が表示されます。





ストロングネーム：

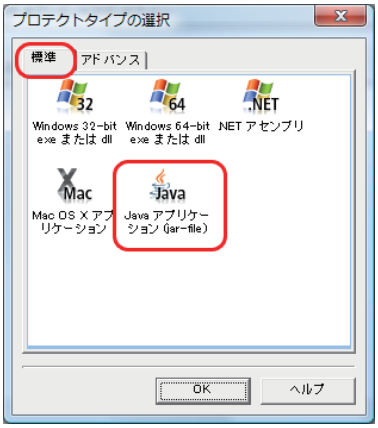
○ ストロングネームを使用しない
ストロングネームを使用しない場合に選択します。

○ キーペアファイルを指定してストロングネームを使用
ストロングネームをキーファイルから使用する場合にチェックし、任意のキーファイルの保存先を指定します。

○ コンテナを指定してストロングネームを使用
ストロングネームをCSPコンテナから使用する場合にチェックし、任意のキーコンテナ名を指定します。

7b. Java オプション（Java アプリケーションの場合）

AxProtector起動時に、「プロテクトタイプの選択」画面で「Java アプリケーション(jar-file)」を選択した場合、この「Javaオプション」画面が表示されます。



Javaランタイム：
使用するJavaランタイムを選択することができます。

オプション：
メインクラス：
起動するMain Classを指定することができます。マニュフェストでMain-Classを指定していない場合、必ずこのオプションでMain-Classを指定してください。

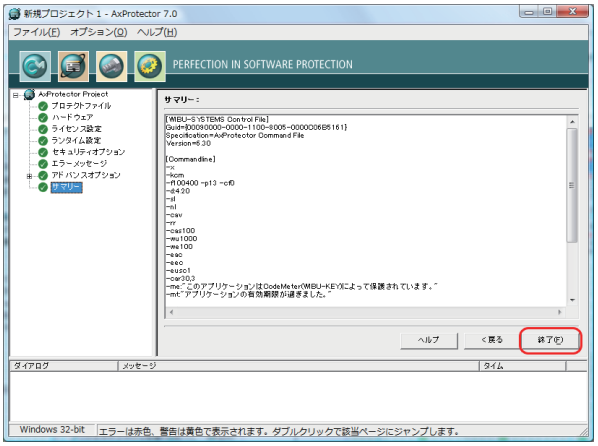
パラメータ：
Main-Classのための引数を指定することができます。

最低限必要とされるJavaバージョン番号：
使用するJava Versionの下限を定義することができます。

☐ System.exit()関数の呼び出しでアプリケーションを終了
System.exit()関数を呼び出すことにより、アプリケーションを終了します。

8. サマリー：

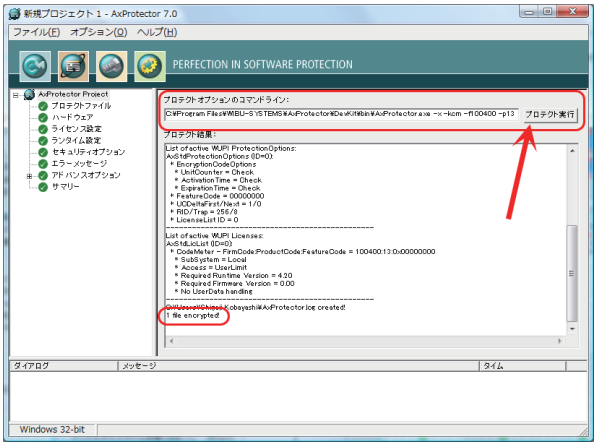
設定したセキュリティ内容の概要を表示します。
設定内容は、コマンドラインによるオプションパラメータで表示されます。



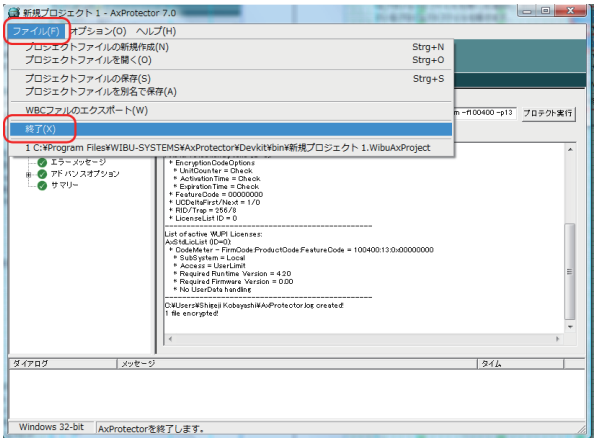
設定に問題がなければ、右下の「終了」ボタンがアクティブになります。もし、設定に問題がある場合は、「終了」ボタンが非アクティブの状態になります。

「終了」ボタンをクリックすると、暗号化処理が開始されます。

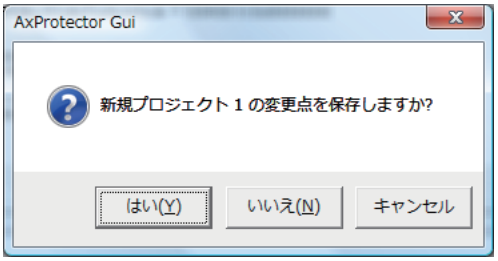
暗号化処理に成功すると、「プロテクト結果」画面の最下行に" 1 file encrypted! "が表示されます。



[ファイル(F)]メニューから[終了(X)]を選択してAxProtectorを閉じます。



閉じる際、プロジェクトファイルを保存するかの画面が表示されます。プロジェクトファイルとは、各項目で設定したセキュリティ内容を保存しておくもので、どのファイルをどのようなセキュリティ内容で暗号化したかの管理ができます。また、プロジェクトファイルを開くことにより、最初から各項目を入力する手間が省けます。

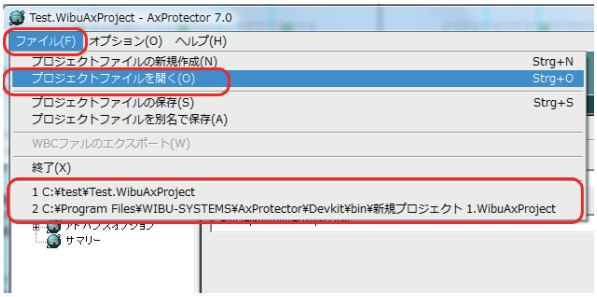


9. プロジェクトファイルを開く

AxProtectorを起動したあと、[ファイル]メニューから[プロジェクトファイルを開く(O)]を選択し、保存されているプロジェクトファイルを開きます。

プロジェクトファイルに設定されている内容が反映されます。

また、直近に使用したファイルが最下行に表示されますので、そこからファイルを開くこともできます。



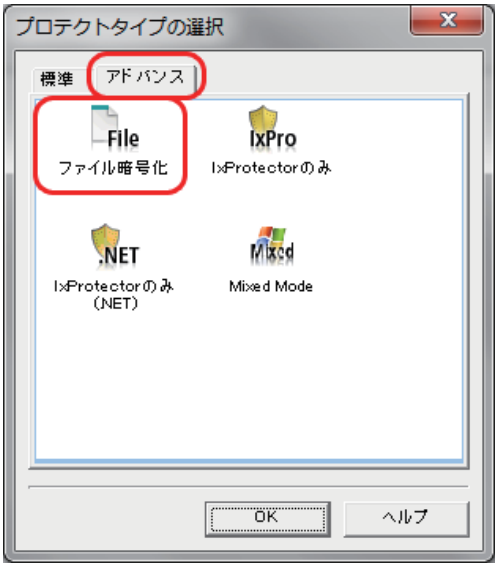
5-5. データファイルを暗号化する

データファイルを暗号化する場合は、AxProtectorの"ファイル暗号化"機能を使用します。AxProtector起動後、「プロテクトタイプの選択」で「アドバンス」タブを選択し、「ファイル暗号化」を選択して"OK"ボタンをクリックするか、「ファイル暗号化」のアイコンをダブルクリックします。

この「ファイル暗号化」機能で暗号化可能なファイルは、txtなどのテキストファイル、Flash, WMV, MPEGなどの動画ファイル、その他のドキュメントファイルです。

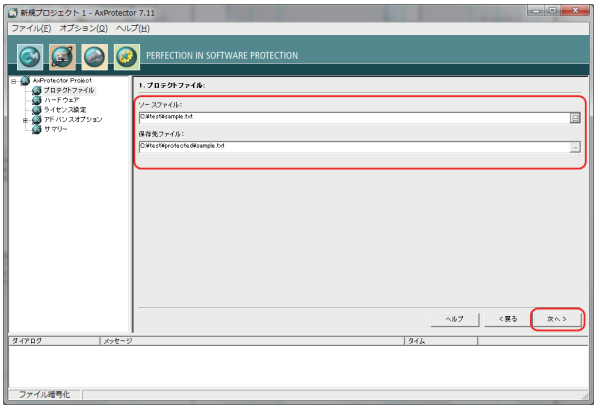
ただし、暗号化されたファイルを開くプログラム(exe等)にあらかじめAxProtectorで暗号化処理する必要がありますので、Excel.exeやMedia Player等、ライセンス上暗号化処理ができない場合は例外になります。

あくまで、貴社のプログラムから利用するデータファイルが対象になります。



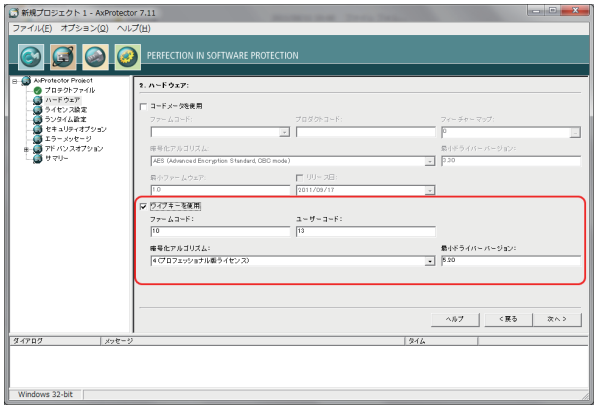
① ファイル名を入力する

「プロテクトファイル」画面で暗号化するファイル(ソースファイル)と暗号化後に作成されるファイル(保存先ファイル)を指定します。ソースファイルと保存先ファイルがフォルダも含めて同じ場合は上書き保存されますのでご注意ください。



② ファームコード、ユーザーコードを入力する

「ワイブキーを使用」にチェックを入れ、ファームコード、ユーザーコード、暗号化アルゴリズム、最小ドライバーバージョン(ランタイムバージョン)等を設定します。「コードメータを使用」のチェックははずします。

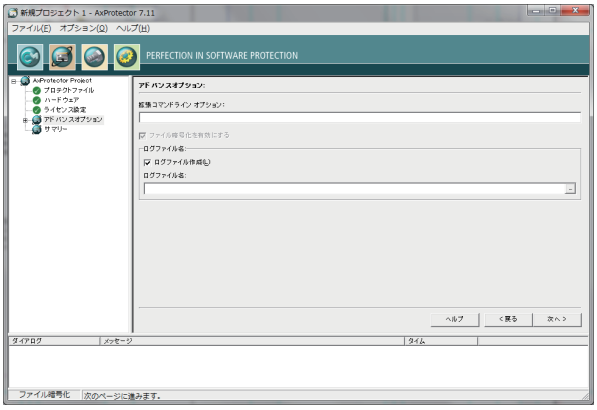


③ ライセンス設定を行う

データファイルの場合は、ネットワーク上のワイブキーをサーチしないため、必ず"ローカル"で使用します。また、ライセンスオプションは、"標準的なユーザー制限（アプリケーション単位で制限）"を選択してください。

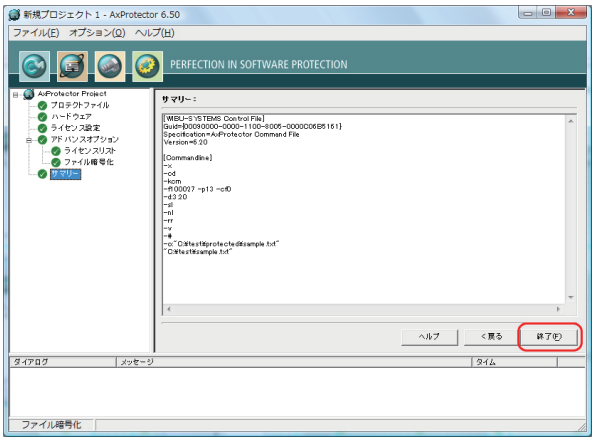
④ アドバンスオプション

アドバンスオプション画面では特に設定する項目はありません。また、次のライセンスリストおよびファイル暗号化オプションでも設定する項目はありません。「次へ」ボタンで進めます。

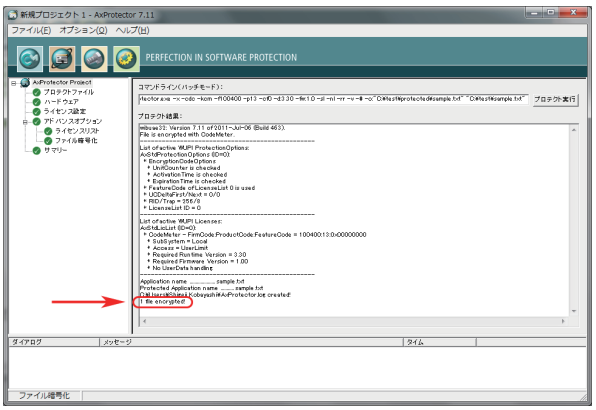


⑤ 暗号化処理を開始する

サマリー画面で「終了(F)」ボタンをクリックすると暗号化処理が開始されます。



サマリー画面で"1 file encrypted!"のメッセージが表示されれば暗号化に成功したことになります。保存先のフォルダに暗号化されたファイルが存在するか確認してください。



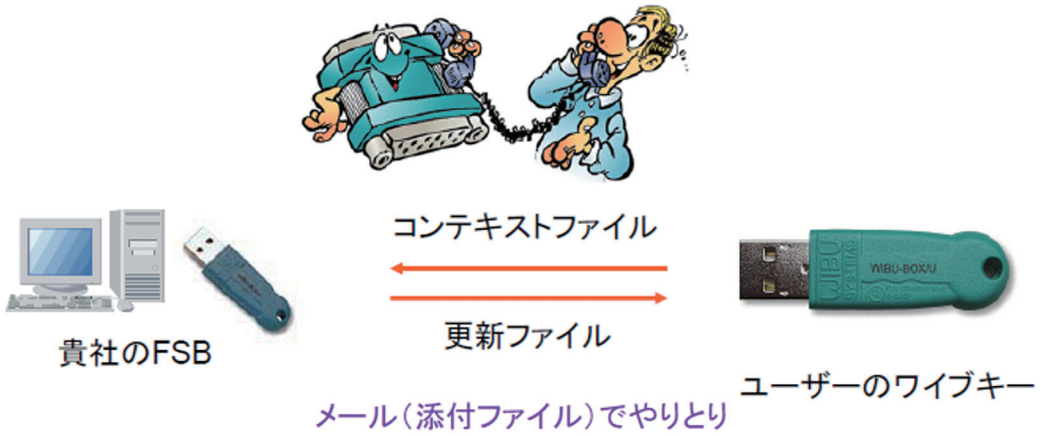
Chapter 6

リモートアップデート機能について

6-1. リモートアップデート機能とは

リモートアップデート機能とは、ユーザー側にあるワイブキーの内容をファイル操作で更新する機能です。基本的に、ワイブキーの内容を変更するには、貴社のFSB(Firm Security Box)を使ってローカルPC上で作業する必要がありますが、一度ユーザー先に配布したワイブキーを変更のつど送り返してもらうことは時間とコストの面からあまり得策ではありません。

ワイブキーのリモートアップデート機能を使うことで、ファイルをメールなどでやりとりすることにより、ユーザー先のワイブキーの内容を更新することが可能になります。商品コード（Firm Code/User Code）の追加/変更、使用期限の更新、回数制限の更新または削除など、必要な時にスピーディに対応が可能になります。



ユーザーに配布した更新ファイルは指定したワイブキーに対して１回だけしか使用できないため、同一のワイブキーに対して２回以上使用したり、別のワイブキーに使用することはできません。確実にセキュリティを確保します。

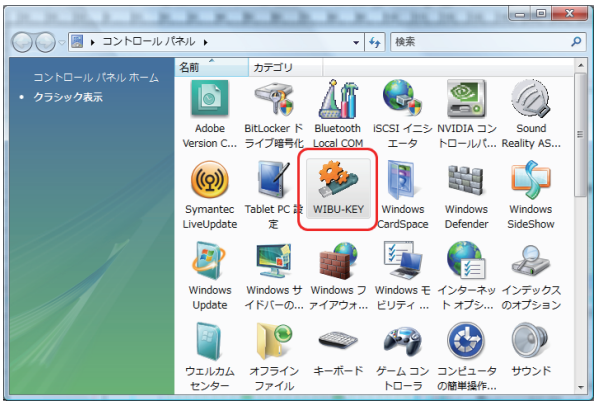
また、更新ファイルを作成するには、貴社のFSBが必ず必要になりますので、第三者が勝手に更新ファイルを作成することはできません。

6-2. コンテキストファイル（RTC ファイル）の作成

まず、ユーザー側にあるワイブキーのコンテキストファイル（RTCファイル）をユーザー側で作成してもらいます。作成方法は簡単です。

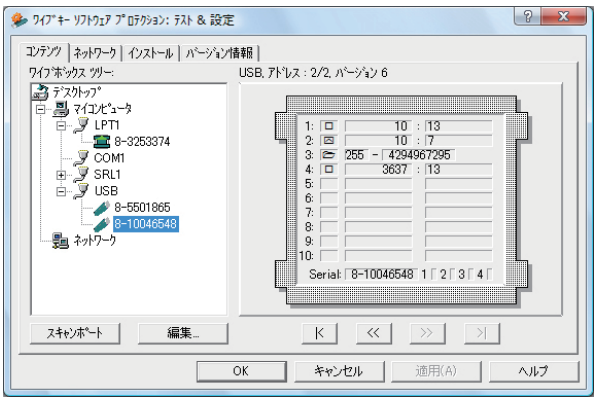
1. ワイブキーアプレットを起動する

ワイブキーが装着された状態で、コントロールパネル上のワイブキーアイコンをダブルクリックしてワイブキーアプレット"WIBUKE32.CPL"を起動します。Windows XPの場合、コントロールパネルをクラシック表示に切り替えるとワイブキーのアイコンが表示されます。Windows Vistaの場合、クラシック表示に切り替えるか、「コントロールパネルホーム」-「ハードウェアとサウンド」をクリックし、画面を下にスクロールします。 「WIBUKE32.CPL」は、起動直後は必要最小限の情報（タブ）だけが表示されます。



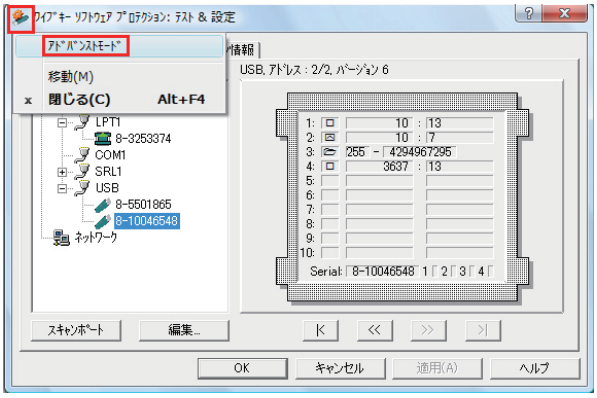
この"WIBUKE32.CPL"は、ユーザー用のサービス・メンテナンスプログラムで、ワイブキーのランタイムキットWkRuntime.exeを実行すると、ランタイムモジュールと一緒にインストールされます。

この"WIBUKE32.CPL"は、ワイブキーの内容を表示するだけで、ユーザー側でワイブキーの内容を編集することはできません。



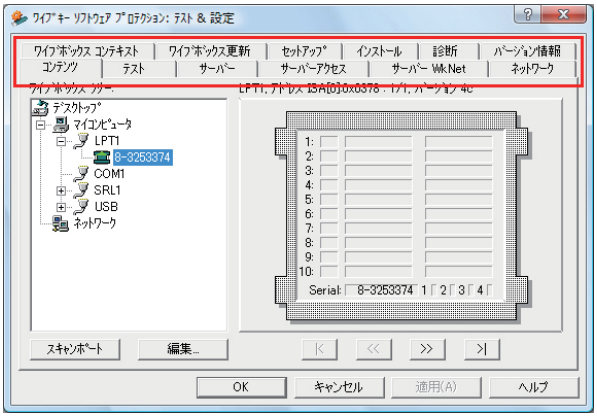
2. アドバンストモードにする

リモートプログラミング機能を有効にするには、画面左上端のアプリケーションアイコンをクリックしてポップアップメニューを表示し、「アドバンストモード」を選択してください。



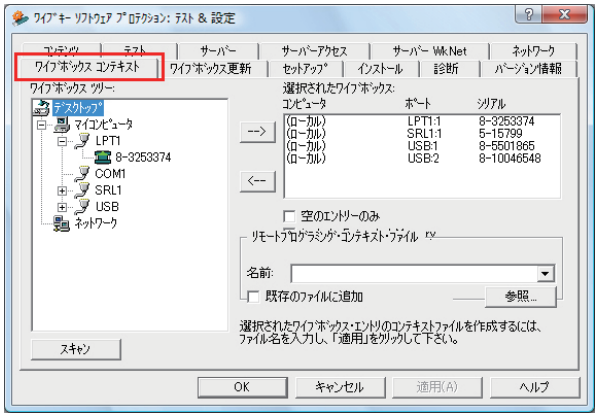
[NOTE]
クリックは、右クリックでなく左クリックになります。

"WIBUKE32.CPL"が「アドバンストモード」で表示されます。



「ワイブボックスコンテキスト」タブページを選択します。

コンテキストファイル（RTCファイル）の対象となるワイブキーが「選択されたワイブボックス」に表示されます。複数のワイブキーを装着している場合、必要のないワイブキーはPCから外してから［スキャン］ボタンで情報を更新するか、または、「選択されたワイブボックス」で不要なワイブキーを選択し［<--］ボタンでリストから除外します。



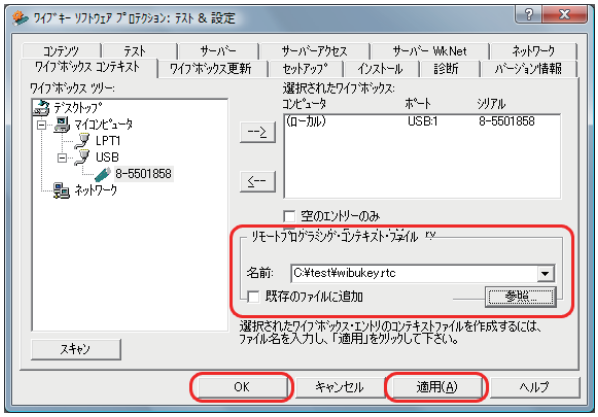
※「選択されたワイブボックス」に複数のワイブキーが表示された状態 でコンテキストファイル（RTCファイル）を作成すると、1 つのコンテキストファイル（RTCファイル）に複数のワイブキー情報が記録されますのでご注意ください。

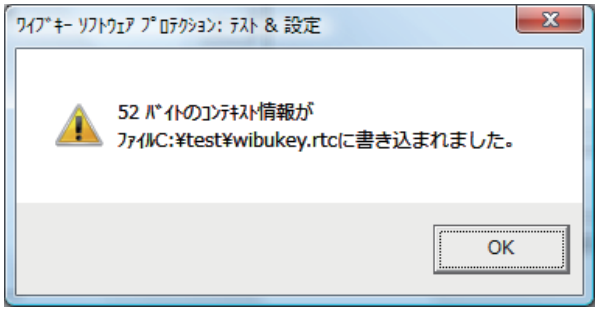
3. コンテキストファイル(RTCファイル)を作成する

コンテキストファイル(RTCファイル)を作成します。［参照］ボタンをクリックし、コンテキストファイル（RTCファイル）を保存するフォルダ、ファイル名を指定します。デフォルトでは、”wibukey.rtc”というファイル名が指定されます。必要に応じてリネームしてください。

［OK］をクリックすると、選択されたワイブキー情報のコンテキストファイル ”wibukey.rtc”が作成されます。

作成されたコンテキストファイル（RTCファイル）をメール添付などでユーザー側から送ってもらいます。

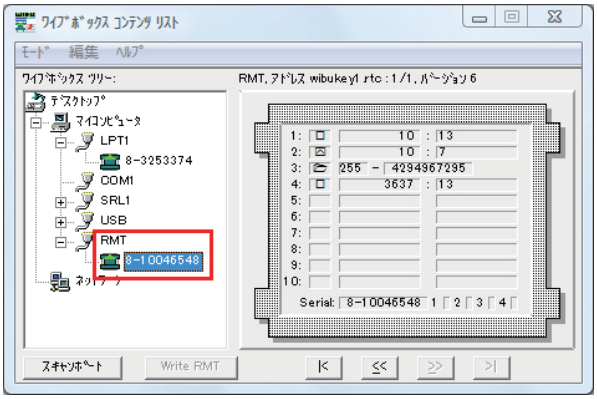




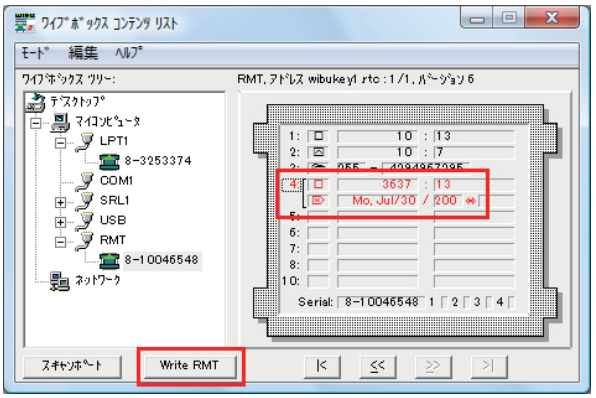
6-3. アップデートファイル（RTU ファイル）の作成

ユーザーから送られてきたコンテキストファイル（RTCファイル）を使って、アップデートファイル(RTUファイル)を貴社にて作成します。

アップデートファイル(RTUファイル）は、コード設定ツール"Wklist32.exe"を使い、貴社のFSBをPCに装着して作成します。* FirmCode＝10（試用版）のアップデートファイル（RTUファイル）を作成する場合、FSBは不要です。

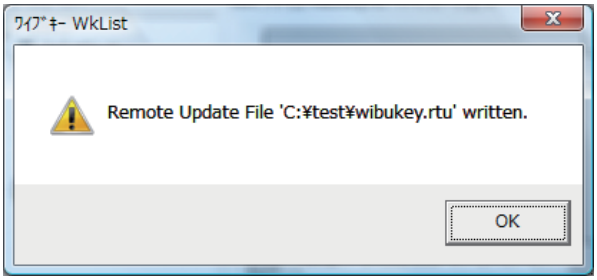


貴社のFSBをPCに装着してから、"Wklist32.exe" を起動します。[編集]メニュー→[Remote Context Files...] からコンテキストファイル（RTCファイル）を読み込みます。右図はFirmCode =10（試用版）のRTCファイルを読み込んだ例です。



読み込まれたアップデートファイル（RTCファイル）は、[RMT] というデバイス名で ワイブボックスツリーに表示されます。エントリを選択して、更新したい内容を追加変更します。

編集を行なうと、変更部分のエントリは赤字で表示され、「Write RMT」ボタンが押せるようになります。[Write RMT] ボタンをクリックするとアップデートファイル(RTUファイル)が作成されます。



アップデートファイルは、コンテキストファイルと同じフォルダに作成され、ファイル名はコンテキストファイルと同じで拡張子が "RTU"になります。

ここで作成されたアップデートファイルをユーザー側にメール添付などで送ります。

6-4. ワイブキーを更新する

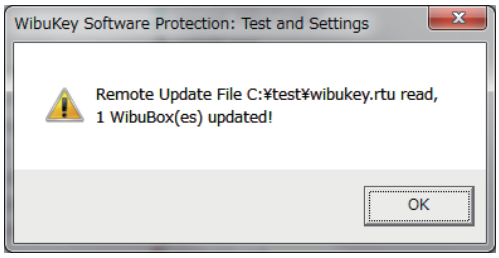
ユーザー側にて、アップデートファイル（RTUファイル）を使ってワイブキーの内容を更新します。更新方法は２通りがあります。

1．アップデートファイルを直接ダブルクリックする
2．ワイブキーアプレット"WIBUKE32.CPL"から更新する

1．アップデートファイルを直接ダブルクリックする

更新するワイブキーをPCに装着した状態で、アップデートファイル(RTUファイル）を直接ダブルクリックします。

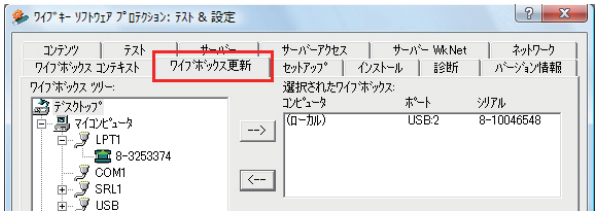
更新に成功すると、右のメッセージが表示されます。



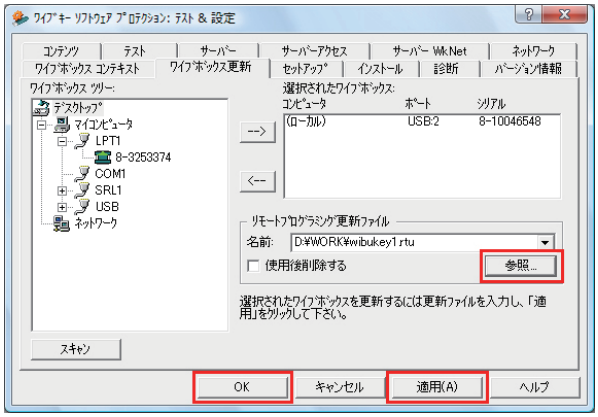
2．ワイブキーアプレット "WIBUKE32.CPL" から更新する

更新するためのワイブキーをPCに装着した状態で、コンテキストファイル（RTCファイル）を作成した時と同じように、コントロールパネルからワイブキーアプレット"WIBUKE32.CPL"を立ち上げます。

アドバンストモードにし、"ワイブボックス更新"メニューを選択します。

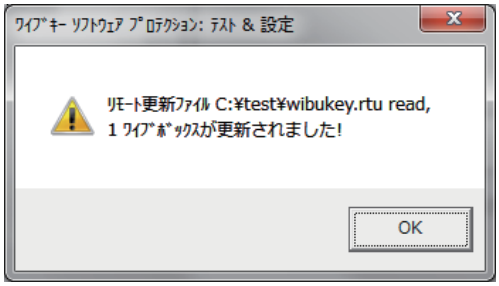


[参照] ボタンをクリックして、リモートプログラミング更新ファイル（.rtu）を選択します。
（例では"wibukey1.rtu"）



「使用後削除する」にチェックすると、更新完了後、使用したRTUファイルが削除されます。
[OK]ボタン、または［適用］ボタンをクリックすると、ワイブキーの更新が行なわれます。

これで更新作業は終了です。



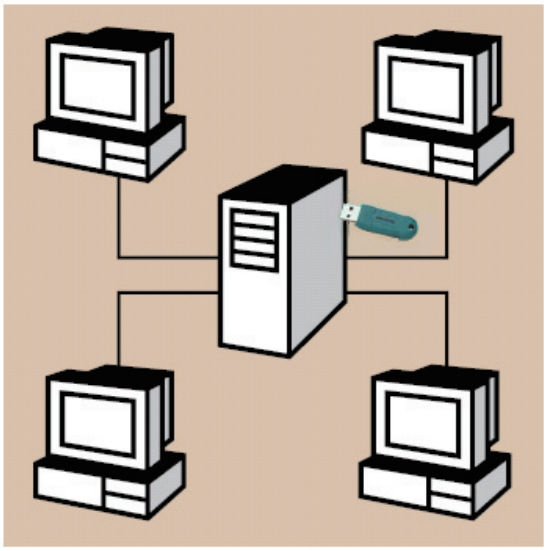
[注意]
１度使用したアップデートファイル(RTUファイル）は２回使用することはできません。また、コンテキストファイル（RTCファイル）を作成した時とは異なるワイブキーに対してアップデートファイルを適用することもできません。再度、ワイブキーの更新をする場合は、あらたにコンテキストファイルを作成し、そのコンテキストファイルに対してアップデートファイルを作成する必要があります。

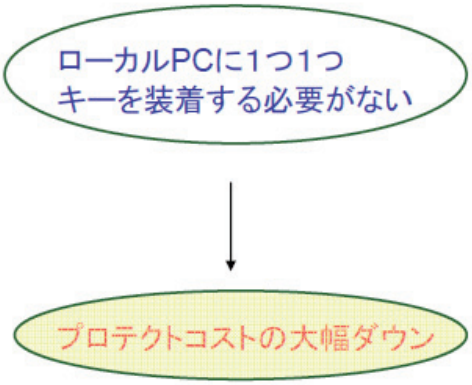
Chapter 7

ネットワーク機能について

7-1. ネットワーク機能とは

ワイブキーのネットワーク機能とは、ネットワーク上のサーバーにワイブキーを１つ装着することにより、1台から最大2,250台までのクライアント数を制限する機能です。ライセンス契約において、使用可能なライセンス数をワイブキーに登録してサーバーに装着することで、同時アクセス数によるライセンス制御が可能になります。ローカルPCに１つ１つ装着する必要がありませんので、プロテクトコストを大幅に節約できます。また、ライセンス数を指定しないで使用することも可能です。その場合は、ネットワーク上にあるすべてのクライアント（最大2,250台まで）からプログラムを起動することが可能になります。このネットワーク機能はプロフェッショナル版のみ対応しており、スケルトン版は対応しておりません。





ワイブキーを装着するサーバーは、必ずしもネットワークを管理するサーバーである必要がなく、クライアントの1台をワイブキーサーバーにすることが可能です。実際には、ワイブキーのネットワークサーバーソフト（WkSv32.exe）が起動・常駐しているPCがワイブキー・サーバーになります。

ネットワーク機能を構築する２種類の方式

ワイブキーのネットワーク機能を構築するには、２種類の方式があります。
１つは、ベーシック方式、もう１つはヒュージライセンスマネジメント(HLM)方式です。

ベーシック方式は、２つのワイブキーエントリを使い、ユーザーコードの差（引き算値）をクライアント制限数とする方法です。

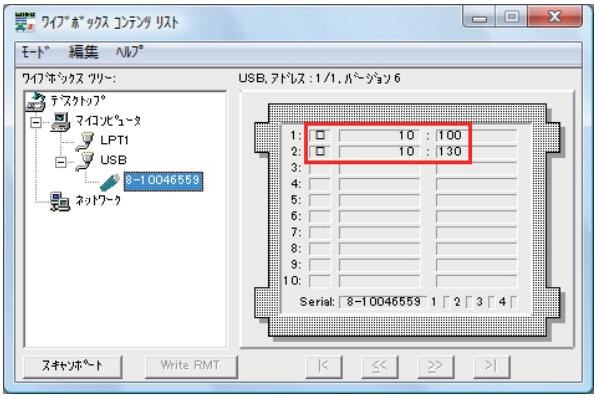
ヒュージライセンスマネジメント（HLM)方式は、コントロールファイルを作成し、そのコントロールファイル管理下でクライアント制限を行う方法です。この方式の場合、１個のワイブキーで制御できるアプリケーションは無限になり、各アプリケーションに対してそれぞれクライアント数を制限することが可能になります。

7-2. ベーシック方式

ベーシック方式の場合、クライアント数を制限するには、2 つのエントリを利用し、ユーザーコード（User Code）の差（引き算値）をクライアント制限数とします。例えば、プロテクトされたアプリケーションのコードがFirmCode =10、UserCode=100とします。このアプリケーションを使用できるクライアント数を30に設定する場合は、

エントリ 1 にFirmCode=10, UserCode=100を書き込みます。
エントリ 2 にFirmCode=10, UserCode=130を書き込みます。

これで、FirmCode=10,UserCode=100のプログラムは、130 -100=30のクライアント数まで使用できる状態になります。つまり、エントリ 2 のUserCodeからエントリ 1 のUserCodeを引いた数が、許可するクライアント数になるわけです。従い、クライアント数を50にする場合は、エントリ 2 のUserCodeを150にします。(150 - 100 = 50)



ここで、注意していただきたいのは、必ずエントリ 2 の方を大きな数にして下さい。（当然のことですが）また、UserCodeは100からではなく、1以上であれば、どんな整数値でもOKです。

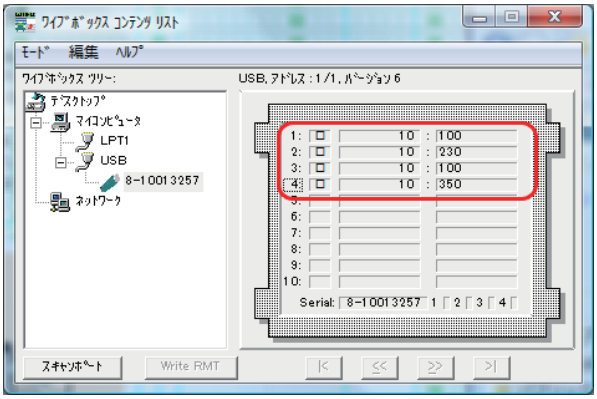
常に、{（エントリ 2 のUserCode）－（エントリ 1 のUserCode）} が「許可されるクライアント数」になります。（FirmCodeは同じである必要があります。）

UserCode(エントリ 2) - UserCode（エントリ 1 ）= クライアント数

ただし、エントリ 1 とエントリ 2 の差で設定するクライアント数は250以内である必要があります。従い、設定するクライアント数を380クライアントにする場合は、エントリ 3 とエントリ 4 も使用します。
FirmCode=10, UserCode=100のプログラムのライセンス数を380に設定する場合、

エントリ1: FirmCode=10、UserCode=100
エントリ2: FirmCode=10、UserCode=230
エントリ3: FirmCode=10、UserCode=100
エントリ4: FirmCode=10、UserCode=350

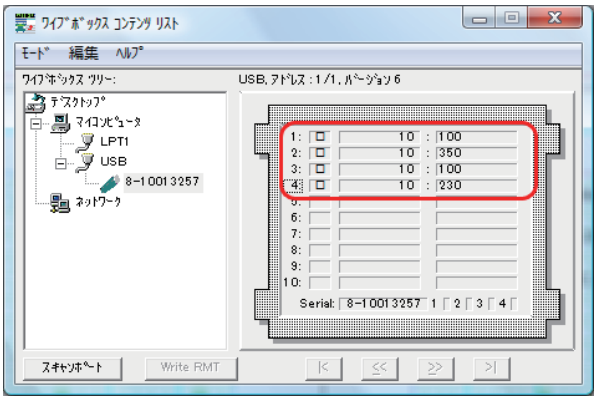
と登録します。
エントリ 1 とエントリ 2 の差が230-100=130
エントリ 3 とエントリ 4 の差が350-100=250
従い、ライセンス数は、130 + 250 = 380になります。
エントリ 1 とエントリ 3 は、必ずプログラムのFirmCode/UserCodeを設定します。



この場合、ご注意していただく点は、必ずエントリ１からエントリ10に向かって大きな数字を設定する必要があることです。例えば、次のように設定すると、クライアント数が変わります。

エントリ1: FirmCode=10、UserCode=100
エントリ2: FirmCode=10、UserCode=350
エントリ3: FirmCode=10、UserCode=100
エントリ4: FirmCode=10、UserCode=230

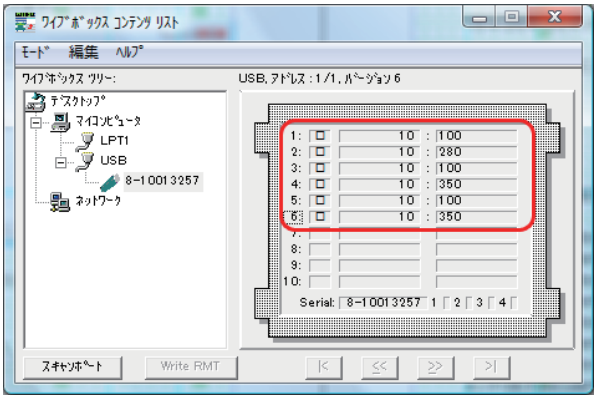
この場合、エントリ４の数値=230が優先され、エントリ２の350はスキップされるので、実際に設定されるクライアント数は230-100=130になり、380にはなりません。必ずエントリ１からエントリ10に向かって大きな数字を設定してください。



同じようにして、例えばクライアント数を680に設定する場合は、エントリ５とエントリ６も使用して次のように設定します。

エントリ1: FirmCode=10、UserCode=100
エントリ2: FirmCode=10、UserCode=280
エントリ3: FirmCode=10、UserCode=100
エントリ4: FirmCode=10、UserCode=350
エントリ5: FirmCode=10、UserCode=100
エントリ6: FirmCode=10、UserCode=350

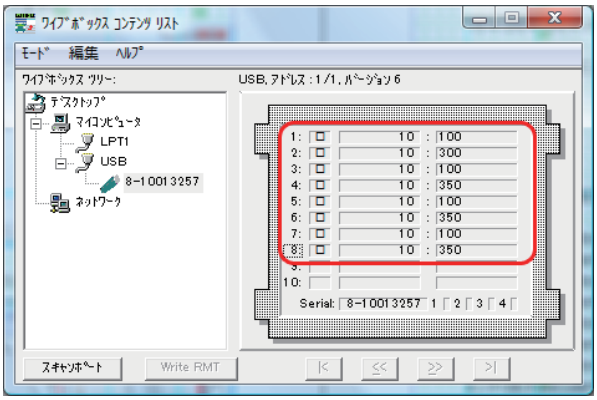
180 + 250 + 250 = 680クライアント



クライアント数=950の場合は、さらにエントリ７とエントリ８を使用します。

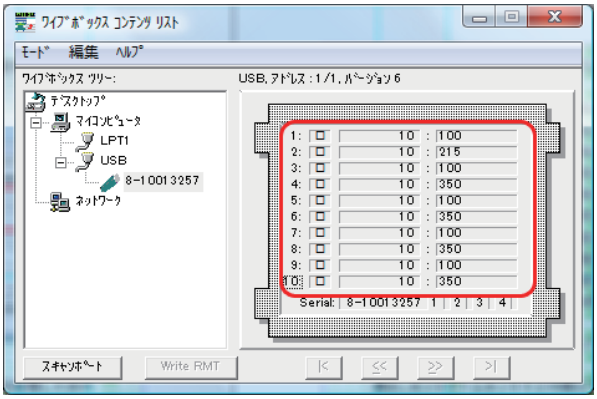
エントリ1: FirmCode=10、UserCode=100
エントリ2: FirmCode=10、UserCode=300
エントリ3: FirmCode=10、UserCode=100
エントリ4: FirmCode=10、UserCode=350
エントリ5: FirmCode=10、UserCode=100
エントリ6: FirmCode=10、UserCode=350
エントリ7: FirmCode=10、UserCode=100
エントリ8: FirmCode=10、UserCode=350

200 + 250 + 250 + 250 = 950クライアント



クライアント数=1,115の場合は、さらにエントリ９とエントリ10を使用します。

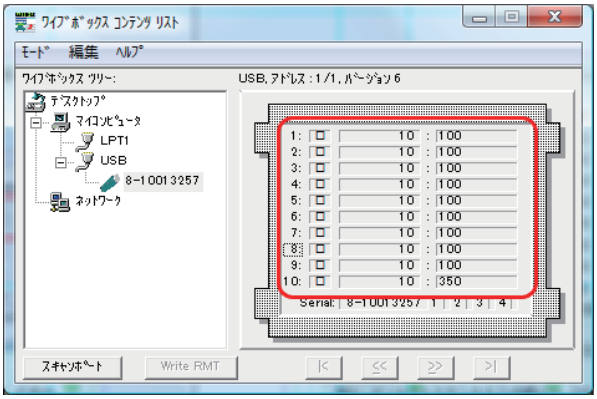
エントリ1: FirmCode=10、UserCode=100
エントリ2: FirmCode=10、UserCode=215
エントリ3: FirmCode=10、UserCode=100
エントリ4: FirmCode=10、UserCode=350
エントリ5: FirmCode=10、UserCode=100
エントリ6: FirmCode=10、UserCode=350
エントリ7: FirmCode=10、UserCode=100
エントリ8: FirmCode=10、UserCode=350
エントリ9: FirmCode=10、UserCode=100
エントリ10: FirmCode=10、UserCode=350



115 + 250 + 250 + 250 + 250 = 1,115クライアント

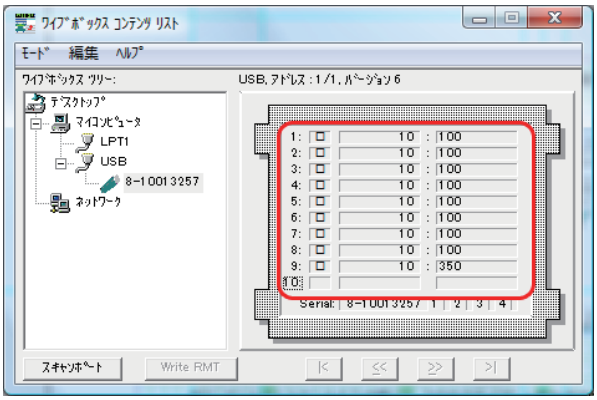
常に、エントリ１とエントリ２の間に1 - 250のクライアント数を設定することになります。
さらに、次のような設定も可能です。

エントリ1: FirmCode=10、UserCode=100
エントリ2: FirmCode=10、UserCode=100
エントリ3: FirmCode=10、UserCode=100
エントリ4: FirmCode=10、UserCode=100
エントリ5: FirmCode=10、UserCode=100
エントリ6: FirmCode=10、UserCode=100
エントリ7: FirmCode=10、UserCode=100
エントリ8: FirmCode=10、UserCode=100
エントリ9: FirmCode=10、UserCode=100
エントリ10: FirmCode=10、UserCode=350



この場合は、250 X 9 = 2,250クライアントになります。(FirmCode=10, UserCode=100)
また、エントリ１からエントリ９までを使用すると（エントリ10は使用しない）、

エントリ1: FirmCode=10、UserCode=100
エントリ2: FirmCode=10、UserCode=100
エントリ3: FirmCode=10、UserCode=100
エントリ4: FirmCode=10、UserCode=100
エントリ5: FirmCode=10、UserCode=100
エントリ6: FirmCode=10、UserCode=100
エントリ7: FirmCode=10、UserCode=100
エントリ8: FirmCode=10、UserCode=100
エントリ9: FirmCode=10、UserCode=350
エントリ10: ----------



250 x 8 = 2,000クライアントになります。(FirmCode=10, UserCode=100)

[NOTE]
ライセンス数を設定したエントリと同じFirmCode/UserCode（例えば、FirmCode=10,UserCode=350）を持つアプリケーションが混在すると、クライアント数=1として動作しますのでご注意ください。基本的には、ネットワーク対応アプリケーションのユーザーコードをローカル版とは異なるユーザーコードに設定し、1 個のワイブキーで、1 種類のアプリケーションをライセンス管理することをお勧めいたします。異なる複数のワイブキーを 1 台のサーバーに装着しても、ワイブキーはそれぞれのアプリケーションのライセンス管理をそれぞれ独立しておこないます。

[NOTE]
1 個のワイブキーで管理できる最大数は2,250クライアントですが、実際には、使用するサーバ環境およびワイブキーにかかる負荷等を考慮に入れてライセンス数を決定してください。また、複数のワイブキーを異なるサーバーに装着し、各負荷を軽減することも可能です。この場合、トータルライセンス数は、各ワイブキーに登録されたライセンス数の合計になります。
ワイブキーAのライセンス数:100 + ワイブキーBのライセンス数:100 = 200（トータルライセンス数）

7-3. ヒュージ・ライセンス・マネジメント (HLM) 方式

ベーシック方式の場合、1 つのアプリケーションのクライアント制限数の設定にワイブキーのエントリを 2 つ使用します。1 個のワイブキーには10個のエントリがあるため、1個のワイブキーで最大 5 種類のアプリケーションのクライアント管理をすることができます。
（2 エントリ x 5 アプリケーション = 10エントリ）

1 個のワイブキーで 5 種類以上のアプリケーションのクライアント管理をする場合は、ヒュージ・ライセンス・マネジメント（Huge License Management）方式を使用します。このヒュージ・ライセンス・マネジメント（HLM）方式を使用することで、次の 2 点が可能になります。

1. 1 つのワイブキーで無限のアプリケーションをプロテクト管理できる。
2. アプリケーションごとにクライアント数の設定ができる。

ヒュージ・ライセンス・マネジメント（HLM）方式は以下の手順で使用します。

1 . ヒュージ・ライセンス・マネジメント・コントロールファイル(Huge License Management Control File、以下、HLMコントロールファイル)を作成する。
2 . ワイブキーにHLMコントロールファイルの使用を登録する。
FC/UCエントリとマスターコードエントリを使用します。
3 . ワイブキーサーバーのWIBU-KEYコントールパネルで使用するHLMコントロールファイルを指定します。
以下、標準のワイブキーサーバーと同じです。

1. コントロールファイルの作成方法

HLMコントロールファイルは、最初にテキストファイルでスクリプトを記述（.wkcファイルを作成）したあと、“Wkcrypt.exe” を適用することによって生成されます。HLMコントロールファイルの拡張子は .wbb になります。

最初にHLMコントロールファイルのスクリプトファイルを作成します。テキストファイルの操作になりますので、現在ご利用のテキストエディタを使ってください。以下は、HLMコントロールファイルのスクリプトの雛型です。テキストエディタで入力して、拡張子を . wkc として保存してください。

```
/net          #Mode: netfile generation
/f10          #FirmCode : 10
/u0xf00000    #UserCode : 15728640
/n:"05070779" #Extended HLM Control File Name
/gh           #Command: generate HLM control file

# 1.Module
/u3145729          #UserCode
/q10               #Client count
/n:"suncarla-1"    #Nic Name
/gl

# HLM control file name
HLM-1.wbb
```

ヘッダー部分です。
/f のFirmCodeを除けばほとんど固定で使用できます。

以降はコメント文です。

モジュールブロック。
ユーザーコードとクライアント数を指定します。

このブロック必要数だけ列挙します。

フッタ部分です。
最後にHLMコントロールファイルの名前を記述します。

各部分について少し説明致します。

ヘッダー部分で変更が必要になるのは、/f で指定するFirmCodeの部分です。/f に続けて貴社のFirmCodeを記述して下さい。(例. /f1234)
ヘッダー部分はそれ以外変更する部分はありません。
/u0x f 00000 のユーザーコードもそのまま使用してください。

モジュールブロックには以下の項目が記述されます。
/u に続いてユーザーコード
/q に続いてクライアント数(最大250クライアント)
/n: に続いてダブルクォーツで囲んでニックネーム
/gl　はこのモジュールブロックの最後をあらわします。

/u で指定するUserCodeは任意の値を設定することができます。
ただし、それぞれのUserCodeがお互いに干渉しない状態で独立してコントロールしようとすると、割り当てるビットに工夫が必要になります。従い、マニュアルに記載のテーブル表をご利用することをお勧めいたします。
(例.　エントリ5用テーブル表を参考にして 5つのUserCodeを設定する場合は、0x300001、0x30002、0x300004、0x300008、0x300010)

/n: で指定するニックネームは、任意の名前を指定できますが、複数のモジュールを記述する場合は重複しない名前を付けて区別してください。

最後にフッター部で "Wkcrypt.exe" で実際に生成される HLMコントロールファイル の名前を記述してください。ここで指定したHLMコントロールファイル名が作成されることになります。拡張子は固定で .wbb でなければなりませんが、ファイル名の部分は必ずしもスクリプトファイルのファイル名と同じである必要はありません。また、.wbbファイルを常に一定のフォルダに作成したい時はフルPathで指定することもできます。(例. D:\WBB_WORK\HlmTest1.wbb)

2. HLM コントロールファイルの使用方法

HLMコントロールファイル(.wbbファイル) を生成するには「wkcrypt.exe」を使用します。
「wkcrypt.exe」は開発ツールのインストール先 WibuKey\DevKit\Bin に入っています。
(尚、「wkcrypt.exe」を他のフォルダにコピーして使う場合は、"WkAutoEnc.dll" と "wkfirm.dll" も同時にコピーしてください)

「wkcrypt.exe」をDOSモードで起動します。DOS互換Ｂoxを起動してください。
起動パラメータとして、スクリプトファイル名(.wkcファイル名)の先頭に@を付けて指定します。

書式　：　WKCRYPT　@[HLMスクリプトファイル名]
例　：　WKCRYPT　@HLMTEST.WKC

実行すると、拡張子が.wbb の HLMコントロールファイルが作成されます。
但し、HLMコントロールファイルはカレントディレクトリ(DIR↲ とした時に表示されるディレクトリ)に作成されますのでご注意ください。(スクリプトで保存先を指定していた場合はそちらが使われます)

HLMのスクリプトファイル(.wkcファイル)にエラーがある場合は、ライン番号が表示されますので、シンタックスエラー等が無いかご確認ください。

3. ワイブキーのプログラミング

ワイブキーにHLMコントロールファイルをプログラミングするには２つのエントリを使います。
１つ目のエントリに FirmCodeと UserCodeをプログミングし、次のエントリに HLMコントロールファイルで指定した全モジュールに対するマスターエントリコードを指定します。
実例を示します。

例では、エントリ１とエントリ２を使います。
最初のエントリには、HLMコントロールファイルのヘッダーで指定したFirmCode と UserCode をプログラミングします。

```
/net          #Mode: netfile generation
/f10          #FirmCode : 10
/u0xf00000    #UserCode : 15728640
/n:"05070779"  #Extended HLM Control File Name
/gh           #Command: generate HLM control file
```

このヘッダーの場合は /f10 と/u0xf00000 になります。
他の項目はワイブキーのプログラミングには必要ありません。
従い、エントリ１は

エントリ１　：　FirmCode 10 、UserCode 15728640

となります。

次に、HLMコントロールファイルに記述した全てのモジュールが使用しているUserCode のマスターエントリコードを計算します。マスターエントリコードの計算は、実際には全ビットのORをとるだけです。
ここでは、HLMコントロールファイルに以下の３つのモジュールを設定していたとします。

```
# 1.Module
/u3145729
/q10
/n:"suncarla-1"
/gl

# 2.Module
/u3145730
/q10
/n:"suncarla-2"
/gl

# 3.Module
/u3145732
/q10
/n:"suncarla-3"
/gl
```

これらの各モジュールで /u で指定している UserCodeを抜き出してORをとります。
（ 3145729 OR 3145730 OR 3145732 ）= 3145735(Dec)
16進表記にすると

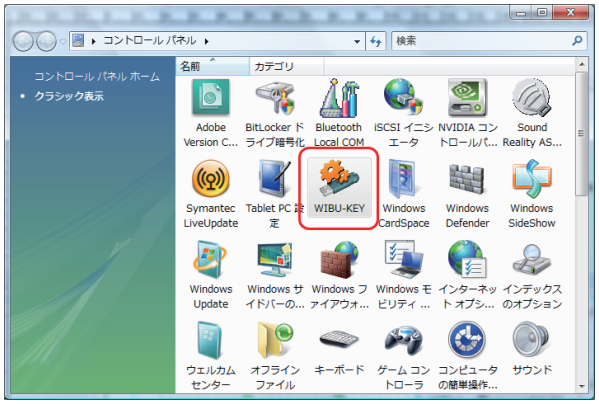
（0x300001 OR 00002 OR 0x300004）＝ 0x300007（Hex）
となります。従い、2番目のエントリは、マスターエントリを選択して FirmCode10、UserCode3145735 をプログラミングします。

エントリ１　：　FirmCode 10 、UserCode 15728640　・・・・FC/UC
エントリ２　：　FirmCode 10 、UserCode 3145735　　・・・・マスター

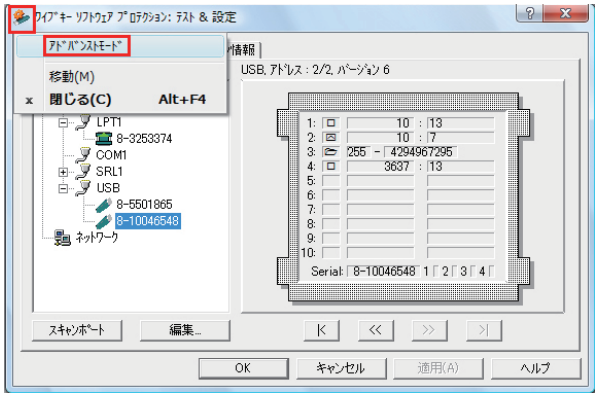
以上でワイブキーのプログラミングは完了です。

4. ワイブキーサーバーで実際に使用する

HLMコントロールファイル（.wbbファイル）を、ワイブキーサーバーの適当なフォルダにコピーします。次に、ワイブキーコントロールパネルの「サーバー」の画面でHLMコントロールファイルを追加登録します。コントロールパネルにあるワイブキーアイコンをダブルクリックして"WIBUKE32.CPL" を起動します。

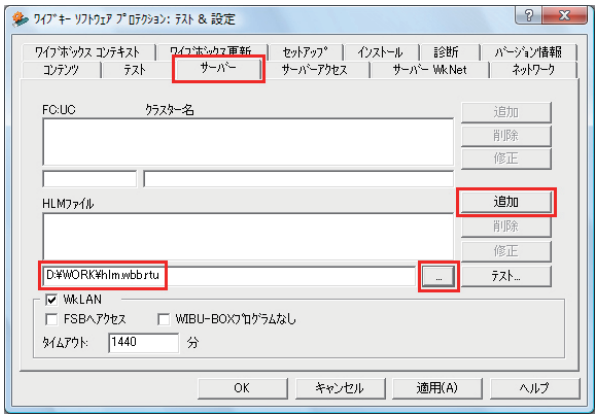


"WIBUKE32.CPL"は、起動直後は必要最小限の情報だけが表示されます。サーバー機能を有効にするには、画面左上端のアプリケーションアイコンをクリックしてポップアップメニューを表示し、「アドバンストモード」を選択して下さい。

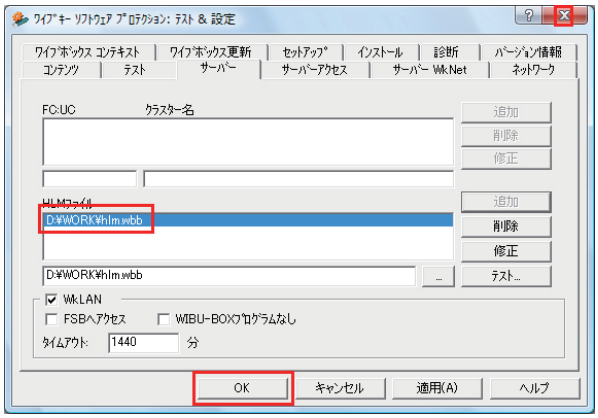


「WIBUKE32.CPL」が「アドバンストモード」で表示されます。（アドバンストモード画面）

「サーバー」タブを開いて、ファイル選択ダイログボックスでHLMコントロールファイル（.wbbファイル）を開くと、選択したファイルが画面に表示されます。[追加]ボタンをクリックして下さい。



上記手順で選択したHLMコントロールファイル（.wbbファイル）が HLMファイルのリストに追加されます。[OK] ボタンをクリックし画面を閉じます。



[NOTE]
追加後は、サーバーを再起動して下さい。

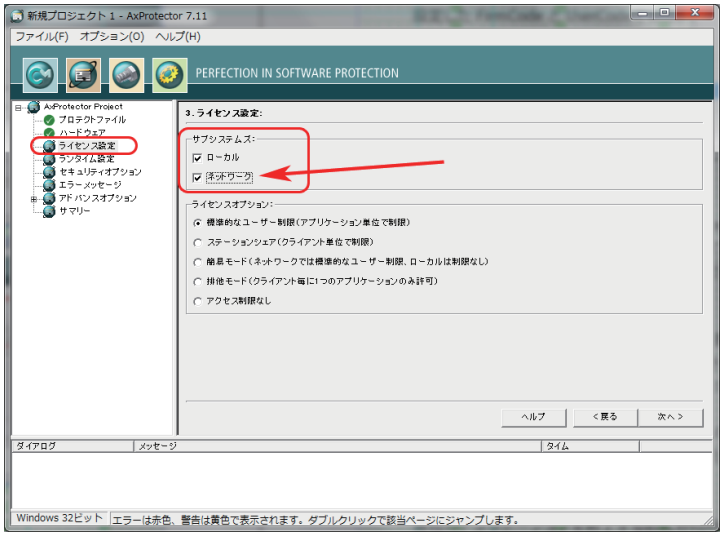
[NOTE]
ワイブキーのヒュージ・ライセンス・マネジメント(HLM)方式によるフローティングライセンス管理は、使用方法が多少複雑です。別商品の「コードメータ」には、ネットワーク上のライセンス管理を簡単に実現できる機能が搭載されていますので、複数アプリケーションのフローティングライセンス管理をシンプルにかつ確実に行いたい場合は、「コードメータ」のご利用をお勧め致します。

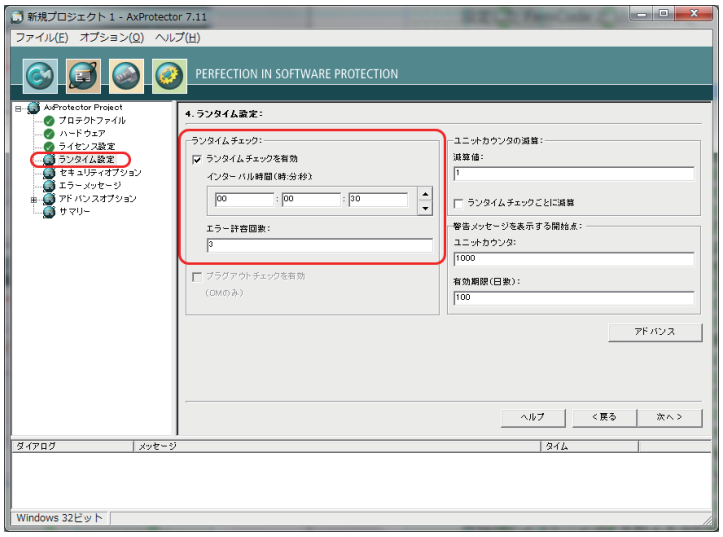
7-4. ネットワーク対応プログラムの作り方

ワイブキーのネットワーク機能を利用するためのプログラム作成には２通りの方法があります。１つは、自動暗号化ツール「AxProtector」のネットワークオプションを使用する方法。もう１つは、ワイブキーのAPIファンクションを使ってソースコードに組み込む方法です。

1. 自動暗号化ツール「AxProtector」を使用する場合

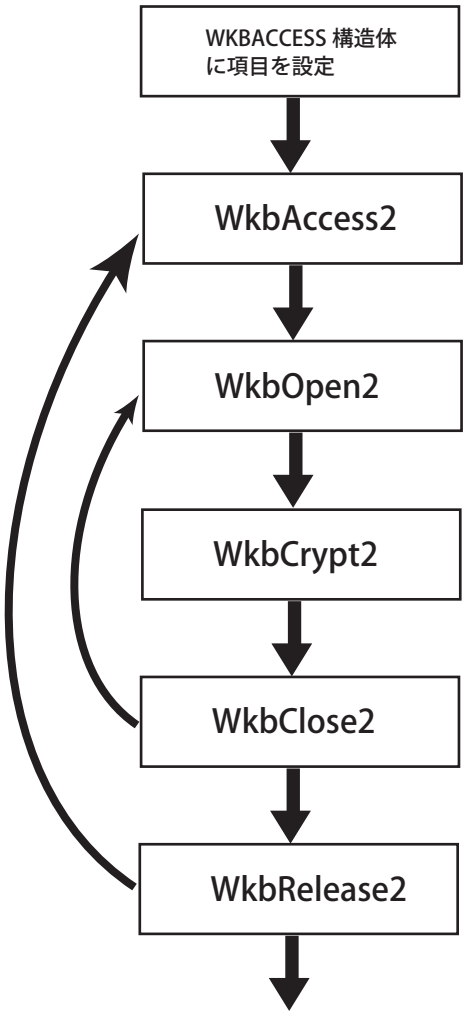
AxProtectorの「ライセンス設定」と「ランタイム設定」を利用します。「ライセンス設定」画面 の「サブシステムズ」で、「ネットワーク」 にチェックを入れます。（初期設定ではチェックが外れています。）
さらに、「ランタイム設定」画面の「ランタイムチェック」で、「ランタイムチェックを有効」にチェックを入れます。インターバル時間は任意に設定可能です。





[注意]
ランタイムチェックを有効にしないと、ライセンス制限が有効になりませんので、必ずチェックを入れてください。もし、ランタイムチェックを有効にしなかった場合、ライセンス数は無視され、プログラムは無制限に動作します。

2. API ファンクションを使用する場合



右図はネットワーク上のワイブキーにAPIファンクションを使ってアクセスする最も簡単な手順を示した図です。

WKBACCESS構造体に必要なFirm Code、User Code、flCtrlなどの項目を設定します。

WkbAccess2を実行すると、WKBACCESS構造体に設定した FirmCode と UserCodeでWIBU-KEYサブシステムにアクセスが行われます。

WkbAccess2が成功するとハンドルが返されます。この段階で、該当するFirmCode、UserCodeのワイブキーへのアクセスは成功しています。従って、単純なチェックの場合はこれで目的を達したことになります。

WkbAccess2が成功するとネットワークへのLogonが行われてワイブキーサーバーにクラスタが1つ生成されます（この時点で接続中のクライアントとして認識されます）。

この後、追加エントリ等更に詳しい情報が必要な時は、WkbAccess2で返されたハンドルを使ってWkbOpen2ファンクションで目的のエントリをオープンします。
WkbOpen2ファンクションで取得したハンドルをベースに各WIBU-KEY APIを実行します。
WkbOpen2などでワイブキーチェックを行った後、WkbClose2でWIBU-BOXエントリをクローズしてから、貴社のアプリケーションを実行します。

貴社のアプリケーションを終了する前にWkbRelease2ファンクションでWIBU-KEYサブシステムを解放します。解放後、LOGOFF状態になりクラスタが 1 つ解放されます。

WkbRelease2を実行すると、ワイブキーサーバーからのログオフが行われて使用中のslotが1個開放されます。接続可能なクライアント数が1つ開放されることになりますので、WkbRelease2を実行するタイミングにご注意下さい。

［NOTE］
flCtrlに設定するコントロールコードによってワイブキーへのアクセス動作を制御することができます。以下はVisual BasicでflCtrlに標準的な値を設定した例です。

```
flCtrl& = WKB_STDCTRL Or WKB_ACC_CREATE Or WKB_ACC_LOCAL _
        Or WKB_ACC_WKLAN Or WKB_ACC_WKNET
```

上記のようにWKB_ACC_LOCAL、WKB_ACC_WKLAN、WKB_ACC_WKNET を同時に指定した場合はLocal→LANの順にワイブキーの自動検索が行われます（WIBU-KEYコントロールパネルで後から検索順を変更することができます）。
flCtrlにWKB_ACC_LOCALを指定しない場合は、ワイブキーサーバーだけが対象になります。逆に、

WKB_ACC_LOCALを指定して、WKB_ACC_WKLANとWKB_ACC_WKNETを指定しないとLocalのワイブキーのみが対象となります。従って、WkbAccess2はLocal版のプロテクトとしてもそのまま使用することができます。

尚、各言語用のサンプルソースが用意されていますので、詳細はそちらをご覧ください。サンプルプログラムのインストール方法につきましては、「Chapter 9 WIBU-KEY APIファンクションについて / 9-2. サンプルプログラムのインストール方法について」をご参照ください。ネットワーク用のサンプルは各言語とも ＜WkDmN＞フォルダになります。

7-5. ワイブキー・サーバーの作成方法

ワイブキーは、ワイブキーのネットサーバー(WkSvW32.exe)が起動・常駐しているPCをサーバーとみなします。従い、実際のネットワークサーバーをワイブキーサーバーにすることも可能ですし、1 クライアント上でWkSvW32.exeを起動・常駐させ、そのクライアント（PC）をワイブキーサーバーにすることも可能です。

ワイブキー・ドライバーを標準インストールするとインストール先の¥Program Files¥WibuKey¥Serverフォルダに"WkSvW32.exe"がインストールされます。ワイブキーサーバーを立てるPCにワイブキーを装着し、"WkSvW32.exe"を起動してください。これでワイブキーサーバーに接続されているクライアントからワイブキーサーバーを参照することができるようになります。（ただし、クライアント側にもワイブキーのドライバーをインストールしておく必要があります。）

Chapter 8

拡張メモリー (ExtMem) について

8-1. 拡張メモリ（ExtMem）について

ExtMemはソフトウェアによってプログラミングが可能な不揮発性のメモリです。この機能を利用してワイブキーに永続的にデータを保存しておくことができます。この機能は、ワイブキーハードウェアバージョン6以降のすべてのワイブキーで使用することができます。

現在、ExtMemに対応するワイブキーは以下のとおりです。

WIBU-BOX/U+（プロフェッショナル版）
WIBU-BOX/P+（プロフェッショナル版）
WIBU-BOX/RU+（スケルトン版）

[NOTE]
ワイブキーハードウェア本体の表面に"+"が印刷表示されていないワイブキー"WIBU-BOX/U"、"WIBU-BXO/P"、"WIBU-BOX/RU"はExtMemに対応していません。現時点でリリースされているハードウェアバージョン6（プロフェッショナル版）および7（スケルトン版）の場合、ExtMem (16Kbytesのメモリ空間)を利用することができます。

ExtMemは主にデータ保存のために設計されています（FirmCode、UserCode等のキー保存のためではありません）。従い、ExtMemをワイブキーの暗号化オペレーション、または他のセキュリティ・オペレーションの初期化目的で使用することはできません。

ExtMemはUserTypeと呼ばれるExtMemブロックと、ProtectedTypeと呼ばれるExtMemブロックの2つのExtMemブロックによって構成されます。それぞれのExtMemは0～255ページで構成されます。1ページは32バイトで構成されます。　従い、ExtMemの全容量は（32bytes×256pages=8KBytes ）×2＝16KBytes となります。

UserTypeの8KByteは読み書きに際して、特別なパスワードやアクセスコードを必要としません。

ProtectedTypeの8KByteは読み書きに際して、8バイトの特別なアクセスコード（以下、シグネチャー ）を必要とします。シグネチャーの値は ワイブキーのエントリ1の設定内容に依存します。

8-2. クラシック API による ExtMem のアドレッシング

ExtMemのページは、以下のWKBEXTMEM構造体によって定義されます。

```
typedef struct {
        USHORT fsCtrl; /* contains WKB_XM_xxx */
        USHORT iPage;  /* contains index of the ExtMem page */
        BYTE abSignature[8];  /* contains access signature for WKB_XM_PROT */
        BYTE abData[32];  /* receives or contains data of ExtMem page */
}  WKBEXTMEM;  /* prefix "wkbxm" */
```

fsCtrl ・・・メモリタイプを指定
WKB_XM_USER (0) ・・・UserType
WKB_XM_PROT (1)・・・ ProtectedType
ビット1～15は、将来の使用のために予約されており、0にセットされます。
iPage ・・・ページ・インデックスを指定。0ページ～255ページ。
abSignature・・・アクセスコード (8バイトのシグネチャー) を指定。
UserTypeのExtMemを利用する場合は8バイトのゼロで埋めます。
ProtectedType ExtMemを利用する場合は有効な8バイトのシグネチャーを設定する必要があります。この値は「WKCRYPT.EXE」で事前に取得します。
abData・・・読み書きするページデータ。32バイトの読み書き用バッファ。

ExtMemページの読込みにはワイブキーAPIのWkbListBox2ファンクションを使用します。
flCtrlパラメータにWKB_LIST_EXTMEMを指定し、pvDestでWKBEXTMEM構造体のアドレスを指定します。cbDestには sizeof(WKBEXTMEM) をセットする必要があります。

ExtMemページの書き込みにはWkbProgram2ファンクションを使用します。
flCtrlパラメータにWKB_PGM_EXTMEMを指定し、pvPgmDataでWKBEXTMEM構造体のアドレスを指定します。

ワイブキーの利用可能なExtMemのページ数は、WKB_LIST_CONFIGパラメータを使ってWkbListBox2ファンクションで取得することができます。返されたWKBBOXCONFIG構造体は、usExtMemSizeフィールドの全てのExtMemページ数を含みます。ページ数はUserTypeとProtectedTypeの合計の数値です。現在のワイブキー/+では512を返します (512page×32byts=16Kbytes)。

[注意]
ExtMemを利用する場合 、ワイブキー・ドライバーはVer4.00以上が必要です。
ExtMemは、Windows 95以降のWin32上、MacOS X以後、Linux上でのみサポートされます。
16ビット-Windows、DOSまたはOS/2ではサポートされません。
WkLANを介してのExtMemは無制限のアクセスを許可します。

8-3. COM API による ExtMem のアドレッシング

現在のところCOM APIによるExtMemへのアドレッシングは、UserTypeのExtMemについてのみサポートされます。COM APIによるExtMemのアドレッシングについては、ProtectedTypeへのアドレッシングがサポートされ次第、順次掲載します。

8-4. Protected Type ExtMem へのアクセス

Protected TypeのExtMemページへの読み書きを行なう場合は、常に8バイトのアクセス用のシグネチャーが必要となります。ExtMemページの読み込みにおいては常に同じシグネチャーが使われますが、書き込みで要求されるシグネチャーとは必ずしも一致しません。また、シグネチャーは設定されたセキュリティーモードに依存しますので、書き込みのためのシグネチャーはページ毎に異なることもあります。以下にシグネチャーについて簡単に説明します。

シグネチャーは、エントリ 1 に設定された FirmCode/UserCodeに依存します。従って、エントリ1がプログラミングされていない場合、またはUser Dataがプログラミングされている場合は、ProtectedTypeへのアドレッシングを行なうことはできません。

エントリ1 以外のエントリはシグネチャーに影響を与えません（但し、追加エントリを設定した場合はエントリ6が追加で使われます）。

エントリ1が追加エントリを含まない場合、読込みと書き込みのシグネチャーは同じになります。

読み込みの為のシグネチャーは、エントリ1の追加エントリに設定されたAdded DataまたはExpiration Dateの変化フラグ（以下、Modify Dependenceの設定 ）の設定に依存します。Modify Dependenceの設定が指定されない場合は、読込みの為のシグネチャーはFirmCode/UserCodeにのみ依存します。

読み込みの為のシグネチャーは、ワイブキーのシリアル番号、内部プログラミングカウンタからは常に独立して機能します。このことは、エントリ1のFirmCode/UserCodeが同じワイブキーにおいては共通のシグネチャーを使って読み込みが可能であることを意味します。但し、追加エントリを設定している場合は、Modify Dependenceの設定によっては変化する場合もありますので注意して下さい（例えば、リモートプログラミングによって追加エントリの設定を更新すると内部プログラムカンタが変化します）。

エントリ1にAdded Data、またはExpiration Dateを設定しModify Dependenceの設定で「Serial」を設定した場合、書き込みの為のシグネチャーはワイブキーのシリアル番号に依存します。従って、この場合は書き込みの為のシグネチャーがワイブキーごとに異なることになります。

書き込みの為のシグネチャーはワイブキーの内部プログラミングカウンタに依存します。

シナリオ	エントリ1の内容	読込みシグネチャー	書き込みシグネチャー
1	未設定、または UserData を設定。	使用不可。	使用不可。
2	FirmCode/UserCode を設定。 LimitCounter の追加エントリは無し。	FirmCode/UserCode に依存。	読込みシグネチャーと同じ。
3	FirmCode/UserCode を設定。 追加エントリで、AddedData または ExpirationDate を指定。 Modify Dependence の設定は無し。	シナリオ 2 と同じ。	シナリオ 2 と同じ。
4	シナリオ 3 に加えて Modify Dependence の「シリアル」を設定。	シナリオ 2 と同じ。	シナリオ 2 と同じだがワイブキーのシリアルに依存。
5	シナリオ 3 に加えて Modify Dependence の「プログラムカウント」を設定。	シナリオ 2 と同じ。	シナリオ 2 と同じだがワイブキーの内部プログラミングカウンターに依存。
6	シナリオ 3 に加えて Modify Dependence の「データコンテント」を設定。	シナリオ 2 と同じで、追加エントリの AddedData または ExpirationDate の設定内容に依存。	シナリオ 2 と同じだがアクセスした ExtMem のインデックスに依存。

8-5. Raw モードと Formatted モードにおける ExtMem 構成

ExtMemのメモリの状態にはRawモードとFormattedモードがあります。

Rawモード

デフォルトのExtMemはRawモードで構成されています。全てのページは構造化による規制を受けませんので、ソフトウェアで自由に読み書きすることができます。

Formattedモード

ExtMemを4つの領域によって構造化します。エンドユーザー、開発会社、WIBU-SYSTEM等、利用者レベルによるアクセス制限領域が設定されます。Formattedモードによって設定される領域は以下の4つです。

- Raw（共有メモリ）
- Licensor
- OEM（WibuSystemsとの特殊なパートナーのための予約領域）
- WIBU(WibuSystemsによる予約領域）

Formattedモードが有効であれば、ExtMemは常にUserTypeとProtectedTypeとして使われます。現在のExtMemがどちらのモードであるかは、「WKCRYPT.EXE」の/Lオプションによって確認することができます。ExtMemのフォーマットおよびアンフォーマットは「WKCRYPT.EXE」で行ないます。

以下に、Formattedモード時のExtMemの構造について説明します。

Formattedモードは、UserTypeとProtectedTypeのそれぞれのブロックの最後のページの offset 00～15に以下の固定のシグネチャー(GUID)が書き込まれることによって有効になります。

{00090001-0000-1200-8002-0000C06B5161}

具体的なバイト列は以下のとおりです。これはWkCryptのフォーマットで自動的に書き込まれます。

01 00 09 00 00 00 00 12 80 02 00 00 C0 6B 51 61

WIBU-SYSTEMSのツールは、UserTypeまたはProtectedTypeの最後のページにおいて、先頭の16バイトでこのシーケンスを見つけると常にFormattedモードとみなして動作します。従って、RawModeで使用する場合は、 UserTypeブロック またはProtectedTypeブロックの最後のページに、前記の16バイトのシーケンスを保存してはいけません。

ExtMemが正常にフォーマットされている場合、UserTypeとProtectedTypeの最後のページは常に同じ内容が保存されます。これらのページでは、最初の16バイトシーケーンスに続く offset 16～31には下位アドレスに配置される4つの領域の配置状況を表すページ情報が保持されています。
offset 16以降の2つの隣り合うバイトはリトルエンディアン16ビット値であり、ページ中の特定のフォーマットブロックのサイズ（ページ数）を示しています。

offset 16～17: Raw Block User Type length, RawUserSize UserType
offset 18～19: Licensor Block User Type, LicensorUserSize
offset 20～21: OEM Block User Type, OemUserSize
offset 22～23: WIBU Block User Type, WibuUserSize
offset 24～25: Raw Block Protected Type, RawProtectedSize ProtectedType
offset 26～27: Licensor Block Protected Type, LicensorProtectedSize
offset 28～29: OEM Block Protected Type, OemProtectedSize
offset 30～31: WIBU Block Protected Type, WibuProtectedSize

offset 16～23に保持された4つの値（word）の合計と、offset24～31に保持された4つの値（word）の

合計は同じでなければなりません。そして、それらの合計はExtMem全体で利用可能なページの総数と同一でなければなりません。

4つのフォーマットブロックの構想は以下のとおりです。

名前	説明	LicensorApplication からのアクセス
Raw	対話型ツールによる自由なアクセスを許可します。	Yes
Licensor	WIBU-KEY Licensor（開発元）のアプリケーションにアクセスを許可します。	Yes
OEM	WIBU-SYSTEMS の特別なパートナーのための予約領域。	No
WIBU	WIBU-SYSTEMS のための予約領域。	No

［注意］
WIBU-KEY APIは、RawモードとFormattedモードを識別しません。ExtMemメモリを常にRawモードで参照します。 従って、APIを使用するユーザーは間違ったRawモード書き込みによってFormattedモードのデータを破損しないようにする責任があります。同様に、Wklist32.exe 等のツールにおいても上記の4ブロックにかかわらず読み書きすることができます。

8-6. Formatted モードにおける Raw データ のアドレッシング

Rawブロックの最大インデックスは、以下のオペレーションによって取得することができます。これはUserTypeとFormattedTypeで共通です。

- 最大インデックスを特定するために、まずExtMemの最後のページをインデックスします。
- 最後のページを読んでGUIDをチェックします。Formattedモードを示す GUID が見つからない場合、ExtMemはRawモードで使うことができます。その場合は、全てのページは規制なしでアドレッシングすることができます。
- Formattedモードを示すGUIDが見つかった場合、UserType用としてoffset 16/17、FormattedType用として offset 24/25の値を読みます。データはリトルエンディアンで格納されていますので、実際の値は byte 16 + 256 * byte 17、byte 24 + 256 * 25. となります。得られる値は、RawUserSizeとRawProtectedSizeです。Rawブロックは、0～（RawUserSize-1）、0～（RawProtectedSize-1）のどのようなインデックスを持つこともできます。
- RawUserSizeと RawProtectedSizeがゼロの場合、Rawブロックは存在しません。

一般的に、ブロックの第一ページが、Rawブロックの構造を記述するために使われますが、実際にはLicensor（またはユーザー）によって自由に定義することができます。

「WKCRYPT.EXE」は、正確な開始位置を規定する目的で、フォーマット時にRawブロックの最初のページに32バイトの0を常に書き込みます。（UserTypeブロック、ProtectedTypeブロックとも書き込みます）

8-7. Formatted モードにおける Licensor データのアドレッシング

Licensorブロックの最初のページは常にインデックス0になるとは限りません。従ってFormattedモードでのLicensorデータのアドレッシングはRawデータに比べて少し複雑になります。

Licensorデータの開始ページと終了ページのインデックスは以下のオペレーションで取得されます。これはUserType、ProtectedTypeで共通です。

- ExtMemの最後のページをインデックスします。
- 最後のページを読んでGUIDをチェックします。
- Formattedモードを示す GUID が見つからない場合、ExtMemはRawモードで使われていますので操作を中止します。
- Formattedモードを示すGUIDが見つかった場合、UserType用としてoffset 16/17、FormattedType用としてoffset 24/25の値を読みます。データはリトルエンディアンで格納されていますので、実際の値は byte 16 + 256 * byte 17、byte 24 + 256 * 25. となります。得られる値は、RawUserSize と RawProtectedSizeであり、Licensorブロックの最初のインデックスを指しています。
- 次にUserType用としてoffset 18/19、FormattedType用としてoffset 26/27の値を読みます。データはリトルエンディアンで格納されていますので、実際の値は byte 16 + 256 * byte 17、byte 24 + 256 * 25. となります。得られる値は、LicensorUserSizeとLicensorProtectedSizeになります。
- LicensorUserSize と LicensorProtectedSize がゼロの場合、Licensorブロックは存在しませんので、操作を中止します。この時、Licensorブロックは、RawUserSize ～ (RawUserSize+LicensorUserdSize-1)、RawProtectedSize ～ (RawProtectedSize+LicensorProtectedSize-1) の範囲のインデックス値を取ることができます。

一般的に、ブロックの第一ページが、そのブロックの構造を記述するために使われますが、実際にはLicensor (またはユーザー) によって自由に定義することができます。

「WKCRYPT.EXE」は、正確な開始位置を規定する目的で、フォーマット時にRawブロックの最初のページに32バイトの0を常に書き込みます (UserTypeブロック、ProtectedTypeブロックとも書き込みます)。

8-8. WkCrypt による ExtMem の操作

WkCrypt Version4.10以降では、RawモードとFormattedモードにおけるExtMemのフォーマット と アンフォーマットの為のコマンドをサポートします。

/L オプションを使って現在の　ExtMemの状態を表示することができます。

以下はデフォルト設定の ワイブキー/U+ を　/Lで参照した例です。
※　日本語版MS-DOSを使用する場合、エンコードの関係でフォントが崩れ、下記表示のようにエントリの枠が文字化けして表示されますが、使用には差し支えありません。

C:>wkcrypt　/PAU　/L

```
wkcrypt - WIBU-KEY Encryption and Programming Tool.
Version 5.00 of 2005-Apr-14 (Build 49) for Win32.
Copyright (C) 1989-2005 by WIBU-SYSTEMS AG. All rights reserved.

WIBU-BOX 1: ExtMem area (512 pages) formatted.
### WIBU-BOX no. 1 at USB (Version 6):

ﾚﾄﾄﾄﾄﾄﾄﾄﾄﾄﾟ°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°ﾄﾄﾄﾄﾄﾄﾄﾄﾄﾂ
ｳ 1:        10:13       ｳ 2:                    ｳ 3:                    ｳ
ｳ 4:                    ｳ 5:                    ｳ 6:                    ｳ
ｳ 7:                    ｳ 8:                    ｳ 9:                    ｳ
ｳ10:                     ｳ Serial-No 008-0010007621 <00101>             ｳ
ﾀﾄﾄﾄﾄﾄﾄﾄﾄﾄﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾄﾄﾄﾄﾄﾄﾄﾄﾄﾙ

* ASIC Build Number: 818, ExtMem: 16 KBytes formatted, page structure:
  * User: Raw =  100, Licensor =  100, OEM =   30, WIBU =   25.
  * Prot: Raw =  100, Licensor =  155, OEM =    0, WIBU =    0.
```

以下はフォーマットされた ワイブキー/U+ を　/Lで参照した例です。

C:>wkcrypt　/PAU　/L

```
wkcrypt - WIBU-KEY Encryption and Programming Tool.
Version 5.00 of 2005-Apr-14 (Build 49) for Win32.
Copyright (C) 1989-2005 by WIBU-SYSTEMS AG. All rights reserved.

WIBU-BOX 1: ExtMem area (512 pages) formatted.
### WIBU-BOX no. 1 at USB (Version 6):

ﾚﾄﾄﾄﾄﾄﾄﾄﾄﾄﾟ°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°ﾄﾄﾄﾄﾄﾄﾄﾄﾄﾂ
ｳ 1:        10:13       ｳ 2:                    ｳ 3:                    ｳ
ｳ 4:                    ｳ 5:                    ｳ 6:                    ｳ
ｳ 7:                    ｳ 8:                    ｳ 9:                    ｳ
ｳ10:                     ｳ Serial-No 008-0010007621 <00101>             ｳ
ﾀﾄﾄﾄﾄﾄﾄﾄﾄﾄﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾄﾄﾄﾄﾄﾄﾄﾄﾄﾙ

* ASIC Build Number: 818, ExtMem: 16 KBytes formatted, page structure:
  * User: Raw =  100, Licensor =  100, OEM =   30, WIBU =   25.
  * Prot: Raw =  100, Licensor =  155, OEM =    0, WIBU =    0.
```

1. ExtMem のフォーマットとアンフォーマット

「WKCRYPT.EXE」の /PXMFオプションで ワイブキーのExtMemをフォーマットします。
/PXMFオプションはコマンドラインまたは WKCファイルで使用することができます。
ワイブキーを特定する為の 、/PAオプション（port）、/PBオプション（特定のインデックス）、/PQオプション（同時に操作するワイブキー数） を併用することができます。

フォーマットコマンドでは、Licensorブロック、OEMブロック、WIBUブロックのサイズ（ページ数）を指定して、以下の4つの領域で構造化します。

・Rawデータ
・Licensor
・OEM
・WIBU

ExtMemのフォーマット

フォーマットコマンドの書式は以下のとおりです。

WkCrypt /PXMF：**パラメータ 1, パラメータ 2,・・・パラメータ 6**

（例．WkCrypt /PAU /PXMF：W64, O10, L100, PW 1O, PO10, PL100 ）

パラメータには以下のブロックとそのサイズ（ページ数）を指定します。

W ・・・・・ UserTypeのWIBUブロック。 ProtectedTypeの場合は PW。

O ・・・・・ UserTypeのOEMブロック。 ProtectedTypeの場合は PO。

L ・・・・・ UserTypeのLicensorブロック。 ProtectedTypeの場合は PL。

例えば、WIBUブロックを10ページ確保する場合はW10 と指定します。他のブロックについて同時に指定する時は、カンマ（, ）で区切って続けて記述します。W, O, L のブロックはデフォルトではUserTypeブロックを指します。ProtectedTypeを指定する場合は、PW, PO, PL のように先頭に P を付けます。

フォーマットコマンドではRawブロックは指定しません。W, O, L で指定した後の残りのページ数が自動的にRawデータに割り当てられます。また、パラメータで明記しないブロックには0ページが割り当てられます。指定したページ数の合計がExtMemの最大ページ数を超えた場合はエラーになり、フォーマット作業は中止します。

次のコマンドは、UserType に WIBUブロック=25、OEMブロック=30、Licensorブロック=100 を設定し、ProtectedType に Licensorブロック=155 を割り当てる例です。

/PAU　でUSBタイプにワイブキーを指定し、最後に　/L で結果を表示しています。
WkCrypt　/PAU　/PXMF:W25,O30,L100,PL155　/L

```
wkcrypt - WIBU-KEY Encryption and Programming Tool.
Version 5.00 of 2005-Apr-14 (Build 49) for Win32.
Copyright (C) 1989-2005 by WIBU-SYSTEMS AG. All rights reserved.

WIBU-BOX 1: ExtMem area (512 pages) formatted.
### WIBU-BOX no. 1 at USB (Version 6):

ﾚﾄﾄﾄﾄﾄﾄﾄﾄﾄﾟ。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。ﾄﾄﾄﾄﾄﾄﾄﾄﾄﾂ
ｳ 1:       10:13      ｳ 2:                  ｳ 3:              ｳ
ｳ 4:                  ｳ 5:                  ｳ 6:              ｳ
ｳ 7:                  ｳ 8:                  ｳ 9:              ｳ
ｳ 10:                  ｳ Serial-No 008-0010007621 <00101>       ｳ
ﾀﾄﾄﾄﾄﾄﾄﾄﾄﾄﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾜﾄﾄﾄﾄﾄﾄﾄﾄﾄﾙ

* ASIC Build Number: 818, ExtMem: 16 KBytes formatted, page structure:
  * User: Raw =  128, Licensor =    0, OEM =    0, WIBU =  127.
  * Prot: Raw =  255, Licensor =    0, OEM =    0, WIBU =    0.
```

UserTypeのRawデータには100ページ（255-25-30-100=100）が割り当てられます。同様に、ProtectedTypeのRawデータには100ページ(255-0-0-155=100)が割り当てられます。
※ FormattedモードではUserType、ProtectedType共に最後のページがシステムの予約ページになります。従って最大利用ページ数はUserType、ProtectedTypeそれぞれ255ページになります。

OEMブロック、ＷＩＢＵブロック、Licensorブロックが不要な場合は、
WkCrypt　/PAU　/PXMF:L0
とすれば UserType、ProtectedType ともに全てRawデータとして割り当てられます。

ExtMemのアンフォーマット

ExtMemをアンフォーマットするために /PXMF－ を指定します。
C>wkcrypt /PA1 /PXMF- /L

[注意]
フォーマットまたはアンフォーマットを行なうと全ての記録内容が破棄されるか、少なくとも各々のブロックの最初のページが32 バイトの0で満たされます。また、フォーマットコマンドとアンフォーマットコマンドはWIPEコマンドを搭載していません。　従って、先頭のページ以外は上書きされずにデータが残ります。

2. WkCrypt での ExtMem の書き込み

「WKCRYPT.EXE」の /PXMWオプションでExtMemにデータを書き込むことができます。/PXMWオプションはコマンドラインまたは WKCファイルで使用することができます。ワイブキーを特定するための 、/PAオプション（port）、/PBオプション（特定のインデックス）、/PQオプション（同時に操作するワイブキー数）を併用することができます。
※「WKCRYPT.EXE」のGUIからもExtMemを読み書きすることができます。

「WKCRYPT.EXE」は、UserTypeまたはProtectedTypeで、Rawデータブロック とLicensorブロックへのページ書き込みだけをサポートします。

/PXMWオプションの書式は以下のとおりです。

WkCrypt /PXMW[指定ブロックとページ],{Data1,Data1,・・・DataN}
例. wkcrypt /PAU /PXMWPR10,{5,6,7,8} /PXMWL5:3,{″ABC″,″B″:2, ″C″:1}

以下に各パラメータについて説明します。

指定ブロック
指定するブロックは R（Rawデータ）と L（Licensor）です。
ProtectedTypeブロックを指定する場合は先頭にPを付けて　P R、　PL のように指定します。
Pを付けなければUserTypeブロックが対象になります。

ページ
書き込むページのページ番号を指定ブロックに続けて記述します。
例.　R10　　（Rawデータの10ページ）
また、複数の連続するページを指定する場合はコロン ： で区切って書き込みを繰り返す回数（ページ数）を記述します。
例.　　R10:3　　（Rawデータの10ページから3ページ分）

書き込みデータ
ページ番号に続けてカンマ　,　で区切って書き込みデータを指定します。
書き込みデータは {} で囲んで必要なだけカンマで区切って記述します。
例. {1,2,3}　01,02,03 という並びの3バイトのデータを指定
同じデータを繰り返す時は、繰り返すデータの後ろにコロン ：　をつけて繰り返す回数を指定します。
例.　　{1:2,2}　　01,01,02 を指定したのと同じです。

データーが文字列の場合はダブルクォーツで囲みます。
例.　　{"test","data"}　　"testdata" を指定したのと同じです。
またデータが文字の場合も前記の繰り返し規則が利用できます。
例.　　{"Test":2,"data"}　　"TestTestdata" を指定したのと同じです。

指定した書き込みデータが 1 ページ（32バイト）に満たない場合、指定したデータが繰り返し書き込まれます。

以下の例は、UserTypeのRawデータの 1 ページから 2 ページ分 01,02,03 を繰り返し書き込み、ProtectTypeのRawデータの 1 ページ目に "SuncarlaSuncarlaTest" という文字列を繰り返し書き込みます。　また　/V オプションを指定して書き込み状況を同時に表示します。

WkCrypt /V /PAU /PXMWR1:2,{1,2,3} /PXMWPR1,{"Suncarla":2,"Test"}

```
Version 5.00 of 2005-Apr-14 (Build 49) for Win32.
Copyright (C) 1989-2005 by WIBU-SYSTEMS AG. All rights reserved.

Used Format for Remote Programming: WIBU-SYSTEMS Control (WBC).
Port USB used for programming WIBU-BOXes.
Preparing of Programming WIBU-BOX no. 1 (serial 8-10007621) at USB:
  Start Writing ExtMem User Raw page 1 to 2.
  ExtMem User Raw page 1 (hexadecimal data) written:
  01 02 03 01.02 03 01 02:03 01 02 03.01 02 03 01
  02 03 01 02.03 01 02 03:01 02 03 01.02 03 01 02
  ExtMem User Raw page 2 (hexadecimal data) written:
  03 01 02 03.01 02 03 01:02 03 01 02.03 01 02 03
  01 02 03 01.02 03 01 02:03 01 02 03.01 02 03 01
  Start Writing ExtMem Protected Raw page 1.
  ExtMem Protected Raw page 1 (hexadecimal data) written:
  53 75 6E 63.61 72 6C 61:53 75 6E 63.61 72 6C 61
  54 65 73 74.53 75 6E 63:61 72 6C 61.53 75 6E 63
Completing of Programming WIBU-BOX no. 1 (serial 8-10007621) at USB:
  3 programming commands executed.
```

3. WkCrypt での ExtMem の読込みとリスト表示

「WKCRYPT.EXE」 の /PXMLオプションで 1 つ以上のExtMemページを読み込んで表示します。
/PXMLオプションはコマンドラインまたは WKCファイルで使用することができます。
ワイブキーを特定する為の 、/PAオプション（port）、/PBオプション（特定のインデックス）、/PQオプション（同時に操作するワイブキー数）を併用することができます。
※「WKCRYPT.EXE」のGUIからExtMemを読み書きすることができます。

「WKCRYPT.EXE」 は、UserTypeまたはProtectedTypeで、Rawデータブロック とLicensorブロックへのページ読み込みだけをサポートします。

/PXMLオプションの書式は以下のとおりです。

WkCrypt /PXML[指定ブロックとページ]
例. wkcrypt /PAU /PXMLR1:2 /PXMLPR1

以下に各パラメータについて説明します。

指定ブロック

指定するブロックは R（Rawデータ）と L（Licensor）です。ProtectedTypeブロックを指定する場合は先頭にPを付けてPR, PLのように指定します。Pを付けなければUserTypeブロックが対象になります。

ページ

読み込むページのページ番号を指定ブロックに続けて記述します。
例. R10 （Rawデータの10ページ）
また、複数の連続するページを指定する場合はコロン：で区切って終了ページを記述します。
例. R10:3 （Rawデータの 10ページから3ページ分）

4. ExtMem にアクセスするためのシグネチャーの取得

ProtectedTypeのExtMemにアクセスするためには8バイトの固有のシグネチャーが要求されます。シグネチャーは WkCryptの /POオプションと /PXMLオプション(読み込み用)、または /PXMWオプション(書き込み用)を組み合わせて指定することによって算出することができます。/POオプションの操作ではシグネチャーの表示が行なわれるだけで、ExtMemページへの書き込みは行なわれません。

以下は、Rawデータの10ページとLicenseの７ページから2ページ分の書き込みシグネチャー表示する例です(実際の書き込みは行なわれませんので指定する書き込みデータはダミーです)。

wkcrypt　/PAU　/PO　/PXMWPR10,{0} /PXMWPL7:2,{0}

```
wkcrypt - WIBU-KEY Encryption and Programming Tool.
Version 5.00 of 2005-Apr-14 (Build 49) for Win32.
Copyright (C) 1989-2005 by WIBU-SYSTEMS AG. All rights reserved.

WIBU-BOX signature output redirection activated, physical programming stopped.
WIBU-BOX 1: Start Writing ExtMem Protected Raw page 10.
WIBU-BOX 1: ExtMem Protected Raw page 10 written.
Used WIBU-BOX ExtMem write Access Signature: 47 42 73 202 58 41 36 145.
WIBU-BOX 1: Start Writing ExtMem Protected Licensor page 7 to 8.
WIBU-BOX 1: ExtMem Protected Raw page 62 written.
Used WIBU-BOX ExtMem write Access Signature: 47 42 73 202 58 41 36 145.
WIBU-BOX 1: ExtMem Protected Raw page 63 written.
Used WIBU-BOX ExtMem write Access Signature: 47 42 73 202 58 41 36 145.
```

以下は、同様のページに対して読み込みシグネチャーを計算した例です。

wkcrypt　/PAU　/PO　/PXMLPR10 /PXMLPL7:2

```
wkcrypt - WIBU-KEY Encryption and Programming Tool.
Version 5.00 of 2005-Apr-14 (Build 49) for Win32.
Copyright (C) 1989-2005 by WIBU-SYSTEMS AG. All rights reserved.

WIBU-BOX signature output redirection activated, physical programming stopped.
WIBU-BOX no. 1 (serial 8-10007621) at USB,
ExtMem Protected Raw page 10:
Used WIBU-BOX ExtMem read Access Signature: 47 42 73 202 58 41 36 145.
WIBU-BOX no. 1 (serial 8-10007621) at USB,
ExtMem Protected Licensor page 7 to 8 (Raw 62 to 63):
Used WIBU-BOX ExtMem read Access Signature: 47 42 73 202 58 41 36 145.
Used WIBU-BOX ExtMem read Access Signature: 47 42 73 202 58 41 36 145.
```

以下に、エントリ1に追加エントリのAddedDataを設定し、Modify Dependenceの設定で 「データコンテント」を指定したために書き込みシグネチャーが変化する例を挙げます。

Rawデータの1ページから2ページ分、書き込みシグネチャーと読込みシグネチャーを表示します。

wkcrypt /PAU /PO /PXMWPR1:2,{0} /PXMLPR1:2

```
Version 5.00 of 2005-Apr-14 (Build 49) for Win32.
Copyright (C) 1989-2005 by WIBU-SYSTEMS AG. All rights reserved.

WIBU-BOX signature output redirection activated, physical programming stopped.
WIBU-BOX 1: Start Writing ExtMem Protected Raw page 1 to 2.
WIBU-BOX 1: ExtMem Protected Raw page 1 written.
Used WIBU-BOX ExtMem write Access Signature: 61 242 130 169 77 3 88 183.
WIBU-BOX 1: ExtMem Protected Raw page 2 written.
Used WIBU-BOX ExtMem write Access Signature: 110 21 83 121 29 147 53 22.
WIBU-BOX no. 1 (serial 8-10007621) at USB,
ExtMem Protected Raw page 1 to 2:
Used WIBU-BOX ExtMem read Access Signature: 180 198 94 15 247 6 226 152.
Used WIBU-BOX ExtMem read Access Signature: 180 198 94 15 247 6 226 152.
```

ExtMemの1ページ目と2ページ目に異なるデータが格納されていて、Modify Dependenceの設定で「データコンテント」が指定された為、2つのページで書き込みシグネチャーが異なっています。
一方、読込みシグネチャーはModify Dependenceの設定に依存しないため変化しません。

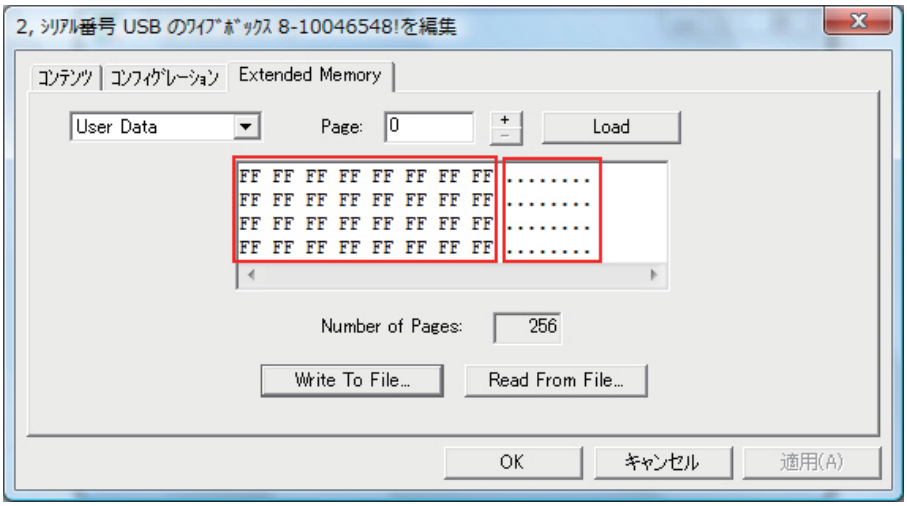
※ /Lオプションを併用する場合は、 /POの前に指定して下さい。
※ USBを特定するための /PAUパラメータは一番先頭に指定して下さい。
例. WkCrypt /PAU /L /PO /PXMLPL10:4

8-9. WkList32 での ExtMem の操作

「WKCRYPT.EXE」 のGUIを使ってExtMemの読み書きをすることができます。以下にWkList32でExtMemの編集をする手順を説明します。

尚、WkList32はExtMemにRawモードでアクセスしますので、ExtMemのモードに関わらず読み書きすることができますが、必要に応じて事前に「WKCRYPT.EXE」 でExtMemのフォーマットを行なってください。また、ProtectedTypeブロックにアクセスするためにはエントリ 1 にFirmCode/UserCodeがプログラミングされている必要があります（FirmCode10以外をプログラミングした場合は、ProtectedTypeのExtMemにアクセスする時は必ずFSBが必要になります）。

(1) 「WKCRYPT.EXE」で編集画面に入り、「Extended Memory」ページを表示してください。以下のExtMem操作画面が表示されます。



(2) 以下の手順でExtMemのページデータの編集を行って保存してください。

1. ドロップダウンリストで、UserTypeまたはProtectedTypeを選択。
2. ＋　ボタン、－　ボタン で、編集するページ番号を指定。
3. バイナリ編集エリア、または、テキスト編集エリアでページデータを編集。
4. 「OK」ボタン、または「適用」ボタン　で編集データをExtMemに保存。

【データ編集】
バイナリデータを編集する時は、「バイナリデータ編集エリア」をクリックして、キーボードからバイナリデータを直接編集して下さい。テキストデータを編集する時は、「テキストデータ編集エリア」をクリックして、キーボードからテキストデータを入力してください。

但し、テキストデータ編集エリアの表示は 1 バイト単位のASCIIデータのみをサポートします。日本語入力を行なう場合、以下の点にご注意ください。

※入力された日本語データはテキストデータ編集エリアに正常に表示されません。入力されたデータは有効です。従って、アプリケーションから読み出して日本語データとして利用するのは問題ありません。

※WkList32でのテキストデータ編集はページ単位で行われますので32バイトを越えるデータの入力はできません。32バイトを越えるデータは無条件で切り捨てられますので、日本語データ等の2バイトで構成されるデータを入力する場合は最後のデータ部分にご注意ください。

【データ保存】
編集データの保存は、ページの編集が完了する毎に「適用」ボタンで保存するようにして下さい。
※ UserType（またはProtectedType）でページデータを編集した後、保存しないまま続けて他のブロックに移動した場合、アプリケーションエラーによって WkList32が終了することがありますので確実に保存を行ってください。

【ファイルへの書き出し】
ExtMemのメモリイメージをファイルに保存することができます。データは、ページ番号の指定にかかわらず常にページ0から全ページ分（8Kbytes）が保存されます。
（ProtectedTypeのExtMemを操作する場合はFSBが必要です。）

【ファイルから読み込み】
任意のデータファイルを指定してExtMemに読み込むことができます。ファイルの読み込みも書き出しと同様、常にページ0からの読み込みとなります。

ファイルからデータを読み込んでExtMemのページデータを編集する場合、ページをまたがったデータを読み込むことができますので、日本語テキストを保存する場合には大変便利です。但し、読込みデータによって以下の制約がありますのでご注意ください。

※ファイルのサイズは16バイト以上必要です（正確には1ページについて16バイト以上のデータが必要です）。

※読み込まれたデータはページ単位（32バイト単位）で0ページから順にマッピングされます。

※最後のデータが、1 ページ（32バイト）に満たない場合、そのページの残りはゼロで埋められます。

※最後のデータが読み込まれると読込み動作は終了します。従って、最後のページより後のページのデータは変更されずにそのまま残ります。

※最後のページにおいて、読み込まれたデータが16バイト未満の場合は、(1)の制約によりエラーが発生しますのでご注意ください。つまり、ファイルサイズを32で割った余りが、16バイト以上必要です。この問題はファイルサイズを16バイトで割り切れるサイズに調整することによって回避することができます。

8-10. サンプルプログラムについて

ワイブキーCD の¥Program Files¥WibuKey¥DevKit¥SampleフォルダにExtMemを利用するためのサンプルソースが収録されています。ExtMemのためのサンプルは各言語サンプルのClassic¥WkDmExtMemフォルダをご覧ください。

参考までに以下に、エントリ 1 にプログラミングされた特定のFirmCode/UserCodeを指定してLocal接続のワイブキーのチェックを行ない、見つかったワイブキーのポート番号を使ってExtMem を読み出す場合の手順について簡単に説明します。

(1)　FirmCode/UserCode を指定し ローカルハンドル (HWKB_LOCAL) を使って WkbOpen2を呼び出します。

(2) WkbOpen2 で該当するFirmCode/UserCode が見つかった場合は、WkbOpen2で得られたハンドルを使って WkbQueryEntry2 を呼び出します。

(3)　WkbQueryEntry2 が成功すると WKBOPENENTRY構造体にWkbListBox2　を呼び出すために必要な情報が返されます。

WKBOPENENTRY. lPort　　・・・　見つかったワイブキーのポート番号。
WKBOPENENTRY. wkbsr　　・・・　見つかったワイブキーのWKBSERIAL構造体。

(4) WKB_LIST_CONFIGパラメータを指定して WkbListBox2 を呼び出します。WkbListBox2 の呼び出しが成功するとWKBBOXCONFIG構造体にExtMemに関する情報が返されます。

WKBBOXCONFIG. usAsicBuild　　・・・・・ ASIC のバージョン番号。
WKBBOXCONFIG. usExtMemSize　・・・・・　ExtMemのサイズ (最大ページ数)。

usExtMemSizeがゼロの場合はExtMemを利用できません。

(5) ExtMemが利用できる場合、WKB_LIST_EXTMEMパラメータを指定してもう一度　WkbListBox2 を 呼び出します。WkbListBox2の呼び出しが成功するとWKBEXTMEM構造体のabData に 32バイトの ExtMemページデータが返されます。(WkbOpen2で取得したハンドルは適当なタイミングで WkbClose2によって閉じてください。)

尚、参考資料としてワイブキーを特定する際のポート番号の対応表を掲載します。アプリケーションからポート番号を直接指定する場合などに参考にしてください。

Port	Port 番号	標準規定
LPT1	#0	1 or L1
LPT2	# 1	2 or L2
LPT3	#2	3 or L3
LPT4	#3	4 or L4
COM1	#8	C1
COM2	#9	C2
COM3	#10	C3
COM4	#11	C4

Port	Port 番号	標準規定
PcCard1	#32	P1
PcCard2	#33	P2
PcCard3	#34	P3
PcCard4	#35	P4
PcCard5	#36	P5
PcCard6	#37	P6
ADB	#40	ADB
USB	#48	U

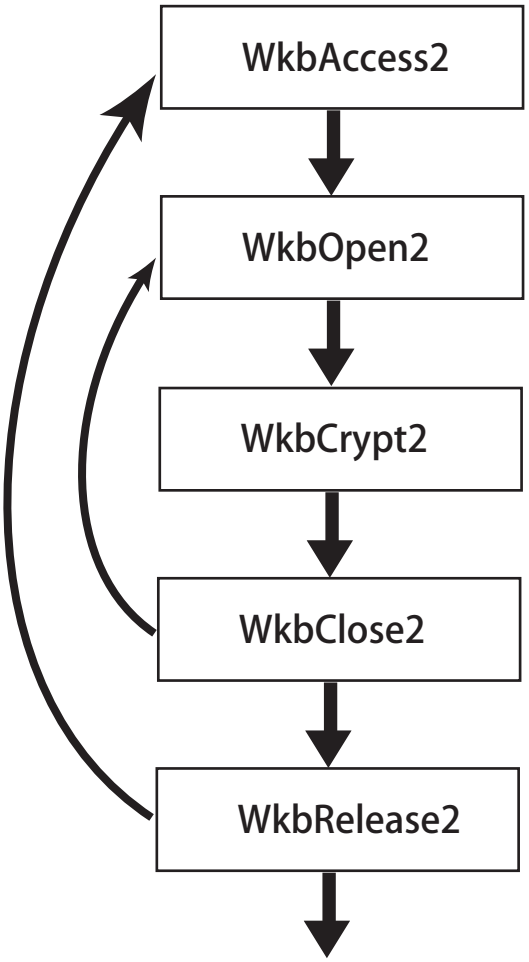
Chapter 9

WIBU-KEY API ファンクションについて

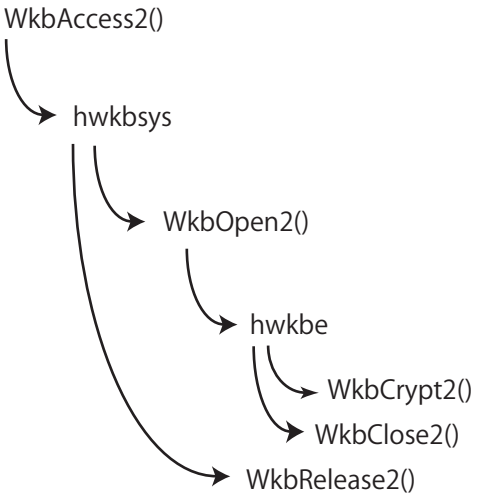
9-1. WIBU-KEY API ファンクション

ワイブキーには、多数のAPIファンクションが用意されています。これらのAPIファンクションを、貴社のソースコードに直接組み込むことにより、きめの細かいセキュリティを実現することができます。APIファンクションを組み込んだ後は、自動暗号化ツールAxProtectorを使って、ファイル全体を暗号化することも可能です。逆コンパイラやデバッガによる解析からファイルを守るためにも、自動暗号化ツールAxProtectorとの併用をお勧めいたします。

基本的なWIBU-KEY APIの呼び出しスキームは下記の図のようになります。



ハンドルによるアドレッシングは右図のようになります。WkbAccess2ファンクションを使って、ローカルまたはネットワーク上のWIBU-BOXにアクセスし、WkbOpen2ファンクションを使って、WIBU-BOXのエントリにアクセスします。WkbOpen2ファンクションで取得したWIBU-BOXのハンドルをベースにWkbCrypt2やWkbListBox2、WkbProgram2などの各ファンクションを実行します。処理終了後は、WkbClose2ファンクションを使って、WkbOpen2でオープンしたWIBU-BOXエントリをクローズし、WkbRelease2を使って、WkbAccess2でオープンしたWIBU-BOXをクローズします。各ファンクションについての詳細は、「Chapter 15 WIBU-KEY APIファンクション一覧」をご参照ください。



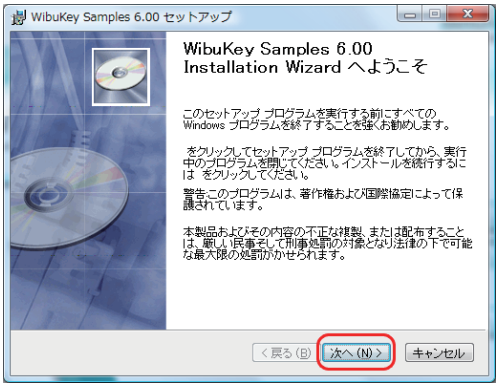
9-2. サンプルプログラムのインストール方法について

ワイブキーCDの中には、ワイブキーのAPIファンクションを利用したサンプルプログラムが開発言語ごとに入っています。APIファンクションを利用される方は、ぜひ参考にしてください。サンプルプログラムのインストールの方法は、ワイブキーCDから次のようにインストールします。

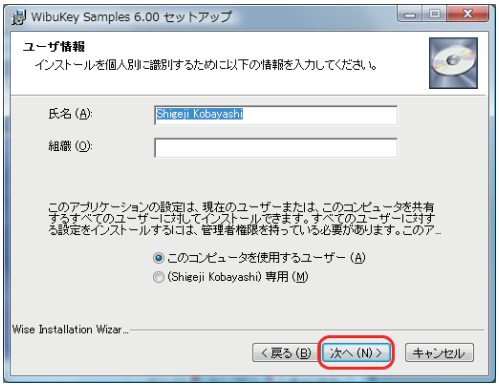
①ワイブキーCD Ver7.11を立ち上げ、「English」をクリックします。

②「Wibukey Samples」をクリックします。

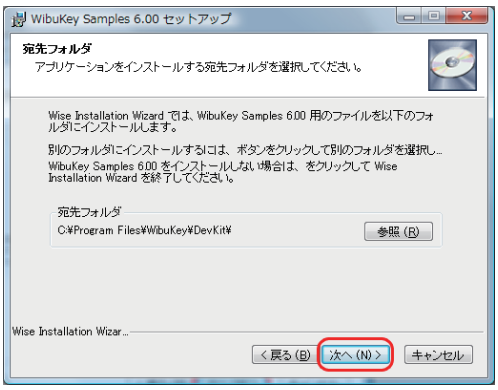
③「WIBU-KEY Samples 6.00 Installation Wizard へようこそ」画面で、[次へ]をクリックします。



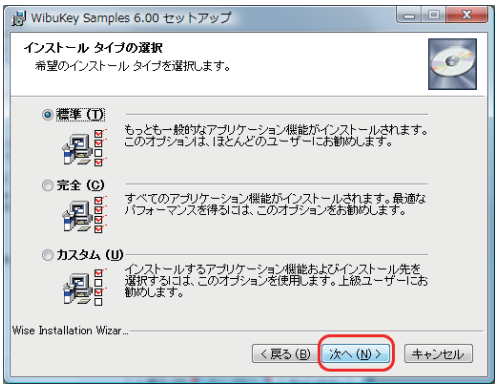
④ ユーザー情報を入力します。



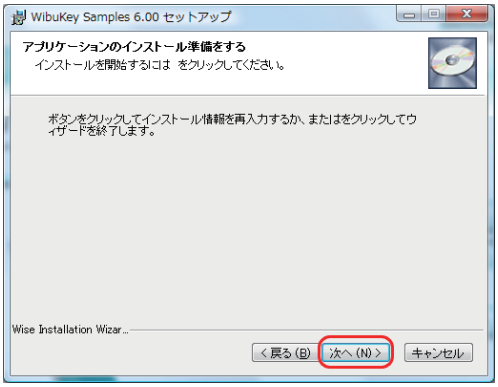
⑤ インストール先フォルダを選択します。



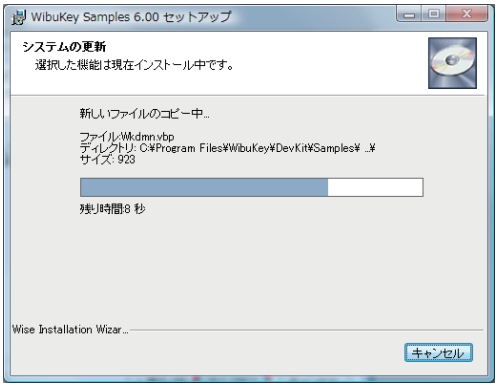
⑥ インストールタイプを選択します。



⑦ インストールを開始するには「次へ」をクリックします。

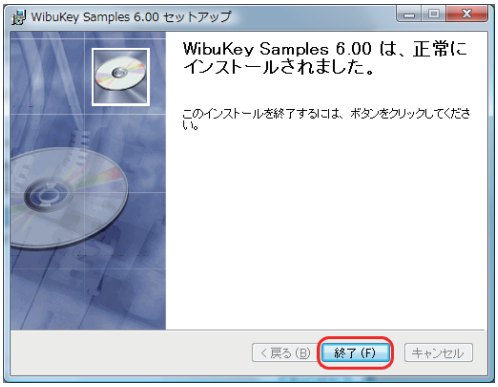


⑧ インストールが開始されます。



⑨「WIBU-KEY Samples 6.00 は、正常にインストールされました」の画面が表示されたらインストールは終了です。[終了]をクリックして画面を閉じて下さい。

¥Program Files¥WibuKey¥DevKit¥Samplesフォルダにサンプルプログラムがインストールされます。開発言語ごとに、APIファンクションの使い方を参考にしてください。



Chapter 10

WIBU-KEY COM コンポーネントについて

10-1. WIBU-KEY COM コンポーネントについて

ワイブキーには MicrosoftのCOM標準と完全互換の WIBU-KEY ActiveXコンポーネントが用意されています。WIBU-KEY ActiveXコンポーネントは、Microsoft VisualC++、VisualBasic、Borland Delphi、VBScript、JavaScript等でそのまま使用することができます。WIBU-KEY ActiveXコンポーネントは、COMインターフェースによってほとんど完全なWIBU-KEY APIをサポートしています。

WIBU-KEY COM/ActiveXコンポーネントは“Wibukey.dll”にインプリメントされています。
“Wibukey.dll”はCOM APIとCOM APIクラスとして振る舞いますが、ワイブキーへの物理的なアクセスは、“WkWin32.dll”の従来のWIBU-KEY APIで実現されます。

ワイブキーの COM APIを利用する為にはWIBU-KEY APIのVersion3.00以上が必要です。

10-2. ワイブキーのクラスについて

WIBU-KEY ActiveXコンポーネントには完全な COM APIが格納されています。
このコンポーネントは、Strings、Integer、Boolean等の単純な型タイプと、コンポーネントによって定義される複雑なオブジェクト・クラス・タイプを使用します。

定義されているオブジェクト・クラス・タイプは以下のとおりです。

WibuKeyクラス・・・・・・・・・・基本コンポーネント。
Encryptionクラス・・・・・・・・暗号化／復号化の情報とデータ関係。
BoxEntryクラス ・・・・・・・・・WIBU-BOXエントリ関係。
AddedEntryクラス・・・・・・・・WIBU-BOX 追加エントリ関係。
WIBU-BOXクラス・・・・・・・・・・WIBU-BOXの記述指定関係。
SubSystemクラス ・・・・・・・・WIBU-KEYサブシステムの記述指定関係。
RemoteProgrammingクラス・・・リモートプログラムの操作関係。
Versionクラス・・・・・・・・・・コンポーネント、ドライバーのバージョン関係。

ActiveXコンポーネント内のプロパティはすべて自動的に管理（＝Object Binding）され、プロパティの内容の変更は自動的に他のいくつかのプロパティに反映します。

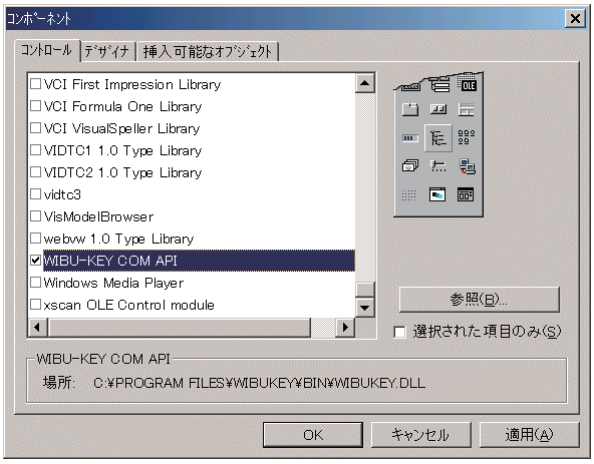
10-3. WIBU-KEY ActiveX コントロールの使い方

ワイブキーランタイムキットをインストールすると、自動的にWIBU-KEY ActiveXコンポーネントがインストールされます。以下に、WIBU-KEY ActiveXコンポーネントの使い方をVisualBasicの例で説明します。

（1）VisualBasicを起動し、テスト用の新しいプロジェクト（標準EXE）を作成してください。

（2）WIBU-KEY ActiveXコンポーネントを取り込みます。

【プロジェクト(P)】→【コンポーネント(O)】の「コントロール」タブでWIBU-KEY COM APIにチェックを入れて、[適用(A)]ボタンをクリックします。



ツールボックスにWIBU-KEYコンポーネントが追加されます。

WIBU-KEYコンポーネントが追加されたらコンポーネントウィンドウを閉じて下さい。

これでVisualBasicから何時でも WIBU-KEY ActiveX コントロールを使用することができます。

実際にワイブキーをチェックしてみます。

WIBU-KEYコンポーネントをフォームに配置します。Commandボタンを 1 つ配置して、CommandボタンのClickイベントに以下の4行を記述してください。

```
Private Sub Command1_Click ()

   Dim LastError As Long
   Wibukey1.CheckBox
   LastError = Wibukey1.LastErrorCode
   MsgBox(Wibukey1.LastErrorText)

End Sub
```

次に、WIBU-KEYコンポーネントのプロパティに、FirmCodeとUserCode を設定します。
以上が完了したら【完全コンパイル後に実行(F)】で実行してみてください。
Commandボタンをクリックすると、ワイブキーのアクセスが行われ結果が表示されます。

万一、期待した結果が得られない時は、ワイブキーの装着状況、FirmCode、UserCode等に間違いがないかご確認ください。

デフォルトではLocalのワイブキーだけが対象になります。ネットワーク上のワイブキーサーバーもアクセスの対象にする場合は、UsedSubSystemsプロパティに3（Local + WkLAN）、RestrictedUserQuantityModeプロパティにTrue を指定してください。また、その他のプロパティでネットワークの動作を更にコントロールすることができます（プロパティの詳細については後述します）。

以下に、CommandボタンのClickイベントで使用したメソッドについて簡単に説明します。

CheckBoxメソッド

WIBU-KEYコンポーネントのプロパティの値に従ってワイブキーにアクセスを行ないます。
戻り値はありません。

LastErrorCodeメソッド

最後に行われた動作（直前の動作）の結果を返します。戻り値は Long型です。
LastErrorCodeで返された値が 0 の時、ワイブキーへのアクセスは成功しています。
ゼロ以外の時は エラーコード を表します。CheckBoxメソッドは結果を返しませんので、プロテクトの判定はLastErrorCodeの戻り値で行ないます。

LastErrorTextメソッド

LastErrorCodeと同様、最後に行われた動作の結果を返しますが戻り値は String型です。
前記のサンプルでは、結果を表示するだけの目的で呼び出しています。

以上で WIBU-KEY ActiveXコンポーネントの基本的な使い方は終了です。

※各言語用のサンプルプログラムが ¥Program Files¥WibuKey¥DevKit¥Samplesフォルダにありますのでご覧ください。ActiveXコンポーネント用サンプルのフォルダ名にはそれぞれ最後に .COM がついています。

10-4. WIBU-KEY コンポーネントのプロパティ

WIBU-KEY ActiveXコンポーネントの設計時プロパティについて説明します。
これらのプロパティは当然、実行時にもアプリケーションから任意に変更することができます。

AlgorithmVersion (Long, read/write) Default: 2

WIBU-KEY algorithmを指定。

1：WIBU-BOX/1, WIBU-BOX/3 または WIBU-BOX/4（旧ワイブキー）
2：WIBU-BOX/3 または WIBU-BOX/4（標準）
3：WIBU-BOX/5
4：WIBU-BOX/6（将来のために予約）

ApplicationName (String, read/write) Default: Empty String

ワイブキーサーバーにアクセスする時のアプリケーション名。ApplicationNameはアプリケーションID等の情報表示で使われます。例えば、ネットワークにアクセスした時、ワイブキーサーバー側の情報表示画面ではクライアントのコンピュータ名に加えてこのApplicationNameが表示されます。設定しなくても動作上は問題ありませんが、クライアントPCに複数のアプリケーションが存在する時、サーバー側の情報表示からはどのアプリケーションがアクセスしているのか判断できなくなります。

AutoCancelMode (Boolean, read/write) Default: False

ワイブキーサーバーにアクセスする際のログオンの排他制御を指定します。True（=有効）にすると、すでにワイブキーサーバーにログインしているアプリケーションを重ねて起動することができなくなります。同一アプリケーションの2重起動が禁止されますので、接続可能なクライアント数を無駄に消費するのを防止することができます。Falseの場合、起動したアプリケーションの数だけ接続クライアント数が消費されます。

DialogOptions (Long, read/write) Default: wkDlgNormal (0)

ワイブキーシステムによって表示されるダイアログボックスの表示方式を指定します。同時に複数指定することができます。

0：wkDlgNormal (0)
標準モード。[キャンセル]ボタンが付いたワイブキー検索ダイアログボックスを表示します。
1：wkDlgNoCancel (1)
[キャンセル]ボタンが付かないワイブキー検索ダイアログボックスを表示します。
2：wkDlgNoScanDialog (2)
ワイブキー検索ダイアログボックスを表示しません。
4：wkDlgNoVersionError (4)
指定したバージョンが高すぎるために表示されるバージョンエラーダイアログの表示を行ないません。
8：wkDlgNoHelpButton (8)
情報ダイアログボックスに[ヘルプ]ボタンを表示しません。
16：wkDlgNoPanelButton (16)
情報ダイアログボックスに[設定]ボタン（WIBU-KEYコントロールパネル）を表示しません。

EncryptMode (Long, read/write) Default: wkEncDirect (0)

Encryption/Decryptionモード。

0：wkEncDirect (0)
1：wkEncIndirect (1).

ExclusiveAccessMode (Boolean, read/write) Default: False
ワイブキーへのアクセスに対する排他制御を行ないます。Trueにするとアプリケーションの2重起動ができなくなります。falseの時は無効です。

ExpirationDateCheckMode (Boolean, read/write) Default: False
有効期限チェックの指定。Falseの時、有効期限チェックは無効です。暗号化クラスのSelectionOptions特性のビット1と同じです。

FirmCode (Long, read/write) Default: 0 (Invalid)
アクセスするワイブキーのFirmCodeを指定します。

LimitCounterDecrementMode (Boolean, read/write) Default: False
LimitCounterの指定。Falseの時、LimitCounterは無効です。暗号化クラスのSelectionOptions特性のビット0と同じです。

RestrictedUserQuantityMode (Boolean, read/write) Default: False
ワイブキーをネットワークで使用する時、同時に接続が可能なクライアント数を制限するかどうかを指定します。このプロパティは、EncryptionクラスのSelectionOptions特性のビット2と同じです。Falseにするとクライアント制限が無効になります。Falseの場合は、ワイブキーサーバーへのLogonとConnectionは行われますがクラスタのslotを消費しなくなります。

ServerName (String, read/write) Default: Empty String
ワイブキーサーバーのマシン名、または IPアドレスを指定します。指定しない（空白）の場合は使用可能なWIBU-KEYサーバーが自動検索されます。

StockData (String, read/write) Default: Empty String
Encryptionウイザードで設定した暗号化データ。暗号化データはDecryptStockDataメソッドで復号化し、結果をTextDataプロパティで受け取ることできます。

SubSystemVersion (Long, read/write) Default: 252
サブシステムのバージョン。値は、メジャーバージョンを100倍したものにマイナーバージョンを加えたものです。(例えばVersion2.52の場合は 252)

TextData (String, read/write) Default: empty string
Encryption/Decryptionされる文字列データとその結果。

UsedSubSystems (Long, read/write) Default: wkAccLocal (1)
アクセスするWIBU-KEYサブシステムを指定。同時に複数指定することができます。

1：wkAccLocal (1)
2：wkAccWkLAN (2)
4：wkAccWkNet (4)

UserCode (Long, read/write) Default: 0 (Invalid)
アクセスするワイブキーのUserCodeを指定します。

UserIdentification (Long, read/write) Default: 0
指定したサーバーのスロットを割り付けるためのユーザー識別。0を指定した場合、空いている最初のエントリが使われます。WkLANでは0 を指定してください。

WkBoxSimUsed (Boolean, read/write [設計時], read only [実行時])
いくつかのオペレーション用のモードを指定します。COMドライバーのランタイムモード（=実行時）では、このフラグは常にFalseです。開発時このフラグはデフォルトでTrueにセットされていますが、Falseに設定してランタイムモードをシミュレートすることができます。

MaxStartDelta (Long, read/write) Default: 70
WkNETサブシステムで使用。

RefreshTimeDelta (Long, read/write) Default: 30
WkNETサブシステムで使用

WkNetEncryptSequence (String, read/write) Default: Empty String
WkNETサブシステムで使用。

WkNetFileName (String, read/write) Default: Empty String
WkNETサブシステムで使用。

WkNetVersion (Long, read/write) Default: 200
WkNETサブシステムで使用。

Chapter 11

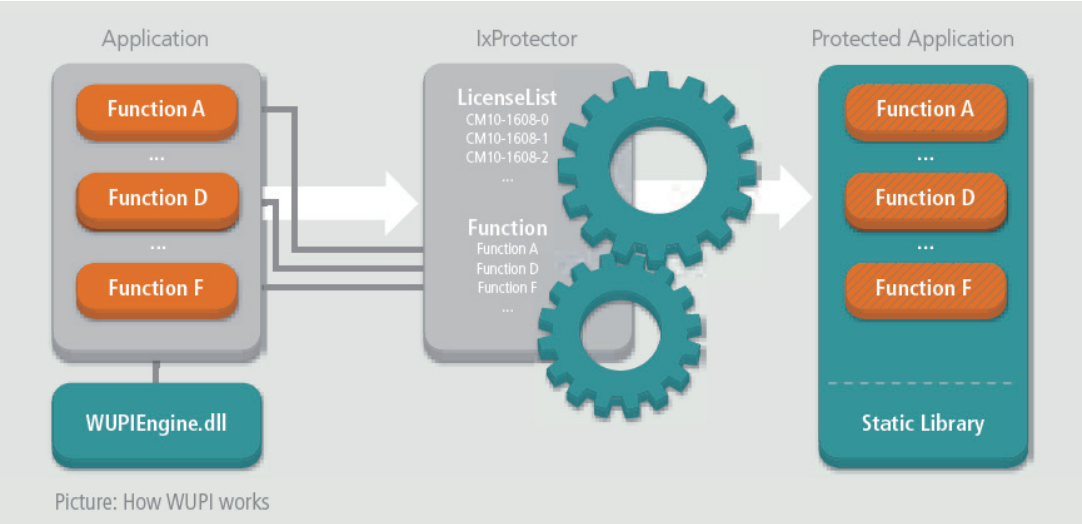
IxProtector について

11-1. IxProtector とは

ワイブキーの自動暗号化ツールAxProtectorや一般的な暗号化ツールでプログラムファイルを暗号化した場合、ディスク上では暗号化された状態を維持します。ただし、プログラムファイルが動作するために一度PCメモリーにロードされ、プロテクトチェックが通過した後は、暗号化されていたプログラムファイル（コード）はメモリー上で復号化された状態になります。ワイブキーの場合は、デバッガー解析などの攻撃に耐える強力なセキュリティ機能を備えておりますが、メモリー上でプログラムコードが復号化されていること自体がセキュリティホールの 1 つになります。

IxProtectorは、この問題を解決しました。メモリー上で展開されるプログラムコードを常に暗号化しておき、必要な時に必要なモジュールを復号化し、実行したあとは再び暗号化しておくという、メモリー上での「オンデマンド復号」を実現する新機能です。AxProtectorで暗号化されたプログラムコードが、メモリー上でも常に暗号化されているため、クラッキングに対して非常に強力なセキュリティを実現することが可能になります。

IxProtectorによる「オンデマンド復号」セキュリティを実現するには、WUPI(Wibu Universal Protection Interface)ファンクションをソースコードに組み組み、ファンクションモジュール単位で暗号化・復号化を行います。



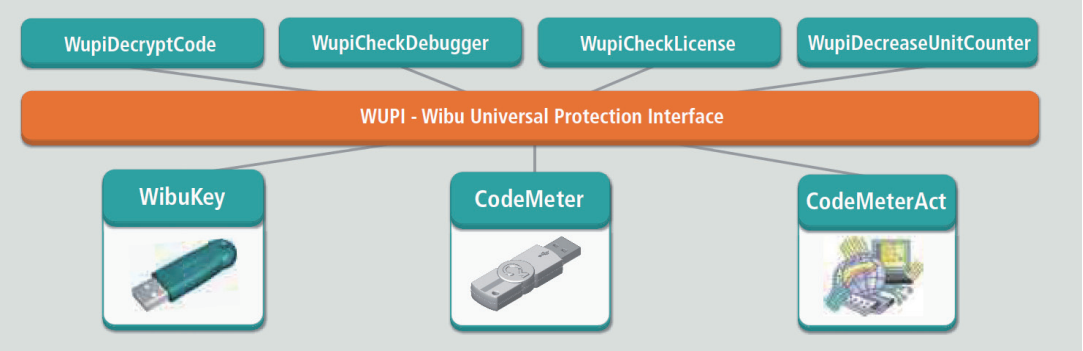
Picture: How WUPI works

[NOTE]
C#やVB.NET等で作成したMicrosoft .NET Framework対応プログラム（マネージコード）の場合は、IxProtector/WUPIを使用しなくても、AxProtector(.NET アセンブリ)で暗号化することで自動的に「オンデマンド復号」機能が組み込まれます。ソースコードを修正する必要がありません。IxProtector/WUPIをソースコードに組み込む必要があるのは、VC++やVB6などのネイティブコード（アンマネージコード）の場合です。

11-2. WUPI ファンクションについて

WUPI（ウーピィ）とは、Wibu Universal Protection Interfaceの略で、ワイブキー(Wibukey)、コードメータ(CodeMeter)、コードメータAct(CodeMeterAct)に共通に使用できるユニバーサルなAPIファンクションです。WUPIファンクションで取得したハンドルを、ネイティブの各APIファンクションに渡してWUPIファンクションと従来のAPIファンクションを連携して使用することも可能です。

さらに、WUPIファンクションによって暗号化されるファンクションモジュールは、最新バージョンのAxProtectorツールで暗号化し直すことにより、ソースコードを変更せずに、常に最新の暗号化セキュリティ技術でモジュールをプロテクトすることが可能になります。



11-3. WUPI ファンクション一覧

WUPIファンクションの一覧です。各WUPIファンクションの詳細につきましては、後述の「WUPIファンクション詳細」をご参照ください。

WupiAllocateLicense
指定したライセンスリストのライセンスを割り当てます。

WupiFreeLicense
割り当てたライセンスを解放します。

WupiGetHandle
エントリのネイティブハンドルを返します。
CodeMeterの場合はHCMSysEntryに、WIBU-KEYの場合はHWKBENTRYに指定されます。

WupiEncryptionCode
WupiEncryptionCodeはファンクションを暗号化します。暗号化するファンクションはIxProtector設定で指定する必要があります。

WupiDecryptCode
WupiDecryptCodeはファンクションを復号します。復号するファンクションはIxProtector設定で指定する必要があります。

WupiDecreaseUnitCounter
CodeMeterの場合はUnit Counter（ユニットカウンター）、WIBU-KEYの場合はLimit Counter（リミットカウンター）を指定された数値分減らします。

WupiQueryInfo
使用中のライセンス情報を返します。

WupiGetLicenseType
ライセンスタイプを返します。

WupiCheckDebugger
プロテクトされたアプリケーションに対して、デバッグ処理が施されたかどうかをチェックします。

WupiCheckLicense
与えられたライセンスリストからライセンスをチェックします。使用されるセキュリティチェックは実行中に変化します。このファンクションは、ライセンスを自動的に割り当てます。

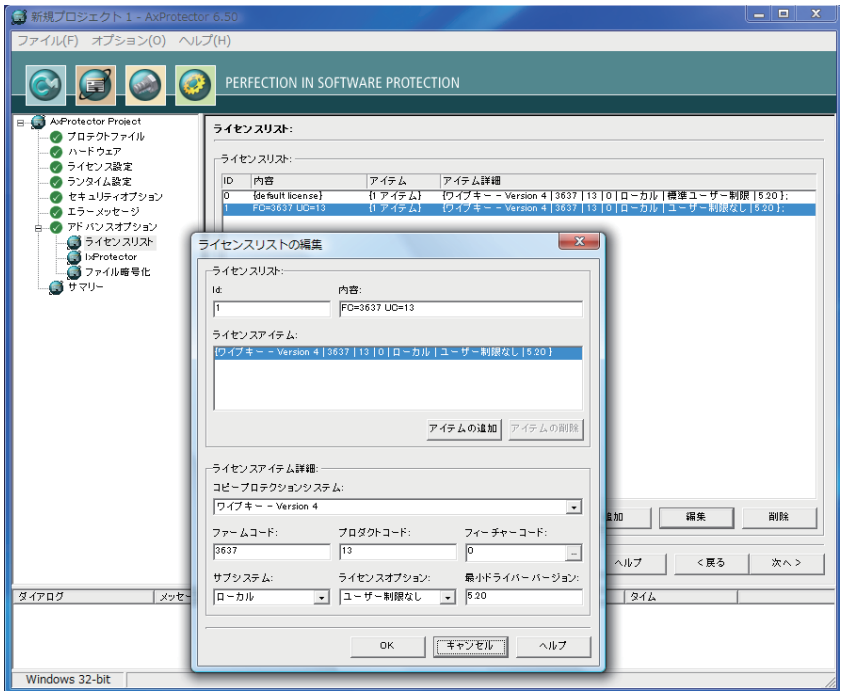
WupiGetLastError
直前に実行したWUPIファンクションのエラーコードを返します。

11-4. WUPI ファンクションの使い方

WUPIには、インデックスベースWUPI(Index Based WUPI)とポインタベースWUPI(Pointer Based WUPI)の２種類があります。インデックスベースWUPIは、ほとんどすべての言語から呼び出すことが可能で、プログラム開発とライセンスモデルを切り離して管理することが可能です。ポインタベースWUPIは、C/C++のようにポインタを指定しながら開発します。将来性を考え、できる限りインデックスベースWUPIのご使用を推奨致します。以下は、インデックスベースのWUPIの使い方をご説明します。

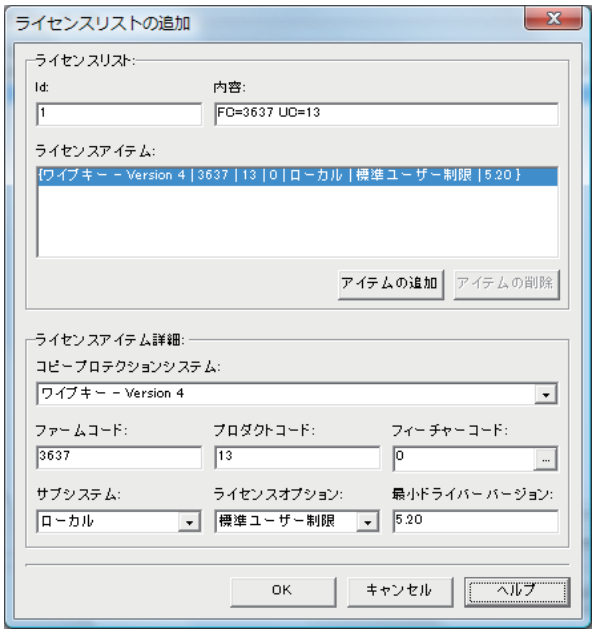
1. ライセンスリスト (License List) について

WUPIファンクションはライセンスリスト(License List)を参照しながら動作します。このライセンスリストは、自動暗号化ツールAxProtectorの「アドバンスオプション」の「ライセンスリスト」画面で作成・編集します。

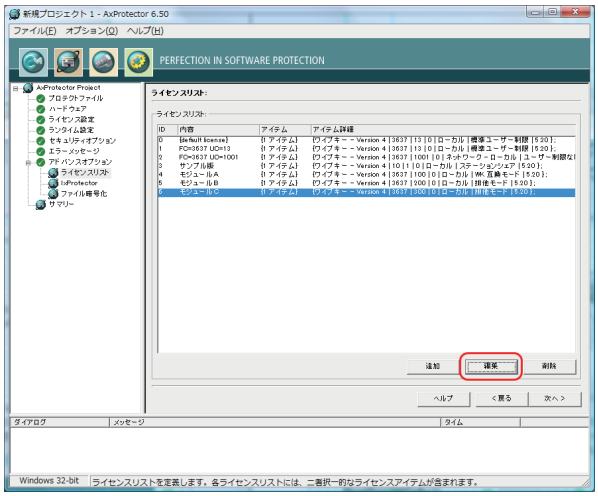


① ライセンスリストを作成する

「ライセンスリストの追加」画面で、インデックスベースのライセンスを作成します。Id=1は、Index=1を意味します。"内容"には、ライセンス名称を明記します。下段の「ライセンスアイテム詳細:」には、使用するハードウェアキー（コピープロテクションシステム）、ファームコード、ユーザーコード（プロダクトコード）などのライセンス内容を設定します。



② ライセンス内容を追加する



必要に応じて「ライセンスリストの追加」画面でライセンス内容を追加します。また、すでに登録されたライセンスは「編集」ボタンをクリックして、「ライセンスリストの編集」画面で編集できます。

ここで作成されたライセンスリストが、WUPIファンクションから参照されます。

このライセンスリストにアクセスする最初のWUPIファンクションが "WupiCheckLicense()"です。

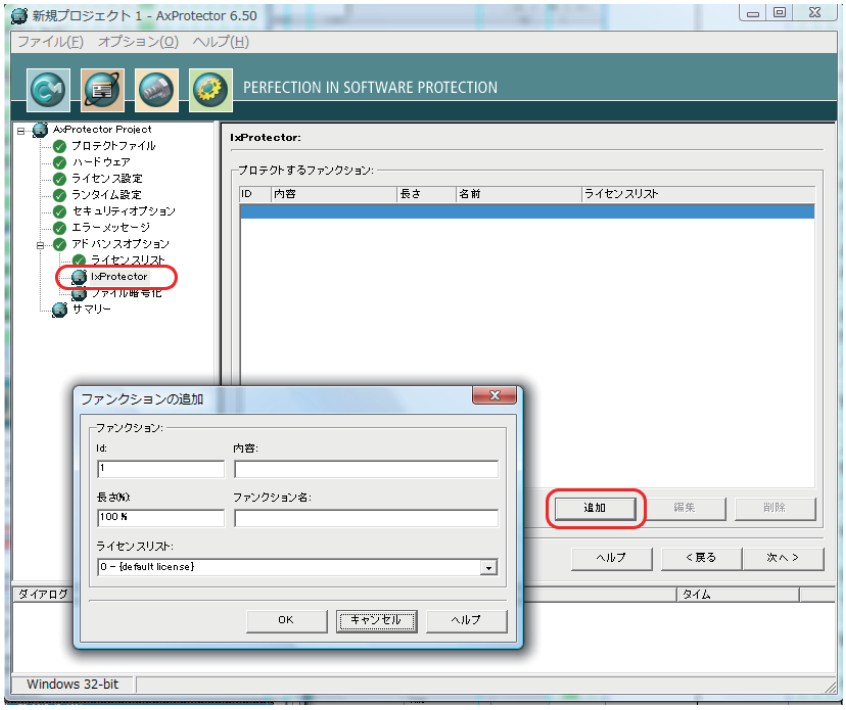
例えば、プログラムから、

WupiCheckLicense(1)

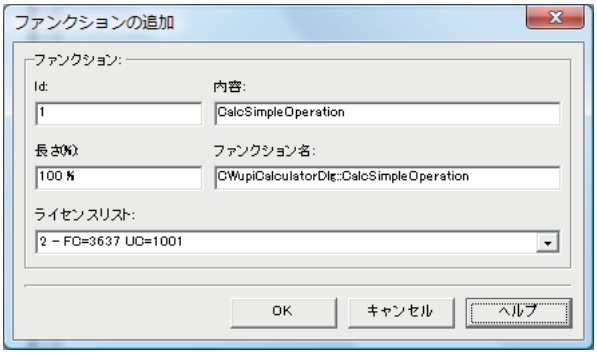
を実行すると、ライセンスリスト上のId=1（Index=1)にアクセスし、ライセンス内容を取得し、その内容に応じたセキュリティチェック（ハードウェアキーチェック）を行います。チェックエラーの場合はFALSE(0)を返し、それ以外はTRUE(1)を返します。

2.「ファンクション」インデックスリストについて

次に、「ファンクション」インデックスリストを作成します。「ファンクション」インデックスリストとは、IxProtectorで暗号化するファンクション（モジュール）をリスト登録したものです。このリストに登録されたファンクション（モジュール）が実際にIxProtectorで暗号化・復号化されます。ファンクションを登録するには、「ファンクションの追加」画面で行い、ソースコード内で実際に使用されているファンクション名を正確に入力し、使用するライセンス内容をライセンスリストから指定します。

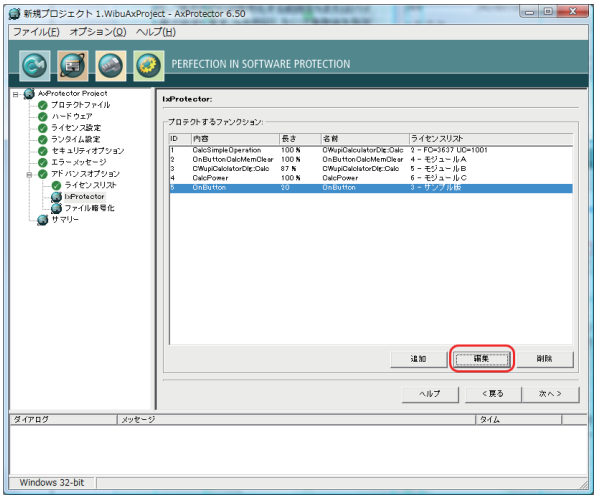


① 暗号化するファンクションを登録する



「ファンクションの追加」画面で、暗号化するファンクションを登録します。"Id"は、登録されたファンクションを識別するためのId番号で、自動的に連番が割り当てられます。"内容"には、ファンクション内容を明記し、"長さ(%)"には暗号化する範囲を%またはバイト数で設定します。%を明記しないで整数値を指定すると、指定されたバイト数を暗号化します。"ファンクション名"には、ソースコードで使用されているファンクション名を忠実に明記してください。"ライセンスリスト"では、このファンクションが使用するライセンス内容をライセンスリストから選択します。

② ファンクションを追加する



暗号化するファンクションを追加します。すでに登録されたファンクションは「編集」ボタンをクリックして、「ファンクションの編集」画面で編集できます。

ここで定義されたファンクションモジュールは、IxProtectorで暗号化され、メモリー上でも暗号化された状態になります。実際に、ファンクションを実行するためには、そのファンクションが呼び出される直前にWupiDecryptCode()ファンクションを使って復号化する必要があります。

例えば、上記①で、CalcSimpleOperationファンクションをファンクションId=1（ライセンスリスト=2/FC=3637,UC=1001）で定義しましたが、実際に使用するには、WupiDecryptCode()ファンクションのパラメータにファンクションID=1を入れて実行します。

WupiDecryptCode(1)

上記 1 行を実行することで、ライセンスリスト=2に登録されたFC=3637/UC=1001を持つワイブキーを検索し、見つかった場合に、暗号化されているCalcSimpleOperationファンクションモジュールをメモリー上で復号化します。（該当するワイブキーが見つからなかった場合は復号化しない。）

WupiDecryptCode(1)の後に、CalcSimpleOperationファンクションを実行します。

実行後は、CalcSimpleOperationファンクションモジュールを暗号化しておくために、暗号化ファンクションWupiEncryptCode()を実行します。復号時と同様、ファンクションId=1をパラメータに入れ、

WupiEncyptCode(1)

を実行します。これで、CalcSimpleOperationファンクションモジュールはメモリー上で暗号化されます。以降、WupiDecryptCodeとWupiEncryptCodeを繰り返すことで「オンデマンド復号」を実現することが可能になります。

WUPIファンクションの詳しい使用方法は、サンプルプログラムが用意されていますので、そちらをご参照ください。¥Program Files¥WIBU-SYSTEMS¥AxProtector¥DevKit¥Samples¥IxProtectorの中にあります。(C/C++, Delphi)

3. 作成方法

C/C++における作成方法についてご説明します。

① ヘッダファイル<wibuixap.h>をプロジェクトにインクルードします。(#include "wibuixap.h")
<wibuixap.h>は、¥Program Files¥WIBU-SYSTEMS¥AxProtector¥DevKit¥includeの中にあります。

② ソースファイルをコンパイル＋リンクする。WupiEngine(32/64).libをリンクします。
WupiEngine(32/64).libは、¥Program Files¥WIBU-SYSTEMS¥AxProtector¥DevKit¥libの中にあります。

③ 上記で生成されたプログラムをWupiEngine(32/64).dllを使って動作検証します。

④ 動作検証後、問題なければプログラムをAxProtectorで暗号化します。
暗号化後は、WupiEngine(32/64).dllは不要です。(必要なモジュールが静的にリンクされます。)

⑤ これで完成です。

[NOTE]
WUPIファンクションの具体的な組み込み方法は、サンプルプログラムが用意されていますので、そちらをご参照ください。¥Program Files¥WIBU-SYSTEMS¥AxProtector¥DevKit¥Samples¥IxProtectorの中にあります。(C/C++, Delphi)

[NOTE]
IxProtectorの対象になるファイルは、以下の３つの条件を満たす必要があります。
- Windows EXE実行形式プログラムまたはDLLプログラム
- 自動暗号化ツールAxProtectorで暗号化し、問題なく動作すること。
  (IxProtectorオプションは指定しないで暗号化すること)
- 一般的なDLLファイルを呼び出せるプログラムであること。

11-5. WUPI ファンクション詳細

WupiAllocateLicense

[内容]
指定したライセンスリストのライセンスを割り当てます。

ライセンスの割り当てには、WupiCheckLicenseファンクションを使用する方が一般的です。
WibuCheckLicenseはライセンスを自動的に割り当てます。

[Syntax]
int WupiAllocateLicense (int iLicenseList);

[パラメータ]
iLicenseList
ライセンスリストインデックスを参照します。

[リターン値]
エラーの場合はFALSE(0)、それ以外はTRUE(1)を返します。

WupiFreeLicense

[内容]
割り当てたライセンスを解放します。

[Syntax]
int WupiFreeLicense (int iLicenseList);

[パラメータ]
iLicenseList
ライセンスリストインデックスを参照します。

[リターン値]
エラーの場合はFALSE(0)、それ以外はTRUE(1)を返します。

WupiGetHandle

[内容]
エントリのネイティブハンドルを返します。
CodeMeterの場合はHCMSysEntryに、WIBU-KEYの場合はHWKBENTRYに指定されます。

[Syntax]
WupiGetHandle (int iLicenseList);

[パラメータ]
iLicenseList
ライセンスリストインデックスを参照します。

[リターン値]
エラーの場合は0を返します。

WupiEncryptionCode

[内容]
WupiEncryptionCodeはファンクションを暗号化します。暗号化するファンクションはIxProtector設定で指定する必要があります。

[Syntax]
int WupiEncryptCode (int iFunction);

[パラメータ]
iFunction
ファンクションインデックスを参照します。

[リターン値]
エラーの場合はFALSE(0)、それ以外はTRUE(1)を返します。

WupiDecryptCode

[内容]
WupiDecryptCodeはファンクションを復号します。復号するファンクションはIxProtector設定で指定する必要があります。

[Syntax]
int WupiDecryptCode (int iFunction);

[パラメータ]
iFunction
ファンクションインデックスを参照します。

[リターン値]
エラーの場合はFALSE(0)、それ以外はTRUE(1)を返します。

WupiDecreaseUnitCounter

[内容]
CodeMeterの場合はUnit Counter（ユニットカウンター）、WIBU-KEYの場合はLimit Counter（リミットカウンター）を指定された数値分減らします。

[Syntax]
int WupiDecreaseUnitCounter (int iLicenseList, int cUnits);

[パラメータ]
iLicenseList
ライセンスリストインデックスを参照します。
cUnits
削減する数値。

[リターン値]
エラーの場合はFALSE(0)、それ以外はTRUE(1)を返します。

WupiQueryInfo

[内容]
使用中のライセンス情報を返します。

[Syntax]
int WupiQueryInfo (int iLicenseList, int iCmd);

[パラメータ]
iLicenseList
ライセンスリストインデックスを参照します。
iCmd（フラグ）

WupiQIUnitCounter
WIBU-KEYの場合はLimit Counter（リミットカウンター）の値、CodeMeterの場合はUnit Counter（ユニットカウンター）の値を返します。

WupiQIExpTime
Expiration Date（有効期限）を返します。
Expiration Dateは、2000年1月1日からの秒数になります。

WupiQIActTime
Activation Time（使用開始期日）を返します。（CodeMeterのみ）
Activation Timeは、2000年1月1日からの秒数になります。

WupiQIUsagePeriod
使用期間の最終日時からの有効な秒数を返します。使用期間が過ぎている場合は0を返します。（CodeMeterのみ）

WupiQIFirmCode
使用中のライセンスのファームコード(Firm Code）を返します。

WupiQIProductCode
使用中のライセンスのプロダクトコード(Product Code)を返します。WIBU-KEYの場合は、ユーザーコード（User Code)を返します。

WupiQIFeatureMap
使用中のライセンスのフィーチャーマップ（Feature Map)を返します。（CodeMeterのみ）

WupiQIBoxMask
使用中のライセンスを持つデバイスキーのボックスマスク(Box Mask)を返します。

WupiQIBoxSerial
使用中のライセンスを持つデバイスキーのシリアル番号(Serial Number)を返します。

[リターン値]
エラーの場合、または取得する情報が無い場合は、-1を返します。
ただし、WupiQIFeatureMapフラグを使って取得した値が-1の場合は、エラーではありません。
FeatureMap=0xFFFFFFFF(=-1)になります。WupiQIFeatureMapのエラーコードは0になります。
WupiGetLastErrorファンクションを使って正確なエラーコードを取得することができます。

WupiGetLicenseType

[内容]
ライセンスタイプを返します。

[Syntax]
int WupiGetLicenseType (int iLicenseList);

[パラメータ]
iLicenseList
ライセンスリストインデックスを参照します。

[リターン値]
UpiLicenseNone
ライセンスタイプは内部で設定されている。
UpiLicenseWibuKey
ライセンスタイプはWIBU-KEY。
UpiLicenseCodeMeter
ライセンスタイプはCodeMeter。
UpiLicenseCodeMeterAct
ライセンスタイプはCodeMeterAct。

WupiCheckDebugger

[内容]
プロテクトされたアプリケーションに対して、デバッグ処理が施されたかどうかをチェックします。

[Syntax]
int WupiCheckDebugger (int iLicenseList, int nLevel);

[パラメータ]
iLicenseList
ライセンスリストインデックスを参照します。
nLevel
アンチデバッグレベルを指定します。

Level 0: デバッガーチェックをしない。
Level 1: 簡易デバッガーチェック。
Level 2: カーネルデバッガーチェック。
Level 4: アドバンスデバッガーチェック。
Level 8: IDEデバッガーチェック。デバッガーが完全に使用できなくなる。
Level 16: デバッガーを検知するとCodeMeterまたはWIBU-KEYのエントリーを無効化（アクセス不可）する。
Level 64: 仮想マシン上での動作をチェックする。プロテクトされたアプリケーションが仮想マシン上で動作することを禁止する。

[リターン値]
デバッガーが検知されるとTRUE(1)を返し、それ以外はFALSE(0)を返します。Level16が含まれると、何も返さずにエントリーを無効化しアプリケーションを速やかに終了させます。デバッガーが検知された場合、WupiGetLastErrorファンクションは、error wibu::UpiErrorDebuggerDetected(-5)を返します。

WupiCheckLicense

[内容]
与えられたライセンスリストからライセンスをチェックします。使用されるセキュリティチェックは実行中に変化します。このファンクションは、ライセンスを自動的に割り当てます。

[Syntax]
int WupiCheckLicense (int iLicenseList);

[パラメータ]
iLicenseList
ライセンスリストインデックスを参照します。

[リターン値]
エラーの場合FALSE(0)を返し、それ以外はTRUE(1)を返します。

WupiGetLastError

[内容]
直前に実行したWUPIファンクションのエラーコードを返します。

[Syntax]
int WupiGetLastError();

[リターン値]
エラーコードを返します。

Error Codes

Errorcode: UpiErrorDebuggerDetected

デバッガーによる解析行為が検知されました。(-5)

Errorcode: UpiErrorFunctionNotFound

指定されたファンクションポイントが見つかりませんでした。エラー(-3)

Errorcode: UpiErrorInfoNotAvailable

WupiQueryInfoファンクションで取得した情報は無効です。(-7)

Errorcode: UpiErrorLicenseModuleNotLoaded

ライセンスモジュールがロードできません。(-6)
このエラーは、WIBU-KEYの場合WkWin32/64.dll、CodeMeter/CodeMeterActの場合WibuCm32/64.dllが見つからないか、またはインストールされていない場合に発生します。

Errorcode: UpiErrorLicenseNotFound

指定されたライセンスが見つかりませんでした。エラー(-2)

Errorcode: UpiErrorNoDefaultLicense

デフォルトライセンスが見つかりません。アプリケーションはAxProtectorでプロテクトされていません。エラー(-1)

Errorcode: UpiErrorNoError

エラーは発生しませんでした。(0)

Errorcode: UpiErrorRuntimeTooOld

インストールされているランタイムは古いバージョンです。エラー(-4)

Chapter 12

バッチ処理でコードを登録する

12-1. ワイブキー (ハードウェア） にバッチ処理でコードを登録する

ワイブキー（ハードウェア）WIBU-BOXに、コード設定ツール"Wklist32.exe"を使わずに、Windows上のコマンドライン環境でファイルバッチ処理により、コードなどを書き込むことができます。ファームコードやユーザーコードだけでなく、使用回数や使用期限などのオプションも登録可能です。

コマンドライン環境での書き変え方法を説明致します。
バッチ処理で使用するコード設定プログラムは “WKCRYPT.EXE” です。“WKCRYPT.EXE” は ¥Program Files¥Wibukey¥DevKit¥Bin フォルダにあります。"WKCRYPT.EXE"は、“WkAutoEnc.dll”、“wkfirm.dll”、“wkfirm.wbc” が必要です。デフォルトフォルダには、すでにファイルが存在していますが、"WKCRYPT" を別のフォルダにコピーして使用する場合は、“WkAutoEnc.dll”、“wkfirm.dll”、“wkfirm.wbc” も一緒にコピーする必要があります。

最初に、パラメータの説明は必要最小限にとどめて、実例をあげて基本動作について説明します。
パラメータの詳細については、その後で「WKCRYPTのパラメータ」として説明します。また、起動パラメータはスクリプトファイル（.wkcファイル）に記述して実行させることもできます（スクリプトファイルの作成方法は、基本動作の説明の最後で説明します）。

[NOTE]
Windows Vista/7にて、ユーザーアカウント制御(UAC)を有効にしているとワイブキーのバッチ処理が正常に動作しない場合があります。その場合は、ユーザーアカウント制御(UAC)を無効にしてください。

(1) コマンドプロンプトを起動します。

”WKCRYPT.EXE” があるフォルダをカレントにする、あるいは Pathを通す等で”WKCRYPT.EXE” を実行できる状態にしてください。

(2) ファームコード（FirmCode）=10 以外を操作する場合、該当するファームコードのFSBを接続してください。

以下の例では、ファームコード=10 で説明します。なお、プログラミングする（＝ファームコード、ユーザーコード等を設定する）WIBU-BOXは1個だけ接続するようにしてください。

以下は、ファームコード（FirmCode）=10、ユーザーコード（UserCode）=13 を設定する例です。

WKCRYPT /F10 /U13 /PAU /PN /L ↓　　（↓はリターンキーです）

/F ・・・　ファームコード（FirmCode）を指定します。
/U ・・・　ユーザーコード（UserCode）を指定します。
/PAU ・・　/PAでWIBU-BOXが接続されているデバイスポートを指定します。/PAU はUSBポートの指定です。プリンターポート、COMポートについては、後述の「WKCRYPTのパラメータ」の「デバイスポート指定のパラメータ」の項を参照してください。

/PN ・・・ 書き込みコマンドです。
/L ・・・ リスト表示コマンドです。

WKUSBを1個接続して、上記のコマンド"WKCRYPT /F10 /U13 /PAU /PN /L"を実行すると、
"FirmCode=10, UserCode=13"を空いている最初のエントリに書き込んで、結果をリスト表示します。
以上が書き込みの基本動作になります。

(3) 次に、削除コマンドを説明します。

【削除-1】
WIBU-BOXのすべての内容を初期化します。

WKCRYPT /PAU /PI /L ↓

/PI ・・・ WIBU-BOXのすべての内容を初期化します。異なるFirmCodeや、マスターエントリ、追加エントリ等が設定されている場合もすべて初期化します。ただし、10以外のFirmCodeを初期化するためには同じFirmCodeのFSBを装着する必要があります。もし必要なFSBが見つからなかった場合、初期化に失敗し、FirmCode=10を持つエントリも削除されません。(PI=Program Initialize)

【削除-2】
指定したFirmCodeに一致するすべてのエントリを削除します。

WKCRYPT /F10 /PAU /PC /L ↓

/PC ・・・ 特定のエントリを削除するコマンドです。(PC=Program Clear)
/FでFirmCodeを指定します。上記のコマンドを実行すると、FirmCode=10 を持つすべてのエントリが削除されます。/PCにユーザーコードを指定することも可能です。

WKCRYPT /F10 /PAU /PC13 /L ↓

上記のコマンドを実行すると、FirmCode=10, UserCode=13を持つすべてのエントリが削除されます。また、/PCMでMasterCode（マスターコード）を指定することも可能です。

WKCRYPT /F10 /PAU /PCM80 /L ↓

上記のコマンドを実行すると、FirmCode=10でMasterCode（マスターコード）=80を持つすべてのエントリが削除されます。

以上が削除の基本動作になります。

(4) 以下、操作のバリエーションです。
FirmCodeとUserCodeを指定のエントリに書き込みます。

WKCRYPT /F10 /U13 /PAU /PE3 /PN /L ↓

/PE で書き込みを行うエントリ（1～10）を個別に指定します。指定したエントリにすでに情報が設定されている場合はエラーになります。上記のコマンドを実行すると、FirmCode=10 : UserCode=13 をエントリ3 に書き込みます。

一度に複数のWKUSBを指定します。

WKCRYPT /F10 /U13 /PAU /PQ3 /PN /L ↓

/PQ で同時に操作を行うWKUSBの個数を指定します。
上記のコマンドを実行すると、PCに接続されている3個のWKUSBにFirmCode=10 : UserCode=13 を書き込みます。3個以上接続されている場合は、最初に見つけた3個に書き込まれます。

スクリプトファイルを使った場合

スクリプトファイルを使ってプログラミングすることも可能です。
スクリプトファイルは、"WKCRYPT.EXE" に与える一連のパラメータを記述したテキストファイルです。
通常のテキストエディタで作成し、拡張子を .wkc としてテキストファイルで保存します。起動パラメータとしてスクリプトファイルを指定する場合は、ファイル名の先頭にアットマーク (@) を付けます。

例. "WkEdit1.wkc" というスクリプトファイルを指定する場合・・・

WKCRYPT @WkEdit1.wkc ↓

以下に、スクリプトファイルの記述例をあげます。

【記述例-1】
コマンドラインで指定するパラメータを 1 ライン単位で記述します。

```
/F10 /U13 /PAU /PE1 /PN
/F10 /U14 /PAU /PE2 /PN /L
```

上記の例は、エントリ1にFirmCode10:UserCode13 、エントリ2にFirmCode=10,UserCode=14 を書き込んでから、リスト表示します。

【記述例-2】
コマンドラインで指定するパラメータを、パラメータ単位で記述します。

```
/F10
/U13
/PAU
/PE1
/PN
/F10
/U14
/PAU
/PE2
/PN
/L
```

上記の例は、エントリ1にFirmCode=10、UserCode=13 、エントリ2にFirmCode=10、UserCode=14 を書き込んでから、リスト表示します。

12-2. WKCRYPT のパラメータ

コマンド

/L　・・・リスト表示コマンド。
/PC　・・・削除コマンド。削除するFirmCodeは /Fパラメータで指定します。
例-1. FirmCode10のエントリを全て削除して結果をリスト表示します。
　　/F10 /PAU /PC /L
例-2.　FirmCode10のUserCode13のエントリを全て削除します。
　　/F10 /PAU /PC13 /L

同じユーザーコードが複数設定されていた場合、複数エントリについて削除を行います。例えばエントリ1とエントリ3 にFirmCode10:UserCode13 が登録されていた場合、エントリ1と 3 が削除されます。ただし、一度に削除できるエントリの個数は最大で10エントリまでです。複数のWIBU-BOXを接続して同時に全てのエントリを削除した場合、一度の操作で10エントリを超える削除はできません。この場合複数回に渡って操作を行ってください。

/PN　・・・作成コマンド
/Fパラメーターと/Uパラメータ で指定された FirmCodeとUserCode を書き込みます。
例-1.　FirmCode10:UserCode13を最初に見つけた空きエントリに書き 込みます。
　　/F10 /U13 /PAU /PN　/L
例-2.　FirmCode10:UserCode13を5番目のエントリに書き込みます。
　　/F10 /U13 /PAU /PE5 /PN　/L

/PEで指定するエントリがすでに使用されている場合はエラーになります。

/PR ・・・変更コマンド。すでに登録されているFirmCode:UserCodeを変更します・
/FパラメータでFirmCodeを指定します。
/Uパラメータで変更後のUserCodeを指定します。
/PRの直後に変更したいUserCodeを指定します。

例-1. 登録済みのFirmCode=13,UserCode=10を FirmCode=10,UserCode=77に書き変えてリスト表示します。
　　/F10 /U77 /PAU /PR13 /L

対象となるエントリが複数存在する場合、それらすべてのエントリが書き換わります。

デバイスポート指定のパラメータ /PA

/PA (PA=Port Addressing)

/PAU ・・・USBポート指定。
/PAL ・・・LPTポート（プリンターポート）指定。直後にLPT番号を指定します。
（例） LPT1は"/PAL1"、LPT2は"/PAL2"
/PAC ・・・COMポート指定。直後にCOMポート番号を指定。
（例）COM1は"/PAC1"、COM2は"/PAC2"

個数の指定のパラメータ /PQ

/PQ ・・・操作対象となる個数を指定。
例. /PQ1 、/PQ2 、/PQ3 ・・・

ただし、見つかったWIBU-BOXの最後からの個数になります。例えば、1,2,3,4,5とWKUSBが接続されていて、/PQ2を指定した場合、4と5のWKUSBが操作対象となります。

ファームコードの指定 /F

/F ・・・ファームコードを指定します。
例. /F10 、/F1234

10番以外のファームコードを操作する場合、FSBと “wkfirm.wbc” がセットされている必要があります。

ユーザーコードの指定 /U

/U ・・・ ユーザーコードを指定します。
例. /U1 、/U2 、/U3 ・・・

エントリ番号の指定 /PE

/PE ・・・ 1から10までのエントリ番号を指定します。

※指定したFirmCode:UserCodeが登録されていない場合は何もしません。

追加エントリの設定

追加エントリを設定する場合は、あらかじめ基本エントリが作成されていなければなりません。追加エントリの設定が可能な基本エントリは Entry1～Entry5 です。

Entry6～Entry10は、追加エントリの記録領域として使用されます。従って、Entry6～Entry10が既に使用済みの場合は追加エントリを設定できないことがあります。

※基本エントリの特定には、FirmCodeとUserCodeを使用することもできますが、エントリが複数存在する場合、期待する動作が得られない場合があります。確実に作業を行う為にはエントリ番号を指定（例. /PE1, /PE3 etc）することをお勧めします。

以下の説明でもエントリ番号によって特定しています。

【 HighData と LowData の設定 と変更 】

/PXD ・・・追加エントリにHighDataとLowDataを書込みます。
指定方法は /PXD10-20 のようにHighDataとLowDataをハイフンで区切って指定します。
/PXD10 のように一方だけを記述した場合はHighDataがゼロとして扱われ ます。

例-1. WKUSBのエントリ 1 の追加エントリにHighData=10、LowData=123 を書込みます。

WKCRYPT /PAU /PE1 /PXD10-123 /L ↓

/PMD　・・・すでに設定されているHighDataとLowDataを修正します。

【有効期限の設定と変更】

/PXT　・・・有効期限を指定します。

例-1.　WKUSBのFirmCode10:UserCode13 に現在の日付から10日間の有効期限を設定します。

WKCRYPT /PAU /F10 /U13 /PXT10 /L ↓

例-2.　WKUSBのFirmCode10:UserCode13 に2013年3月31日までの有効期限を設定します。

WKCRYPT /PAU /F10 /U13 /PXT13Mar31 /L ↓

年の指定は70～99 が1970 ～1999 に対応します。00～69 が 2000 ～ 2069 に対応します。
月の指定は Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Decになります。

/PMT　・・・すでに設定されている制限期日を変更する点を除いて /PXT と同じです。

Chapter 13

その他

13-1. ワイブキーランタイムキットのインストール方法について

ワイブキーランタイムキットのインストール方法には２通りあります。

1. ワイブキーランタイムキットWkRuntime(32/64).exeを使用する方法（推奨）
2. Setup32(64).exeを使用する方法（サイレント・インストール）

1. ワイブキーランタイムキット WkRuntime(32/64).exe からインストールする

ワイブキーランタイムキットWkRuntime(32/64).exeには、ワイブキードライバーをはじめ、ワイブキーセキュリティに必要なファイルが含まれています。エクスプローラ上からWkRuntime(32/64).exeを起動すると、メッセージ画面が表示されますので、メッセージに従ってインストール作業を行います。

ランタイムキットには3種類用意されており、OSにより使い分けます。

WkRuntime.exe (Windows OS 32bit/64bit兼用)
WkRuntime32.exe (Windows OS 32bit専用)
WkRuntime64.exe (Windows OS 64bit専用)

上記ファイルは、ワイブキーCDの¥RunTimeフォルダにあります。また、最新のランタイムキットは弊社サイトからダウンロードできます。

http://www.suncarla.co.jp/download/

2. Setup(32/64).exe からインストールする（サイレント・インストール）

上述のWkRuntime(32/64).exeの場合、インストール時にワイブキーのメッセージ画面が表示されますが、Setup(32/64).exeを使うと、画面上には何も表示されずにワイブキーランタイムキットをインストールすることができます。（サイレントインストール）

Setup32.exeは、Windows 32bit用インストールプログラムで、Windows95/98/Me/NT/2000/2003/2008/XP/Vista/7(32bit)をサポートします。Setup32.exeは、ワイブキー開発キット(SDK)をインストールしたPC(32bit OS)の¥Program Files¥WibuKey¥DevKit¥RuntimeKitフォルダの中にあります。

Windows 64bit OSへのインストールはSetup64.exeを使用します。使用方法はSetup32.exeと同じです。Setup64.exeは、ワイブキー開発キット(SDK)をインストールしたPC(64bit OS)の¥Program Files¥WibuKey¥DevKit¥RuntimeKit64フォルダの中にあります。

[NOTE]
RuntimeKit64フォルダは32bitOSのPC上では作成されません。

また、Setup.exeは、Windows 32bit/64bit共通のインストーラです。OSが32bitか64bitかを自動的にチェックし、そのOS環境に合ったランタイムキットをインストールします。通常は、このSetup.exeを使用することを推奨致します。

[NOTE]
Setup.exeは、32bit OS上の¥Program Files¥WibuKey¥DevKit¥RuntimeKitフォルダに作成されます。

【Setup(32/64).EXEの使い方】
RuntimeKitフォルダまたは RuntimeKit64フォルダには、Setup(32/64).exe以外にワイブキー・ドライバー等が圧縮されて格納されています。フォルダ内（RuntimeKitフォルダおよびRuntimeKit64フォルダ）にあるSetup.iniファイルをエディタ等で開き、"Gui=1"を"Gui=0"に書き直し、このフォルダ全体（RuntimeKitフォルダおよびRuntimeKit64フォルダ）を、貴社のアプリケーションCDにコピーして使用します。Gui=0の状態でSetup(32/64).exeを起動すると、サイレント・インストールが開始されます。Gui=1の状態でSetup(32/64).exeを起動すると、メッセージ表示によるインストールが開始されます。（RuntimeKit(32/64).exeと同じインストール方法）

Windows Vista/7でサイレントインストールを行う場合、ユーザーアカウント制御機能の為、Setup(32/64).exeの実行を確認する画面が表示されます。あらかじめご了承ください。

[Setup.ini]

```
; ------------------------------------
; SETUP.INI for WIBU-KEY Enduser Setup
; ------------------------------------
; Driver & Tools (US,BR,CN,DE,ES,FR,HU,IT,JP)
; Date: 2005Feb24
; ------------------------------------

[General]
; Language for the Setup
SetupLang=
; Languages for WIBU-KEY
Languages=us,br,cn,de,es,fr,hu,it,jp
; path to the source files
SourcePath=
; Display WkUseXX.chm
; 1=yes (default), 0=button not checked
Readme=1
; Overwrite existing WIBUKEY.INI entries
; 0=no (default), 1=yes
ForceOverwrite=0
; Display user interface
; 0=no GUI, 1=GUI (default)
Gui=0
; Display error messages
; 0=no error messages, 1=display message box (default)
ErrorMessages=1

[Tools]
; Install WkSvW32 as Service
; 0=no (default), 1=yes
WkSvW32Service=0
; Display tool selection dialog
; 1 = do not display, 0 = display (default)
NoToolSelect=0
; 0=no, 1=yes (default)
Shortcuts=1

   .
   .
   .
以下、略
```

[NOTE]
貴社のアプリケーションCDにワイブキーのランタイムキット用ファイルをコピー配布しても、著作権上、問題ありません。

［注意］
ワイブキー・ドライバーを、Windowsのシステムフォルダにコピーしただけでは動作しませんので、必ず、ランタイムキットインストーラWkRuntime(32/64).exeかSetup(32/64).exeを使ってインストールをしてください。

13-2. Mac 上でワイブキーを使用する場合

Mac上でワイブキーを認識させ、キーに設定を書き込むためには、専用のUSBドライバーと開発ツールをインストールする必要があります。

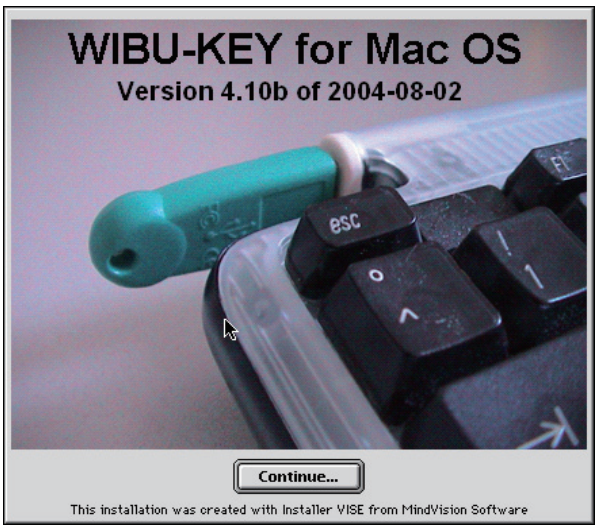
1. 開発ツールとドライバーをインストールする

Mac OS 9 の場合

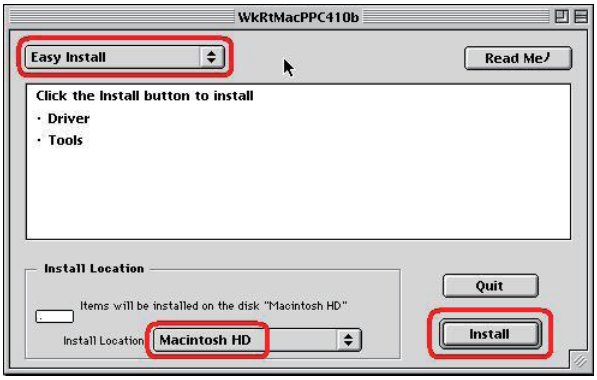
ワイブキーCDのMacOSフォルダ内RunTimeフォルダを開き、WkRtMacPPC410b.hqxファイルをマウントします。

WkRtMacPPC410b.hqxファイルをダブルクリックし解凍します。“WkRtMacPPC410b” フォルダが作成されます。

“WkRtMacPPC410b” フォルダをダブルクリックするとインストーラーが起動し、「WIBU-KEY for Mac OS」画面が表示されます。［Continue］をクリックします。



次画面で「Easy Install」が選択されていることを確認して下さい。
「Install Location」で起動ディスク（Macintosh HD）が選択されていることを確認して下さい。
［Install］ボタンをクリックすると、インストールを開始します。



「Installation was successful. 以下省略」のメッセージが表示されたら、ドライバー、開発ツールのインストールは完了です。［Quit］ボタンをクリックしインストーラーを終了して下さい。インストール先フォルダに［WIBUKEY］フォルダが作成されます。

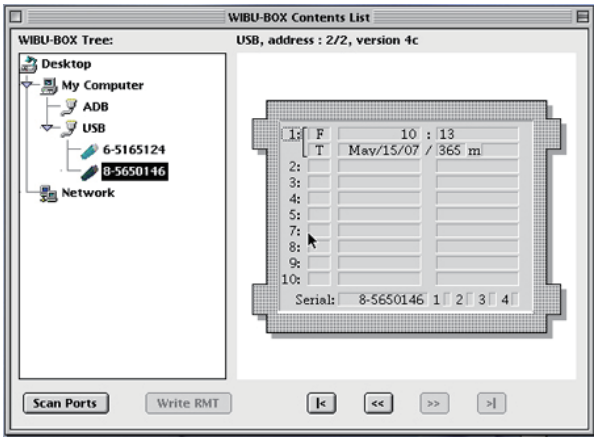
評価版の場合は、ワイブキーCDを開きDEVKITフォルダ – BINフォルダのWKFIRM.WBCファイルを、製品版の場合は、ワイブキーFSB（開発キット）に付属のライセンスCDに含まれる、貴社専用のWKFIRM.WBCファイルを、先ほどインストールした［WIBUKEY］フォルダにコピーします。
ワイブキーを、USBポートに装着し正常に認識されることを確認して下さい。

Mac OS X 10.4, 10.5, 10.6 の場合
ワイブキーCDのMacOSフォルダ内DevKitフォルダの中のWkDevKit_6.0.501.dmgをマウントします。

Mac OS X 10.3 の場合
ワイブキーCDのMacOSフォルダ内DevKitフォルダの中のWkDevKit_5.20.503.dmgをマウントします。

2. ワイブキーへのコード登録（コードの書き込み）

ワイブキーのコード設定（コードの書き込み）は、Windowsの場合と同じです。使用するプログラムは、WkMaclist になります。開発キットに付属のライセンスCDから、貴社のライセンスデータWkFirm.wbcファイルをインストール先フォルダにコピーし、FSBを装着した上で、ワイブキーにコード設定作業を行います。インストール先フォルダの“WkMacList” を起動し、ワイブキーに各種設定を書き込みます。メニュー表示が英語になるなど、多少、Windowsの画面と異なる部分がありますが、基本的な使い方は同じです。



3. ファイルのプロテクト方法

プログラムファイルにプロテクトを施す場合、OSのバージョンにより方法が異なります。

Mac OS X の場合
1. ワイブキーのAPIファンクションをソースプログラムに組み込む方法
2. 「AxProtector」を使って自動暗号化する方法

の２通りの方法があります。

Mac OS 9 の場合
1. ワイブキーのAPIファンクションをソースプログラムに組み込む方法

だけとなり、自動暗号化ツール「AxProtector」は使用できません。

4. その他

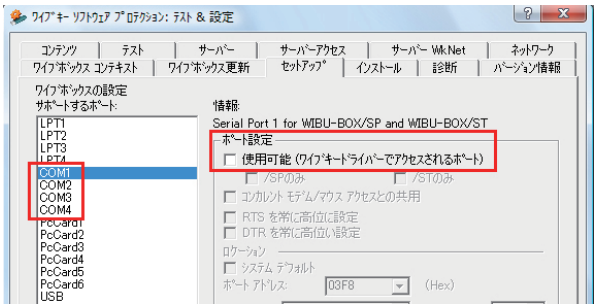
上記以外は、基本的にWindows上での使い方と同じです。

13-3. COM ポート版 (WK25ST) を使用する場合

※ COM ポート版（RS232C シリアルポート版）の WK25ST を使用する場合は、次の点にご注意下さい。

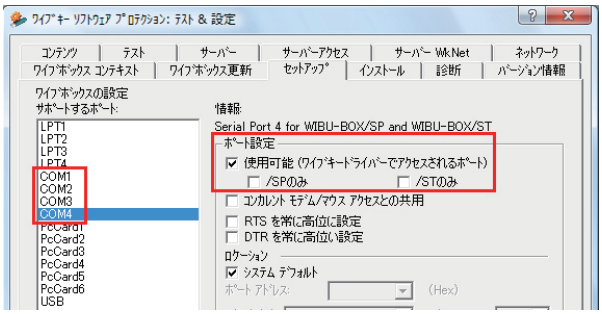
ワイブキー・ドライバーVer3.00以降、デフォルトではCOMポートが使用不可の状態になっています。
従い、COMポート版(WK25ST)を使用する場合は、初期設定として、ワイブキー・ドライバーをインストールした後に、コントロールパネル上の「ワイブキー」(WIBUKE32.CPL)をアドバンストモードで立ち上げ、「セットアップ」メニューから、使用するCOM1からCOM4までを「使用可能」に設定する必要があります。

ポートを選択し、「ポート設定」の「使用可能」にチェックします。



COM1からCOM4まで設定してください。

この作業はインストール直後に一度行なえば、次回以降は行なう必要はありません。
（但し、ワイブキー・ドライバーを更新、再インストールした場合は、この初期設定を行なう必要があります。



[NOTE]
WKUSB, WK25Pの場合は、初期設定が「使用可能」の状態になっているため、この作業は不要です。

[NOTE]
COMポート版(WK25ST)を使用する場合は、ワイブキー・ドライバーVer3.00 c 以降を使用してください。

13-4. 旧ライセンスファイル "WkFirm.wbc" をお使いの場合

ライセンスファイル”WkFirm.wbc”（または“Wkfirm.dat” ）がVersion3.00 以前のものをお使いの場合、最新のワイブキー（ハードウェアバージョン4c、5、6、7）へのFirmCodeの書き込みができませんのでご注意ください。

“WkFirm.wbc”のバージョンは以下の方法で確認することができます。

貴社のライセンスファイル“WkFirm.wbc”または"Wkfirm.dat" をメモ帳などのテキストエディタで開きファイルの先頭から4行目の "Version= "を確認します。

Version=4.00の場合
全てのワイブキーハードウェアバージョンに対応しています。アップデートの必要がありません。

Version=3.00の場合
ワイブキーハードウェアバージョン4c以降には対応しません。ワイブキーハードウェアバージョン4c以降のワイブキーを編集する場合は、ライセンスファイル"WkFirm.wbc"または"Wkfirm.dat"をアップデートする必要があります。弊社（サンカーラ）までご連絡ください。無料にてアップデート致します。

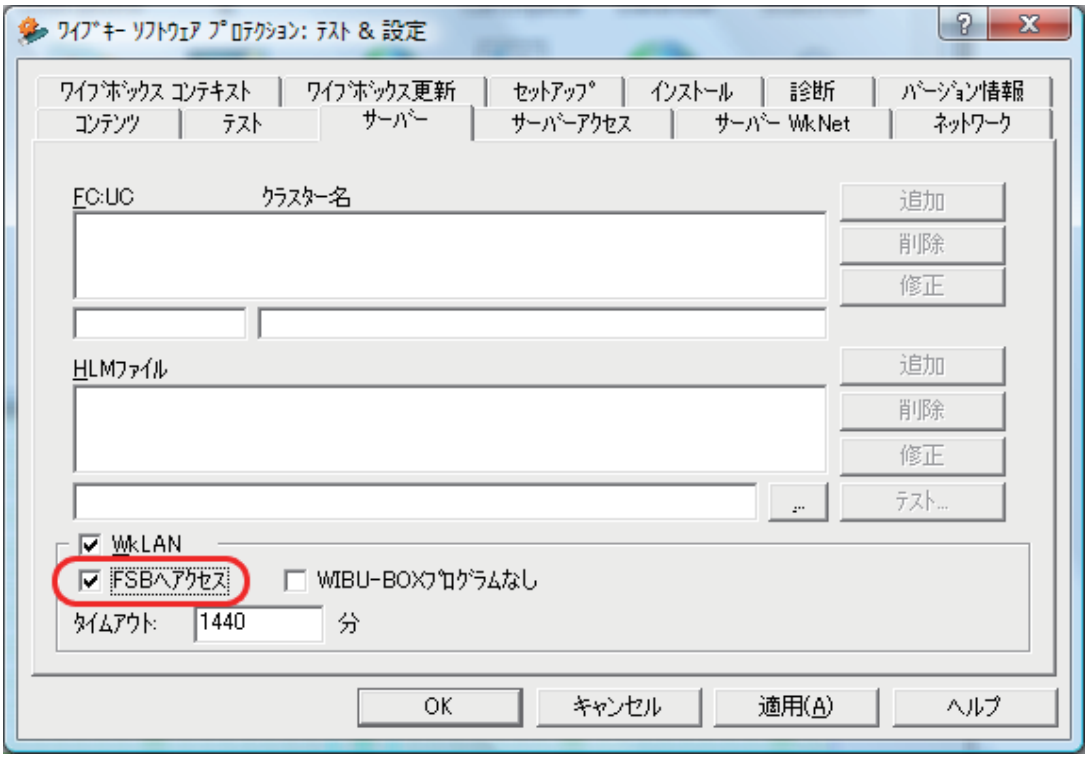
13-5. ネットワーク上で FSB(Firm Security Box) を使用する

FSBとワイブキーを同一のPCに装着することができない場合、ネットワーク上のFSBを参照してプログラミングすることができます。以下はネットワークでの使用手順です。

1. FSBを装着するネットワーク上のPCにワイブキーランタイムキットをインストールします。
2. 【コントロールパネル】にある「WIBU-KEY」ツールをアドバンストモードで起動し、[サーバー]タブの「□FSBへアクセス」にチェックを入れ、OKまたは適用ボタンをクリックします。
3. 【スタート】→【プログラム】→【WIBU-KEY】にある「ネットワークサーバー」を起動します。
   ネットワークサーバーが起動するとタスクトレイにアイコンが表示されます。

以上で、同一ネットワーク上のPCから、ネットワーク経由でFSBを参照することができるようになります。

※ライセンスファイル("WkFirm.wbc")は、プログラミングするワイブキーを装着するPC側にコピーします。



［注意］
ネットワーク上のFSBへのアクセスを許可している場合、開発キットとライセンスファイル"WkFirm.wbc"がインストールされているPCからワイブキーのコード設定・編集が可能になります。FSBの管理には十分ご注意ください。

FSBをネットワーク経由で利用する場合は、充分なセキュリティ管理のもとで実施されるようお願いいたします。

13-6. Access ファイルにプロテクトをかける場合

Accessのデータベースプログラム（アプリケーションプログラム）にプロテクトをかける場合は、自動暗号化ツールAxProtectorが使用できませんので、AccessのVBAからワイブキーのクラシックAPIまたはCOM APIを呼び出してプロテクト処理を行います。

ワイブキーのクラシックAPIおよびCOM APIを使用するために特別なツールをインストールする必要はありません。標準的なワイブキードライバーをインストールするだけでこれらのAPIを利用することができます。

ここでは、少ないコーディングで簡単に実装できるCOM APIを使ってワイブキーをチェックする方法を説明します。

Accessの Visual Basic Editor を起動してください。
まず最初に、【ツール(T)】→【参照設定(R)】で参照可能なライブラリファイルを表示し、［WIBU-KEY COM API］にチェックを入れて、[OK]ボタンを押してください。
これで、VBAからワイブキーCOM APIを使用することができます。

以下は、ローカルのPCに接続された、FirmCode=10、UserCode=13 のワイブキーをチェックする最も簡単な例です。ボタンのクリックイベントなどに記述してお試しください。

```
01  Dim wkb As Wibukey
02  Set wkb = New Wibukey
03  Static Cdata As String * 8
04  '
05  Cdata = "12345678"
06  wkb.FirmCode = 10
07  wkb.UserCode = 13
08  wkb.TextData = Cdata
09  wkb.UsedSubsystems = 1
10  wkb.Encrypt
11  If wkb.LastErrorCode = 0 Then wkb.Encrypt
12  '
13  If (wkb.LastErrorCode = 0) and (wkb.TextData = Cdata) Then
14      MsgBox ("チェック成功")
15  Else
16      MsgBox ("ErrorCode : " & wkb.LastErrorCode)
17  End If
```

各行について簡単に説明します。

01行目で Wibukey型のwkbを定義して、02行目でwkbオブジェクトを生成しています。

サンプルではワイブキーのEncryptメソッドによってチェックを行っています。
03行目でテスト用の暗号化で使用する文字列を定義して、05行目でテスト用の文字列"12345678"を設定しています。

06行目～07行目でチェックするFirmCodeとUserCodeを設定しています。
08行目で暗号化データをTextDataプロパティにセットしています。

09行目の UsedSubSystems はチェックの対象となるサブシステムの設定です。
1 を指定するとローカルPCに接続されているワイブキーだけがチェックの対象になります。
2 を指定するとLAN上のワイブキーサーバーに接続されているワイブキーだけがチェックの対象になります。
3 を指定するとローカルとLANの双方がチェックの対象になります。

※但し、ワイブキーのAPIファンクションでLAN上のワイブキーをチェックする場合は、接続可能なクライアント数について若干の注意が必要となります。これについては、次のLAN上のワイブキーをチェックするサンプルを参照してください。

10行目でEncryptメソッドによってワイブキーのチェックを行っています。 06行～07行で指定したFirmCode/UserCodeが設定されたワイブキーがローカルPCで見つかると、Encryptメソッドが実行されて TextDataプロパティの文字列データが暗号化されます。また、LastErrorCodeにはチェックが成功したことを表すゼロが返されます。LastErrorCodeにゼロ以外が返された場合はチェックが失敗したことを表します。値はエラーコードを示します。

11行目で、1回目のチェックが成功した場合に再度Encryptメソッドを呼び出しています。
これによってTextDataプロパティの文字列データは復号化されます。

13行目で LastErrorCodeの値がゼロであることと、TextDataプロパティの文字列データが正しく復号化されたことを確認して最終的な判定を行っています。

サンプルではチェックの結果をメッセージボックスで表示していますが、実際にはエラーの有無によって貴社アプリケーションに適合した分岐処理を行ってください。

以下は、LAN上のワイブキーサーバーに接続されたFirmCode=10、UserCode=13 のワイブキーをチェックする最も簡単な例です。

```
01  Dim wkb As Wibukey
02  Set wkb = New Wibukey
03  Static Cdata As String * 8
04  '
05  Cdata = "12345678"
06  wkb.FirmCode = 10
07  wkb.UserCode = 13
08  wkb.TextData = Cdata
09  wkb.UsedSubsystems = 2
10  wkb.RestrictedUserQuantityMode = true
11  wkb.Encrypt
12  If wkb.LastErrorCode = 0 Then wkb.Encrypt
13  '
14  If (wkb.LastErrorCode = 0) and (wkb.TextData = Cdata) Then
15     MsgBox ("チェック成功")
16  Else
17     MsgBox ("ErrorCode : " & wkb.LastErrorCode)
18  End If
```

ローカルPCのチェックと異なる点についてのみ説明します。

09行目のUsedSubSystemsで 2 を指定してLAN上のワイブキーだけを対象にしています。

10行目の RestrictedUserQuantityMode プロパティは同時接続クライアント数の制限をするかどうかを決定するためのものです。 trueを設定すると同時に接続できるクライアント数の制限が有効になります。falseを設定すると、クライアント数の制限に関係なく無制限にLAN上のワイブキーチェックだけが行われます。デフォルトはfalseです。

尚、クライアントからの接続は、Wibukeyオブジェクトが破棄された時点で自動的に開放されますのでご注意ください。例えば、ボタンのクリックイベントで上記のサンプルコードを使用した場合、Wibukey型の定義もクリックイベント内に記述すると、クリックイベントから戻った時点でWibukeyオブジェクトが破棄されて接続クライアント数が解放されます。

13-7. Excel ファイルにプロテクトをかける場合

Excelのアプリケーションプログラムにプロテクトをかける場合は、自動暗号化ツール「AxProtector.EXE」が使用できませんので、ExcelのVBAからワイブキーのクラシックAPIまたはCOM APIを呼び出してプロテクト処理を行います。

ワイブキーのクラシックAPIおよびCOM APIを使用するために特別なツールをインストールする必要はありません。標準的なワイブキードライバーをインストールするだけでこれらのAPIを利用することができます。

ここでは、数行のコーディングで簡単に実装できるCOM APIを使ってワイブキーをチェックする方法を説明します。

Excelの Visual Basic Editor を起動してください。
まず最初に、【ツール(T)】→ 【参照設定(R)】 で参照可能なライブラリファイルを表示し、[WIBU-KEY COM API] にチェックを入れて、[OK]ボタンを押してください。
これで、VBAからワイブキーCOM APIを使用することができます。

以下は、ローカルのPCに接続された、FirmCode=10、UserCode=13 のワイブキーをチェックする最も簡単な例です。
ボタンのクリックイベントなどに記述してお試しください。

```
01  Dim wkb As Wibukey
02  Set wkb = New Wibukey
03  Static Cdata As String * 8
04  '
05  Cdata = "12345678"
06  wkb.FirmCode = 10
07  wkb.UserCode = 13
08  wkb.TextData = Cdata
09  wkb.UsedSubsystems = 1
10  wkb.Encrypt
11  If wkb.LastErrorCode = 0 Then wkb.Encrypt
12  '
13  If (wkb.LastErrorCode = 0) and (wkb.TextData = Cdata) Then
14     MsgBox ("チェック成功")
15  Else
16     MsgBox ("ErrorCode : " & wkb.LastErrorCode)
17  End If
```

各行について簡単に説明します。

01行目で Wibukey型のwkbを定義して、02行目でwkbオブジェクトを生成しています。

サンプルではワイブキーのEncryptメソッドによってチェックを行っています。
03行目でテスト用の暗号化で使用する文字列を定義して、05行目でテスト用の文字列"12345678"を設定しています。

06行目～07行目でチェックするFirmCodeとUserCodeを設定しています。

08行目で暗号化データをTextDataプロパティにセットしています。

09行目の UsedSubSystems はチェックの対象となるサブシステムの設定です。
1 を指定するとローカルPCに接続されているワイブキーだけがチェックの対象になります。
2 を指定するとLAN上のワイブキーサーバーに接続されているワイブキーだけがチェックの対象になります。
3 を指定するとローカルとLANの双方がチェックの対象になります。
※但し、ワイブキーのAPIファンクションでLAN上のワイブキーをチェックする場合は、接続可能なクライアント数について若干の注意が必要となります。これについては、次のLAN上のワイブキーをチェックするサンプルを参照してください。

10行目でEncryptメソッドによってワイブキーのチェックを行っています。06行～07行で指定したFirmCode/UserCodeが設定されたワイブキーがローカルPCで見つかると、Encryptメソッドが実行されて TextDataプロパティの文字列データが暗号化されます。また、LastErrorCodeにはチェックが成功したことを表すゼロが返されます。LastErrorCodeにゼロ以外が返された場合はチェックが失敗したことを表します。値はエラーコードを示します。

11行目で、1回目のチェックが成功した場合に再度Encryptメソッドを呼び出しています。
これによってTextDataプロパティの文字列データは復号化されます。

13行目でLastErrorCodeの値がゼロであることと、TextDataプロパティの文字列データが正しく復号化されたことを確認して最終的な判定を行っています。

サンプルではチェックの結果をメッセージボックスで表示していますが、実際にはエラーの有無によって貴社アプリケーションに適合した分岐処理を行ってください。

以下は、LAN上のワイブキーサーバーに接続されたFirmCode=10、UserCode=13 のワイブキーをチェックする最も簡単な例です。

```
01  Dim wkb As Wibukey
02  Set wkb = New Wibukey
03  Static Cdata As String * 8
04  '
05  Cdata = "12345678"
06  wkb.FirmCode = 10
07  wkb.UserCode = 13
08  wkb.TextData = Cdata
09  wkb.UsedSubsystems = 2
10  wkb.RestrictedUserQuantityMode = true
11  wkb.Encrypt
12  If wkb.LastErrorCode = 0 Then wkb.Encrypt
13  '
14  If (wkb.LastErrorCode = 0) and (wkb.TextData = Cdata) Then
15      MsgBox ("チェック成功")
16  Else
17      MsgBox ("ErrorCode : " & wkb.LastErrorCode)
18  End If
```

ローカルPCのチェックと異なる点についてのみ説明します。

09行目のUsedSubSystemsで 2 を指定してLAN上のワイブキーだけを対象にしています。

10行目の RestrictedUserQuantityMode プロパティは同時接続クライアント数の制限をするかどうかを決定するためのものです。trueを設定すると同時に接続できるクライアント数の制限が有効になります。falseを設定すると、クライアント数の制限に関係なく無制限にLAN上のワイブキーチェックだけが行われます。デフォルトはfalseです。

尚、クライアントからの接続は、Wibukeyオブジェクトが破棄された時点で自動的に開放されますのでご注意ください。例えば、ボタンのクリックイベントで上記のサンプルコードを使用した場合、Wibukey型の定義もクリックイベント内に記述すると、クリックイベントから戻った時点でWibukeyオブジェクトが破棄されて接続クライアント数が解放されます。

Chapter 14

ユーザーに配布する場合

14-1. ユーザーに配布する場合

ワイブキーで暗号化したプログラムを起動するには、あらかじめワイブキーランタイムキットをPCにインストールする必要があります。インストールするワイブキーランタイムキットは、使用するOS環境により異なります。また、暗号化方法により、ワイブキーランタイムキット以外のファイルも一緒にインストールする必要があります。それぞれの使用環境によって、ユーザーに配布するファイルを確認してください。なお、ワイブキーに関連するファイルは、貴社のアプリケーションと一緒に配布することは著作権上問題ありません。

14-2. Windows アプリケーション (32bit/64bit 版) を配布する

Windowsアプリケーション(32bit/64bit)用に 3 種類のランタイムキットが用意されています。

WkRuntime.exe (Windows OS 32bit/64bit兼用: 18.6MB)
WkRuntime32.exe (Windows OS 32bit専用: 12.9MB)
WkRuntime64.exe (Windows OS 64bit専用: 13.9MB)

WkRuntime.exeは、Windows OS 32bit/64bit兼用のランタイムキットです。インストールするOSが32bitか64bitかを自動的に判断し、そのOSに必要な内容をインストールします。32bit用、64bit用のランタイムキットを使い分ける必要がないため便利です。

WkRuntime32.exeは、32bitOS専用のランタイムキットです。
WkRuntime64.exeは、64bitOS(EM64T)専用のランタイムキットです。アプリケーションが32bitOS対応の場合でも、Windows 64bit OS(EM64T)環境にインストールする場合は、WkRuntime64.exeを使用します。

[NOTE]
Windows 32bit OSにWkRuntime64.exeをインストールしようとするとエラーになりインストールに失敗します。また、Windows 64bit OSにWkRuntime32.exeをインストールする場合も同じようにエラーになりインストールすることができません。

ワイブキーランタイムキットは、ワイブキーCDの¥RunTimeフォルダにあります。また、最新のランタイムキットは弊社サイトからダウンロードできます。

http://www.suncarla.co.jp/download/

14-3. .NET アプリケーション (32bit/64bit 版) を配布する

.NETアプリケーション(32bit/64bit版)を配布する場合は、Windowsアプリケーション(32bit/64bit版)を配布する場合と同じです。

WkRuntime.exe (Windows OS 32bit/64bit兼用: 18.6MB)
WkRuntime32.exe (Windows OS 32bit専用: 12.9MB)
WkRuntime64.exe (Windows OS 64bit専用: 13.9MB)

14-4. Mac OS アプリケーションを配布する

Mac OSのアプリケーションを配布する場合は、Mac用コードメータランタイムキットをインストールします。

Mac OS X 10.4, 10.5, 10.6
① Mac用コードメータランタイムキット"WkRuntimeUser_6.0.501.dmg"

Mac OS X 10.3
① Mac用コードメータランタイムキット"WkRuntimeUser_5.20.503.dmg"

Mac OS X 10.2まで
① Mac用コードメータランタイムキット"WkRtMacX.tgz"

Mac OS 9
① Mac用コードメータランタイムキット"WkRtMacPPC410b.hqx"

14-5. Linux アプリケーションを配布する

RPM パッケージ (for SuSe, Red Hat)
32bit
① 32bit WIBU-KEY Runtime Kit "WkRt-Lin-6.0.501-1.i386.rpm"
AxProtectorで暗号化した32bit Javaプログラムの場合、さらに
②32bit Java Runtime Kit "AxProtector-6.51.102-1.i386.rpm"が必要
64bit
① 64bit WIBU-KEY Runtime Kit "WkRt-Lin64-6.0.501-1.x86_64.rpm"
AxProtectorで暗号化した64bit Javaプログラムの場合、さらに
②64bit Java Runtime Kit "AxProtector64-6.51.102-1.x86_64.rpm"が必要

DEBパッケージ(for Ubuntu, Debian)
32bit
① 32bit WIBU-KEY Runtime Kit "wkrt-lin_6.0.501_i386.deb"
AxProtectorで暗号化した32bit Javaプログラムの場合、さらに
②32bit Java Runtime Kit "axprotector_6.51.103_i386.deb"が必要
64bit
① 64bit WIBU-KEY Runtime Kit "wkrt-lin64_6.0.501_amd64.deb"
AxProtectorで暗号化した64bit Javaプログラムの場合、さらに
②64bit Java Runtime Kit "axprotector64_6.51.103_amd64.deb"が必要

Chapter 15

WIBU-KEY API ファンクション一覧

WIBU-KEY API ファンクション一覧

最も重要なファンクションは、WkbOpen2ファンクションです。このファンクションは、実行時のパラメータにFirmCodeとUserCode等を指定して実行し、指定したFirm CodeとUser Codeを持つワイブキーエントリを検索します。もし、ワイブキーエントリが見つかった場合、そのハンドルを返し、見つからなかった場合はNULLを返します。従い、リターン値がNULLかどうかで、指定したワイブキーエントリの存在を確認することができます。このWkbOpen2ファンクションが実行するごとに、ワイブキーを物理的にチェックしますので、貴社のソースプログラムの数箇所に組み込んでおくことをお勧めいたします。
なお、WkbOpen2でワイブキーエントリをチェックした後は、その都度、WkbClose2ファンクションでワイブキーエントリをクローズさせて下さい。各言語に対応したサンプルプログラムが用意されています。
（「Chapter 9 WIBU-KEY APIファンクションについて / 9-2. サンプルプログラムのインストール方法について」参照）

WkbAccess2
ローカルサブシステムまたはネットワークサブシステムのWIBU-KEYにアクセスする。

WkbAllocMem
WIBU-KEYドライバーの中にメモリーブロックを割り当てます。

WkbBitResetSet
論理的作業で32ビット整数値のビットを変化させる。

WkbClose2
オープンされたWIBU-BOXエントリをクローズする。

WkbControlSystem2
指定したWIBU-KEYサブシステムをコントロールする。

WkbCrypt2
メモリー上でデータを暗号化・復号化する。

WkbEnumPort2
利用可能なすべてのWIBU-KEYポートのリストを返す。

WkbFreeMem
WIBU-KEYドライバー内のメモリーブロックを解放する。

WkbGetFC
現在設定されているファームコード(Firm Code）を返す。

WkbGetLastError
直前に発生したファンクション処理のエラーコードを返す。

WkbGetUC
現在設定されているユーザーコード(User Code)を返す。

WkbListBox2
指定したWIBU-BOXの内容を調べます。

WkbListPort2
WIBU-BOXのシリアル番号をリスト表示する。

WkbOpen2
WIBU-BOXエントリをオープンする。

WkbProgram2
WIBU-BOXの内容を編集する

WkbSelect2
オープンされたWIBU-BOXエントリで暗号化のアルゴリズム変数を選択する。

WkbUnSelect2
選択されたWIBU-BOXエントリを非選択にします。

WkbWriteMem
割り当てられたメモリブロックにデータを書く。

WkxGetLastError
直前に発生した拡張APIファンクション処理のエラーコードを返す。

WkxGetRemoteContext
リモートコンテキストファイル(Remote Context File)を作成する。

WkxGetRemoteContext2
リモートプログラミングコンテキストデータを返す。

WkxSetLastError
拡張APIのエラーコードを設定する。

WkxSetRemoteUpdate
リモートアップデートファイルを適用する。

WkxSetRemoteUpdate2
リモートプログラミングアップデート処理を実行する。

WkbAccess2

ローカルサブシステムまたはネットワークサブシステムのWIBU-KEYにアクセスする。

[Syntax]
HWKBSYSTEM WkbAccess2 (ULONG flCtrl, const VOID *pvCtrl)

[内容]
このファンクションは、ローカルサブシステムまたはネットワークサブシステムのWIBU-KEYにアクセスします。

レベルの相違点
レベル 1 が指定された場合、WkLANサブシステム上でWIBU-BOXを検索するメッセージBOXが画面に表示されます。このメッセージは、WKBACCESSストラクチャのflCtrlメンバにWKB_ACC_NO_DIALOGオプションを指定することで、画面表示されなくなります。もし、レベル0が指定された場合は、メッセージBOXは表示されません。（レベルに関しては、スタンダードファンクションコントロールフラグを参照してください。）

flCtrl

WKB_ACC_CREATE (0x00)
新規のサブシステムを作成します。

WKB_ACC_RESTORE (0x10)
WkbQuerySystem2ファンクションとWKB_QSY_STATUSコントロールコマンドでステータスが保存されたサブシステムを復元します。

WKB_ACC_SHELL (0x20)
WIBU-BOXエントリでプロテクトされない新規のローカルサブシステムを作成します。WKBACCESSの中にあるFirm Code/User Codeだけがサブシステムに保存され、WKB_ACCESS_CODE定数をともなったWkbOpen2ファンクションで使用されます。シェルベースのサブシステムは、オープンされたWIBU-BOXのような割り当てられたリソースや、WkbRelease2ファンクションで割り当てられたメモリーブロックなどを一掃するために使われます。一掃されるリソースはすべてこのサブシステムで作成されたものです。

WKB_ACC_GLOBAL (0x30)
pvCtrlパラメータで指定されたWIBU-BOXエントリとバンドルした新規のローカルサブシステムを作成します。

WKB_ACC_CREATEを使う場合、次のフラグも一緒に使用することができます。

WKB_ACC_LOCAL (0x01)
指定したFirm CodeとUser Codeを持つローカルWIBU-BOXをオープンします。WIBU-BOXが見つかると、スタンダードサブシステムと互換性のあるサブシステムが使用され、HWKB_LOCAL定数でアドレスされます。

WKB_ACC_WKLAN (0x02)
指定したコントロールパラメータに従い、WkLANネットワークサブシステムにアクセスします。

WKB_ACC_WKNET (0x04)
指定したコントロールパラメータに従い、WkNetネットワークサブシステムにアクセスします。

これらの 3 つのフラグは、WKB_ACC_CREATEコマンドだけに使用可能で、他のコマンド（WKB_ACC_RESTORE, WKB_ACC_SHELL, WKB_ACC_GLOBAL）では使用できません。また、これらのフラグが 2 つあるいは 3 つ指定されている場合、WkbAccess2ファンクションは最初にローカルWIBU-BOXにアクセスし、次にWkLANサブシステムにアクセスし、最後にWkNetサブシステムにアクセスします。この順番がサブシステムにアクセスする最も早い方法になります。

次のフラグは、WKB_ACC_WKNETフラグと一緒に使用できます。他のフラグでは使用できません。

WKB_ACC_DIRECT (0x100000)

ユーザーのアロケーションを行わずにネットワークサブシステムを直接アドレスします。このオプションは、サブシステムの管理をコントロールするために使用し、通常はWkbControlSystem2ファンクションからコントロールします。

次のフラグは、WKB_ACC_CREATEコマンドと一緒に使用できます。他のコマンド（WKB_ACC_RESTORE, WKB_ACC_SHELL, WKB_ACC_GLOBAL）では使用できません。

WKB_ACC_NONETLIMIT (0x200000)

WkbAccess2で作成されたサブシステムを使用するWkbOpen2に対して、WKB_OPEN_NONETLIMITオプションを強制的に使用させるフラグです。このオプションを使ってWkLANサーバーにログインした場合、ネットワーククラスタにスロットを割り当てません。現在使用中のWIBU-BOXエントリが、WKB_OPEN_GLOBALオプションを使ってWkbOpen2ファンクションでオープンされていても、このWKB_ACC_NONETLIMITフラグを使うとあらたなローカルサブシステムを作成することができます。

WKB_ACC_STATIONSHARE (0x800000)

WkbAccess2で作成されたサブシステムを使用するWkbOpen2に対して、WKB_OPEN_STATIONSHAREオプションを強制的に使用させるフラグです。このオプションでWkLANサーバーにログインすると、同一のクライアントPC（同一のネットワークアドレス）上で別のプロセスがそのスロットを割り当てていない場合のみ、ネットワーククラスタにスロットを 1 つ割り当てます。その他の場合は、このスロットは新しいスロットとして使用されます。（「ステーションシェア」の状態）

これら 2 つのオプションが指定されていない場合は、起動するアプリケーションごとにネットワークユーザー制限数が割り当てられます。

次のフラグが使用できますが、WkNetサブシステムでは無視されます。このフラグは、WKB_ACC_STATIONSHAREフラグと一緒に使用することはできません。

WKB_ACC_EXCLUSIVE (0x400000)

アクセスしたWIBU-BOXエントリを排他的モード（Exclusive Mode)にします。このフラグが設定されている場合、WIBU-BOXエントリは指定したFirm CodeとUser Codeで一度だけアクセスされます。このフラグは、同じアプリケーションをローカルサブシステムやWkLANサブシステムで 2 つ以上同時に起動させたくない場合に使用します。

このパラメータは、スタンダードファンクションコントロールフラグを含みます。

pvCtrl

WKB_ACC_CREATEまたはWKB_ACC_SHELLの場合は、アクセス方法が指定されているWKACCESSストラクチャを指定。WKB_ACC_RESTOREの場合は、WkbQuerySystem2ファンクションでWKB_QSY_STATUSフラグを設定したときに返されるこのパラメータのアドレス。WKB_ACC_GLOBALの場合は、グ

ローバルに定義されたサブシステムで使われるWIBU-BOXエントリを指定したWKBOPENENTRYストラクチャ。これらのストラクチャは、WkbQueryEntry2のようなファンクションから受け取るか、メンバを書き込むことで作成されます。（WkbOpen2のWKB_OPEN_ENTRYオプション参照）

[リターン値]

このファンクションはオープンされたサブシステムのハンドルを返し、エラーの場合はNULLを返します。エラーの場合、WkbGetLastErrorファンクションがそのエラーコードを返します。エラーコードはサブシステムにアクセスしたときの方法や状況によって異なります。

利用可能なサブシステムの数とは別に、1 つだけのサブシステムをチェックした場合は、そのサブシステムのエラーコードが返ります。

１つあるいは複数のサブシステムが有害なアクセスエラー（WkNetファイルの破壊、利用可能なユーザーがいない、ローカルサブシステムの通信エラー）を返した場合、最後にアクセスしたサブシステムのエラーコードが返ります。

ローカルサブシステムがWKB_ERROR_ENTRY_NOT_FOUNDを返し、他のサブシステムも失敗した場合、WKB_ERROR_SERVER_NOT_FOUNDが返ります。

要求するサブシステムがアクティブでない場合、WKB_OPTION_NOT_SUPPORTEDが返ります。

[コメント]

WKB_ACC_LOCALオプションでローカルサブシステムにアクセスした場合、検知されたWIBU-BOXエントリはWKB_VERSION_STD定数を使ってオープンされます。（WkbOpen2参照）これは、アクセスファンクションが失敗していないことを保証するものです。古いドライバーを使用している場合、WIBU-BOX/Rシリーズでエラーが発生します。古いドライバーは、WKB_VERSION1を使用しているのに対し、WIBU-BOX/RシリーズはWKB_VERSION3だけをサポートしているからです。

[Platform Specific Limitations]

WKB_ACC_STATIONSHAREは、WIBU-KEYドライバーVersion3.0以降が必要です。
WKB_ACC_EXCLUSIVEは、WIBU-KEYドライバーVersion2.51以降が必要です。
WKB_ACC_NONETLIMITは、WKB_ACC_SHELLと一緒に使用するとWIBU-KEYドライバーVersion3.0以降が必要です。
WKB_ACC_SHELL, WKB_ACC_GLOBAL, WKB_ACC_WKLANは、DosLib static DOS ドライバーライブラリや、OS/2ライブラリでサポートされません。DosLibは、WkNetをサポートするにはWKDOSA.OBJが必要です。
WKB_ACC_RESTOREは、WKDOSA.OBJを拡張したDosLibドライバーによってのみサポートされます。

WkbAllocMem

WIBU-KEYドライバーの中にメモリーブロックを割り当てます。

[Syntax]
HWKBMEM WkbAllocMem (HWKBSYSTEM hwkbsys, UINT cdAlloc)

[内容]
このファンクションは、ドライバーのメモリーヒープの中に、指定した長さのメモリーブロックを割り当てます。

hwkbsys
メモリーブロックが割り当てられるサブシステムのハンドル。
サブシステムがWkbRelease2でクローズされると、メモリーブロックは自動的に開放されます。ローカルサブシステムのメモリーを割り当てる場合は、あらかじめ定義されたHWKB_LOCALを使用できます。

cbAlloc
割り当てるメモリーブロックの長さ。この値は、0から65500の範囲になります。

[リターン値]
このファンクションは、処理が成功した場合はハンドルを返し、エラーの場合はNULLを返します。WkbGetLastErrorファンクションで関連するエラーコードが返ります。

[Preliminary Note]
サブシステムを閉じても、割り当てられたメモリーブロックの一部が解放されないときがあります。

WkbBitResetSet

このファンクションは、論理的作業で32ビット整数値のビットを変化させます。

[Syntax]
ULONG WkbBitResetSet (ULONG ulSrc, ULONG ulResetMask, ULONG ulSetMask)

[内容]
このファンクションは、論理的作業で32ビット整数値のビットを変化させます。
最初は、指定されたソース値の中のビットがリセットマスク値の反対に"AND"でマスクされ、次に、その結果がセットされたマスク値で"OR"でマスクされます。そして、その最終結果が返されます。

ulSrc
修正される32ビット値。

ulResetMask
リセットマスク：この値でセットされたビットは全て次のビットをリセットします。

ulSetMask
セットマスク：この値でセットされたビットは全て次のビットをセットします。

[リターン値]
決定された値が返ります。

WKbClose2

WkbOpen2ファンクションでオープンされたWIBU-BOXエントリをクローズします。

[Syntax]
INT WkbClose2 (HWKBENTRY hwkbe)

[内容]
このファンクションはWkbOpen2ファンクションでオープンされたWIBU-BOXエントリをクローズします。

hwkbe
オープンされたWIBU-BOXエントリのハンドルを指定します。

[リターン値]
処理が成功するとTRUEを返し、エラーが発生するとFALSEを返します。WkbGetLastErrorファンクションで関連するエラーコードが返ります。

WkbControlSystem2

指定したWIBU-KEYサブシステムをコントロールする。

[Syntax]

INT WkbControlSystem2 (HWKBSYSTEM hwkbsys, ULONG flCtrl, const VOID *pvCtrl,
ULONG cbCtrl)

[内容]

このファンクションは、指定したサブシステムの実行をコントロールします。このファンクションはすべてのサブシステムで使用が可能です。

hwkbsys

有効にアクセスされたWIBU-KEYサブシステムのハンドル

flCtrl

ファンクションの実行をコントロールするフラグ。現在、次のコマンドフラグが定義されています。

WKB_CTS_RESET (0x00)

指定されたシステムをリセットします。このファンクションは、ローカルサブシステムだけに有効です。リセットオペレーションモードはcbCtrlパラメータの中で定義されます。pvCtrlパラメータはNULLにセットされなければなりません。

WKB_CTS_CANCEL_USERS (0x10)

WkNetやWkLANサブシステムの中で指定されたユーザーをキャンセルします。キャンセルされたユーザーは、WkbAccess2ファンクションで新規にコールされるまで、自分自身のサブシステムによるネットワークサーバーを使用することはできなくなります。

WKB_CTS_FREE_ACCESS (0x20)

ネットワーク内のユーザーを開放せずに指定されたサブシステムをフリーにする。この作業は、WKB_QSY_STATUSコマンドを伴うWkbQuerySystem2ファンクションとWKB_ACC_RESTOREコマンドを伴うWkbAccess2ファンクションを実行して、内部のサブシステムステートをセーブしながら行われます。

WKB_CTS_KILL_ACTION (0x30)

IDがcbCtrlパラメータの中に保存されているサブシステムのアクションを無効にします。この値が 0 の場合、指定されたサブシステムの全てのアクションは無効になります。

このパラメータは、スタンダードファンクションコントロールフラグを含みます。

pvCtrl

ファンクションの実行をコントロールするデータストラクチャのポインタ。

cbCtrl

pvCtrlポインタがNULLでない場合、pvCtrlポインタで指定されるデータストラクチャの長さ。
PvCtrlポインタがNULLの場合、このパラメータの意味は指定されたWKB_CTS_…オプションによって決まります。WKB_CTS_RESETオペレーションの場合、このパラメータは次の意味を持つフラグのビットマップを意味します。

WKB_RESET_SYSTEM (0x01)
将来の使用のためにリザーブされ、0 にセットされなければなりません。

WKB_RESET_ADDRESS (0x02)
プロファイル（WIBUKEY.INI）の中のアドレッシングスペックを再リードします。

WKB_RESET_TIMING (0x04)
WIBU-BOXタイミングを再計算します。

WKB_RESET_BOXSCAN (0x08)
次のアクセスファンクションがスタートした時に、全てのワイブキーの基本的な検索を再開します。このオプションは、マッキントッシュADBドライバーだけに使用されます。

WKB_RESET_FULL (0xFE)
現在と将来のリセットオプションを全て行なうマスク。

[リターン値]
オペレーションが成功するとTRUEを返し、エラーが生じるとFALSEを返します。WkbGetLastErrorファンクションで関連するエラーコードが返ります。

[Preliminary Note]
WKB_CTS_FREE_ACCESSコマンドはWkNetサブシステムだけをサポートします。

WkbCrypt2

指定されたWIBU-BOXエントリに従い、メモリー上でデータを暗号化・復号化します。

[Syntax]

INT WkbCrypt2 (HWKBENTRY hwkbe, ULONG flCtrl, VOID *pvDest, VOID *pvCtrl, UINT cbSrc, UINT *pcbDest)

[内容]

このファンクションは、WIBU-BOX経由でデータを暗号化・復号化します。使用されるアルゴリズムはWKBAREAストラクチャで直接指定するか、WkbSelect2ファンクション経由でオープンされたエントリのセレクションに従い実行されます。

kwkbe

オープンされたWIBU-BOXエントリのハンドルを指定します。

flCtrl

WKB_CRYPT_CTRL (0x0000)

pvCtrlパラメータには、初期設定されたストラクチャのポインタが指定されます。ストラクチャには暗号化をコントロールする追加データが指定されます。pvCtrlがNULLの場合、コントロールデータは設定されず、WkbSelect2ファンクションで選択する必要があります。

WKB_CRYPT_COPY (0x0001)

pvCtrlパラメータには、暗号化されるソースデータを含むバッファーのポインタが指定されます。pvCtrlはNULLを指定できません。このWKB_CRYPT_COPYオプションは、WkbSelect2ファンクションが選択されたときだけ使用できます。

WKB_CRYPT_BUFFER (0x0002)

暗号化がバッファ内部で実行されます。WKB_CRYPT_BUFFERオプションはダイレクト暗号化で、WKB_CRYPT_NOBUFFERオプションはインダイレクト暗号化です。

WKB_CRYPT_NOBUFFER (0x0004)

暗号化がバッファ外部で実行されます。WKB_CRYPT_BUFFERオプションはダイレクト暗号化で、WKB_CRYPT_NOBUFFERオプションはインダイレクト暗号化です。

さらに、次のオプションフラグは暗号化・復号化されるデータタイプをコントロールします。

WKB_AREA_DATA (0x0000)

自由に変更可能な標準データ。

WKB_AREA_CODE (0x0010)

OSやプログラミング言語上で動作する実行可能なコード。

WKB_AREA_NONE (0x0080)

指定したバッファ部分は暗号化しない。このオプションは、開発プロセスのテスト目的で使用します。

これらのフラグは、pvCtrlパラメータでコントロールデータのポインタが指定されていない場合のみ有効です。それ以外は、WKBAREAストラクチャおflCtrlメンバのフラグが使用されます。

このパラメータはスタンダードファンクションコントロールフラグを含みます。

pvDest
暗号化されるバイト数を受け取るバッファのポインタ。
flCtrlでWKB_CRYPT_CTRLオプションが指定されている場合は、暗号化されるデータアドレスのポインタになります。

pvCtrl
flCtrlパラメータの中のフラグとは独立して、異なる目的に使用されるバッファのポインタ。
WKB_CRYPT_COPYオプションが設定された場合、暗号化されるソースデータのポインタ。WKB_CTRLDATAオプションが設定された場合、暗号化タイプをコントロールするストラクチャのポインタ。この場合、ポインタはNULLになります。もしNULLでない場合、追加的コントロールデータを含むWKBAREAのポインタになります。このデータは、いくつかのメンバがWkbCrypt2ファンクションで更新されるため、固定した集合データとして保存してはいけません。NULLが指定できるのは、WkbSelect2ファンクションでアルゴリズムが選択された場合のみです。

cbSrc
暗号化されるバイト数。順列によるインダイレクトの暗号化・復号化が選択された場合、指定するバイト数は、演算対象の長さの倍数、1,2,4,8バイト、になります。暗号化・復号化がネットワークサブシステムで行われる場合、バイト数は、WKBSYSTEMストラクチャのcbMaxDataメンバで制限されます。WkNetサブシステムの場合、このバイト数は非常に小さく、例えば32バイトのようになります。

pcbDest
暗号化されるバイト数をデスティネーションアドレスで受け取る整数値のポインタ。ポインタがNULLの場合、データは保存されません。

[コメント]
直前のセレクションコールがダイレクト暗号化を選択した場合、WKB_CRYPT_BUFFERオプションがデフォルトとしてセットされます。一方、インダイレクト暗号化が選択されていた場合は、WKB_CRYPT_NOBUFFERオプションが使用されます。

[リターン値]
オペレーションが成功するとTRUEを返し、エラーが生じるとFALSEを返します。WkbGetLastErrorファンクションで関連するエラーコードが返ります。

WkbEnumPort2

利用可能なすべてのWIBU-KEYポートのリストを返します。

[Syntax]
INT WkbEnumPort2 (HWKBSYSTEM hwkbsys, ULONG flCtrl, TCHAR *pszDest, UINT cchDest)

[内容]
このファンクションは、指定したWIBU-KEYサブシステムでの利用可能なWIBU-KEYポートをすべて調べ、それらのすべてのポート名を指定したデスティネーションストリングバッファに書き込みます。バーティカルバー('|')でポート名を区切ります。

hwkbsys
アクセスするWIBU-KEYサブシステムのハンドル。ローカルシステムでWIBU-BOXを使用する場合は、あらかじめ定義されたハンドルHWKB_LOCALが使用できます。

flCtrl

WKB_EPT_LIST (0x0000)
pszDestパラメータで指定されたバッファが、すべての利用可能なWIBU-KEYポート名を含むストリングを受け取ります。ポート名は、バーティカルバー('|')で区切られ、0キャラクターで終了します。

WKB_EPT_NEXT (0x0010)
pszPortパラメータで指定されたバッファは、pszDestバッファで指定されたポート名の次に利用可能なポート名を含むストリングを受け取ります。もし、pszDestバッファで指定されたポート名が無く、あるいは(*)で指定されている場合、例えばLPT*のような場合、最初に利用可能なWIBU-KEYポート名を列挙します。すべての利用可能なポート名が返された後は、0が返ります。そのあとに実行されるWkbGetLastErrorファンクションでは、WKB_ERROR_INVALID_PORTエラーコードが返ります。

このパラメータはスタンダードファンクションコントロールフラグを含みます。

pszDest
すべてのポート名のリスト、あるいは次のポート名を含むストリングを受け取るバッファのポインタ。
後者の場合、バッファには、前のポート名が含まれたストリング、あるいは空のストリング（最初の場合）が含まれる必要があります。保存されるストリングは0キャラクタで終わります。cchDestで指定されたバイト数を超えてバッファに保存されることはありません。もし、cchDestが0の場合は、このパラメータは使用できません。

cchDest
指定したバッファの長さ（キャラクタでの）

[リターン値]
このファンクションは、次のどれか1つを返します。

1. エラーの場合は0を返します。WkbGetLastErrorファンクションで関連するエラーコードが返ります。
2. cchDestの長さが小さすぎる場合、必要なバッファ長（キャラクタでの）が返ります。WkbGetLastErrorファンクションで、WKB_ERROR_BUFFER_OVERFLOWコードが返ります。
3. cchDestの長さが十分な場合は、転送されたキャラクタ数が返ります。(!)は返りません。

[Preliminary Note]
このファンクションは、WkNetサブシステムでは使用できません。

WkbFreeMem

WIBU-KEYドライバー内のメモリーブロックを解放する。

[Syntax]
BOOL WkbFreeMem (HWKBMEM hwkbmem)

[内容]
このファンクションは、WkbAllocMemファンクションで割り当てられたメモリーブロックを解放します。

hwkbmem
WkbAllocMemで割り当てられたメモリーブロックのハンドル。

[リターン値]
オペレーションが成功するとTRUEを返し、エラーが生じるとFALSEを返します。WkbGetLastErrorファンクションで関連するエラーコードが返ります。

WkbGetFC

現在設定されているファームコード(Firm Code)を返します。

[Syntax]
INT WkbGetFC (LONG *plFirmCode, ULONG ulReserved)

[内容]
このファンクションは、現在のプロセスあるいはプログラムのファームコード(Firm Code)を返します。

plFirmCode
ファームコード(Firm Code)を受け取るロング変数のポインタ。

ulReserved
現在使用されておらず、0を設定します。

[リターン値]
このファンクションは、処理が成功するとWKB_NO_ERROR(0)を返し、エラーの場合はWKB_ERROR_NOT_CRYPTIC(10)を返します。

[コメント]
このファンクションは、Windows DLLやOS/2 DLLの一部でなく、コードに静的にリンクされるライブラリの一部です。したがい、マクロやVisual Basicプログラムからは実行することができません。

WkbGetLastError

最後に設定されたエラーコードを返す。

[Syntax]
INT WkbGetLastError (VOID)

[内容]
このファンクションは、WIBU-KEYドライバーにより最後に設定されたエラーコードを返します。
返すコードは、最後に実行されたファンクションがエラーを発生したときに、そのファンクションによって生成されます。エラーが発生しない場合は、コードは生成されません。

パラメータ
無し

[リターン値]
このファンクションは、最後に設定されたWIBU-KEYエラーコードを返します。

[Preliminary Note]
Windows 16bit OSでは、タスクの異なる値は保存されません。

WkbGetUC

現在設定されているユーザーコード(User Code)を返します。

[Syntax]
INT WkbGetUC (LONG *plUserCode, ULONG ulReserved)

[内容]
このファンクションは、現在のプロセスやプログラムのユーザーコード（User Code)を返します。

plUserCode
ユーザーコード(User Code)を受け取るロング変数のポインタ。

ulReserved
現在使用されておらず、0を設定します。

[リターン値]
このファンクションは、処理が成功するとWKB_NO_ERROR(0)を返し、エラーの場合はWKB_ERROR_NOT_CRYPTIC(10)を返します。

[コメント]
このファンクションは、Windows DLLやOS/2 DLLの一部でなく、コードに静的にリンクされるライブラリの一部です。したがい、マクロやVisual Basicプログラムからは実行することができません。

WkbListBox2

指定したWIBU-BOXの内容を調べます。

[Syntax]

INT WkbListBox2 (HWKBSYSTEM hwkbsys, ULONG flCtrl, const TCHAR FAR *pszPort,
const WKBSERIAL FAR *pwkbsr, VOID FAR *pvDest, UINT cdDest)

[内容]

このファンクションは指定したWIBU-BOXの内容をリスト表示します。

[Level Differences]

hwkbsys

WIBU-KEYサブシステムのハンドルを指定します。ローカルサブシステムでWIBU-BOXの内容をリスト表示する場合は、あらかじめ定義されたHWKB_LOCALを使用できます。

flCtrl

WKB_LIST_DATA (0x0000)

WIBU-BOXエントリのすべての内容を、pvDestで指定したWKBBOXDATAストラクチャに返します。

WKB_LIST_CONFIG (0x0010)

WIBU-BOXのコンフィグレーションデータをpvDestで指定したWKBBOXCONFIGストラクチャに返します。

WKB_LIST_EXTMEM (0x0020)

ExtMemストレージページの内容をpvDestで指定したWKBEXTMEMストラクチャに返します。

WKB_PORTNUMBER (0x0008)

ポート名の代わりにポート番号を指定します。

このパラメータはスタンダードファンクションコントロールフラグを含みます。

pszPort

１つのポートを指定します。WKB_PORTNUMBERフラグが設定された場合、ポインターとして解釈されないため、WIBU-KEY Version1と互換性のあるポートスペック番号を指定する必要があります。ポート番号からアドレスへの変換は、MAKEINTRESOURCEマクロを使って行います。

pwbksr

WKBSERIALタイプのシリアル番号を指定して内容を取得するWIBU-BOXを指定します。

pvDest

リスト情報を受け取るデスティネーションバッファを指定します。このバージョンでは、WKBBOXDATA, WKBBOXCONFIG, WKBEXTMEMストラクチャが使用されます。

cdDest

デスティネーションバッファの長さ（バイト）を指定します。WKBBOXDATA, WKBBOXCONFIG, WKBEXTMEMデータストラクチャの長さになります。

[リターン値]
以下の値の 1 つを返します。
* エラーの場合は0を返します。WkbGetLastErrorファンクションで関連するエラーコードを取得できます。
* cdDestで指定した長さが、実際に返されたデータよりも小さい場合、必要なバッファ長が返されます。WkbGetLastErrorファンクションでWKB_ERROR_BUFFER_OVERFLOWコードが返されます。
* cdDestで指定した長さが十分な場合、実際に転送したバイト数を返します。

[Platform Specific Limitations]
Level1は、DosLib static DOS ドライバーライブラリまたはOS/2ドライバーでサポートされません。

[コメント]
WKB_LIST_EXTMEMフラグを使って拡張メモリーページを読む場合、WIBU-KEY APIバージョン4以降が必要になります。

WkbListPort2

指定したWIBU-BOXポートで検索されたすべてのWIBU-BOXのシリアル番号をリスト表示します。

[Syntax]

INT WkbListPort2 (HWKBSYSTEM hwkbsys, ULONG flCtrl, const TCHAR FAR *pszPort,
VOID FAR *pvDest, UINT cbDest)

[内容]

このファンクションは指定したポートで利用可能なすべてのWIBU-BOXを調べます。検索されたWIBU-BOXのすべてのシリアル番号は指定されたバッファに保存されます。（ただし、バッファの長さが小さすぎる場合は例外です。）

hwkbsys

WIBU-KEYサブシステムのハンドルを指定します。ローカルサブシステムでリスト表示する場合は、あらかじめ定義されたHWKB_LOCALを使用できます。

flCtrl

WKB_PORTNUMBER (0x0008)

ポート名の代わりにポート番号を指定します。

このパラメータはスタンダードファンクションコントロールフラグを含みます。

pszPort

１つのポートを指定します。WKB_PORTNUMBERフラグが設定された場合、ポインターとして解釈されないため、WIBU-KEY Version1と互換性のあるポートスペック番号を指定する必要があります。ポート番号からアドレスへの変換は、MAKEINTRESOURCEマクロを使って行います。

pvDest

リスト情報を受け取るデスティネーションバッファを指定します。このバージョンでは、awkbsr[]配列の可変長を持つWKBBOXLISTストラクチャが使用されます。

cbDest

デスティネーションバッファの長さ（バイト）を指定します。

[リターン値]

以下の値の１つを返します。
* エラーの場合は0を返します。WkbGetLastErrorファンクションで関連するエラーコードを取得できます。
* cdDestで指定した長さが、実際に返されたデータよりも小さい場合、必要なバッファ長が返されます。WkbGetLastErrorファンクションでWKB_ERROR_BUFFER_OVERFLOWコードが返されます。
* cdDestで指定した長さが十分な場合、実際に転送したバイト数を返します。

WkbOpen2

指定したFirmCode/UserCodeを持つWIBU-BOXエントリをオープンする。

[Syntax]

HWKBENTRY WkbOpen2 (HWKBSYSTEM hwkbsys, ULONG flCtrl, const TCHAR *pszPort,
LONG lFirmCode, LONG lUserCode, VOID FAR *pvCtrl)

[内容]

このファンクションはWkbSelect2やWkbCrypt2ファンクションで暗号化・復号化を行なうために、ワイブキーの中のエントリをオープンします。ファンクションは３つのモードをサポートします。
第１のモード、WKB_OPEN_FINDは指定されたFirmCode/UserCodeを持つWIBU-BOXエントリを検索し、最初に見つけたエントリのハンドルを返します。WKB_OPEN_FINDは、全ての可能なポートを検索します。
第２のモード、WKB_OPEN_ENTRYは、前にWkbOpen2ファンクションによって検索され、WkbQueryEntry2ファンクションによって返されたエントリや、別のファンクション、例えばWkbListBox2ファンクションから返されたエントリをオープンします。
第３のモード、WKB_OPEN_MULTIFINDは、WKB_OPEN_FINDに似ていますが、指定されたFirmCode/UserCodeに一致するすべてのエントリ（１つのワイブキーの中）を同時にオープンします。

hwkbsys

WkbAccess2で取得したWIBU-KEYサブシステムのハンドルを指定します。 WIBU-BOXをローカルサブシステムでオープンする場合は、WkbAccess2で取得したハンドルの代わりに、あらかじめ定義されているHWKB_LOCALを使用することができます。

flCtrl

第一フラグではコマンドコードを指定します。

WKB_OPEN_FIND (0x0000)

指定されたFirm CodeとUser Codeを持つWIBU-BOXエントリを検索します。WIBU-BOXエントリの検索順序はいくつかのルールに従います。（後述のコメント文を参照してください。）

WKB_OPEN_ENTRY (0x0010)

pvCtrlパラメータで指定されたWIBU-BOXエントリをオープンします。

WKB_OPEN_MULTIFIND (0x0020)

指定されたFirm CodeとUser Codeに一致するWIBU-BOX（１個）の中にあるすべてのエントリをオープンします。WIBU-BOXの検索順序はWKB_OPEN_FINDと同じです。オープンされるエントリは、Firm Code/User CodeエントリとMasterエントリになります。
WkbQueryEntry2ファンクションでは、オープンされたすべてのエントリのリストを返します。（WKB_QE_MULTIENTRYが設定されている場合）

第二フラグでは、オープンされるWIBU-BOXのバージョンを指定します。サポートするアルゴリズムバージョンは下図を参照してください。

WKB_VERSION1 (0x0000)

アルゴリズムバージョン１を使用するためにWIBU-BOXをオープンする。

WKB_VERSION2 (0x0001)

アルゴリズムバージョン２を使用するためにWIBU-BOXをオープンする。

WKB_VERSION3 (0x0002)

アルゴリズムバージョン３を使用するためにWIBU-BOXをオープンする。

WKB_VERSION4 (0x0003)

アルゴリズムバージョン４を使用するためにWIBU-BOXをオープンする。

WKB_VERSION5 (0x0004)

アルゴリズムバージョン５を使用するためにWIBU-BOXをオープンする。

WKB_VERSION_RED (0x0006)

WKB_VERSION_STD (0x0007)

第三フラグでは、追加オプションフラグを指定します。この追加オプションフラグは、第一フラグでWKB_OPEN_FINDを使用したときのみ利用できます。第一フラグにWKB_OPEN_FIND以外のフラグを使用した場合は、追加オプションフラグは利用できません。

WKB_PORTNUMBER (0x0008)

ポート名の代わりにポート番号を使用する。

WKB_OPEN_NONETLIMIT (0x100000)

WIBU-BOXエントリがネットワークのユーザー制限数とは関係なくオープンされます。これは、WKB_SC_NETLIMIT (bit 30、WKBAREA参照）で設定されたセレクションコード(Selection Code)の使用を避け、WKB_OPEN_GLOBALフラグでグローバルにオープンされたWIBU-BOXエントリの使用を許可します。使用されるサブシステムがWkbAccess2ファンクションのWKB_ACC_NONETLIMITフラグで作成されている場合、このWKB_OPEN_NONETLIMITモードは自動的に引き継がれます。

WKB_OPEN_GLOBAL (0x200000)

WIBU-BOXエントリが使用するPC上でグローバルにオープンされます。このことは、オープンされているWIBU-BOXエントリに対して、他のオープン処理ができないことを意味します。ただし、例外として、上記のWKB_OPEN_NONETLIMITだけはオープン処理が可能になります。

WKB_OPEN_EXCLUSIVE (0x400000)

WIBU-BOXエントリが、指定したFirm CodeとUser Codeで排他的モード（Exclusive Mode）でオープンされます。このフラグを使用すると、現在オープンされているWIBU-BOXエントリに対して、別のアクセス処理ができなくなります。このフラグは、同じアプリケーションをローカルサブシステムやWkLANサブシステムで２つ以上同時に起動させたくない場合に使用します。この排他的モードは、WkbAccess2ファンクションでWKB_ACC_EXCLUSIVEフラグを使ってサブシステムを作成した場合、WkbOpen2ファンクションでWKB_OPEN_EXCLUSIVEフラグを設定しなくても、排他的モードがそのまま引き継がれます。

WKB_OPEN_STATIONSHARE (0x800000)

WIBU-BOXエントリが、指定したFirm CodeとUser Codeでステーションシェアモード（共有モード）でオープンされます。このモードは、WkbAccess2ファンクションでWKB_ACC_STATIONSHAREフラグを使ってサブシステムを作成した場合、WkbOpen2ファンクションでWKB_OPEN_STATIONSHAREフラグを設定しなくてもそのまま引き継がれます。それぞれの暗号化処理は、セレクションコード(Selection Code）でWKB_SC_STATIONSHAREフラグを設定する必要があります。（WKBAREA参照）設定しない場合は、暗号化処理が失敗します。

pvCtrlパラメータでコントロールストラクチャが指定されている場合は、flCtrlは使用できず、flCtrlには0が設定されなければなりません。

このパラメータは、スタンダードファンクションコントロールフラグを含みます。

pszPort
このポインタにNULLを設定する（通常はこの方法を推奨）と、利用可能なすべてのWIBU-KEYポートを最適化された順序でスキャンします。このポインタにアスタリスク"*"を設定すると、LPT1から始まりLPT2,,,COM1,COM2,, USB1,,,とすべてのポートを順番にスキャンします。また、ポート名を直接設定することも可能です。２つ以上の場合は、"|"で区切ります。さらに、アスタリスク"*"を使って、ポートグループ単位で"LPT*"や"COM*"のように設定することも可能です。

* flCtrlパラメータでWKB_OPEN_ENTRYを使用する場合は、このpszPortパラメータにはNULLを設定する必要があります。他の設定をしても無視されます。
* WKB_PORTNUMBERフラグを使用する場合は、このパラメータにはWIBU-KEY Version1と互換性のあるポートスペック番号を指定する必要があります。ポート番号をアドレスに変換するにはMAKEINTRESOURCEマクロを使用します。

lFirmCode
WKB_OPEN_FINDまたはWKB_OPEN_MULTIFINDを使用する場合は、Firm Code (1...16777216の範囲）を指定します。pvCtrlパラメータでコントロールストラクチャが指定されている場合はこのパラメータは無視されるため0に設定する必要があります。

lUserCode
WKB_OPEN_FINDまたはWKB_OPEN_MULTIFINDを使用する場合は、User Code (0...16777216の範囲）を指定します。pvCtrlパラメータでコントロールストラクチャが指定されている場合はこのパラメータは無視されるため0に設定する必要があります。

pvCtrl
WKB_OPEN_ENTRYを使用する場合、WIBU-BOXエントリをオープンするのに必要なすべてのデータを含むWKBOPENENTRYストラクチャを指定します。このストラクチャには、要求するWIBU-KEY暗号化アルゴリズム変数(WKB_VERSION1, WKB_VERSION2, WKB_VERSION3あるいはWKB_VERSION_STD)をflCtrlメンバに、ポート仕様をIportメンバに、アクセスされるWIBU-BOXのシリアル番号をwkbsrメンバに、要求するWIBU-BOXエントリのエントリインデックスをusEntryメンバに、それぞれ格納します。マスターエントリ(Master Entry)をオープンするには、lUserCodeメンバに要求するUser Codeを格納します。その他のストラクチャメンバは無視されます。これらのメンバが正しく初期設定されない場合は、メンバは0に設定する必要があります。（WkbQueryEntry2ファンクションで内容を受け取るような場合）

WKB_OPEN_NONETLIMITやWKB_OPEN_EXCLUSIVEのようなオプションは、このストラクチャのflCtrlメンバの中に、WKB_OE_NONETLIMITあるいはWKB_OE_EXCLUSIVEとして設定することができます。WKB_OPEN_FINDやWKB_OPEN_MULTIFINDコマンドが設定された場合、このパラメータはNULLになり、正しく初期設定されたWKBOPENFINDストラクチャを指す必要があります。このストラクチャでは、使用されるFirm CodeとUser Code、WIBU-KEY暗号化アルゴリズム変数、指定したWIBU-BOXのエントリがFIND処理で使用されていることを指定します。

[リターン値]
このファンクションは処理が成功した場合、オープンされたWIBU-BOXエントリのハンドルを返します。エラーの場合はNULLを返します。

１つのポートがpszPortパラメータで指定されている場合、そのエラーコードが返ります。例えば、WKB_ERROR_INVALID_PORTあるいはWKB_ERROR_NOT_READY

複数のポートが指定され検索された場合、エラーコードWKB_ERROR_ENTRY_NOT_FOUNDがエラーコードとして返ります。

WIBU-BOXエントリが見つかったが、そのエントリのWIBU-BOXバージョンが間違っており、他に正しいバージョンを持つエントリが見つからなかった場合、WKB_ERROR_INVALID_VERSIONが返ります。複数のポートを検索した場合でも同様です。

WIBU-BOXエントリが見つかったが、そのエントリが他のプロセスでロックされており、他に利用できるエントリが見つからなかった場合、WKB_ERROR_ENTRY_ALREADY_USEDが返ります。複数のポートを検索した場合でも同様です。

[コメント]

ポート名を指定するストリングの長さは512文字（英数字）までです。ポート名は大文字、小文字どちらでも使用できます。ポートの指定は、ポートグループごとにアスタリスクを使って指定することができます。例えば、LPTの場合、"LPT*"と指定することによって、LPT1から始まり利用可能なLPTすべてを指定できます。また、バーティカルバー"|"を使って、異なるポートを一度に指定することができます。例えば、"COM3|LPT1|LPT*|COM2"と設定すると、COM3から始まり、LPT1,次にLPTすべて、最後にCOM2をサーチします。

WKB_VERSION1フラグは、WIBU-BOXバージョン1,3あるいは4のエントリだけに使用できます。WKB_VERSION2フラグは、WIBU-BOXバージョン2,3あるいは4のエントリだけに使用できます。WKB_VERSION3フラグは、WIBU-BOXバージョン5だけに使用できます。

WKB_OPEN_FINDとWKB_OPEN_MULTIFINDの検索順序はいくつかのルールがあります。

pszPortがNULLの場合（推奨）、検索順序は最適化されます。hwkbsysにローカルあるいはWkLANサブシステムが指定され、さらに指定したFirm Code/User Codeがこのサブシステムのコードと一致した場合は、ポート検索が始まる前に、その指定されているWIBU-BOXエントリが優先されます。

pszPortがNULLで、HWKB_LOCALがhwkbsysに指定されているか、WkNetサブシステムが使用されている場合、検索順序はpszPortで"*"を指定した場合と同じなります。

ある指定した順序で複数のWIBU-BOXを検索する場合、pszPortパラメータで順序を指定します。ただし、この方法はpszPortパラメータでNULLを指定した場合よりも検索スピードが遅くなりますので、ご注意ください。

[サポートするアルゴリズムバージョン]

	シリアル番号 Mask Byte	WKB_VERSION1	WKB_VERSION2	WKB_VERSION3	WKB_VERSION4	WKB_VERSION5
WIBU-BOX/1	1	●				
WIBU-BOX/2,3,4	3, 4, 5, 6	●	●			
WIBU-BOX/5	7			●		
WIBU-BOX/6	8	●	●		●	
WIBU-BOX/7	9			●		●

[Platform Specific Limitations]

* WKB_VERSION_STD定数を使用するには、WIBU-KEYドライバー3.0以降が必要です。
* WKB_VERSION4, WKB_VERSION5, WKB_VERSION_RED定数を使用するにはWIBU-KEYドライバー3.0以降が必要です。
* DosLibドライバーライブラリとOS/2ドライバーは、WKB_OPEN_MULTIFINDコマンドとWKBOPENFINDストラクチャの設定をサポートしません。

[Preliminary Note]

WKB_OPEN_ENTRY, WKB_OPEN_MULTIFIND, WKBOPENFINDストラクチャはWkNetサブシステム上では使用できません。

WkbProgram2

WIBU-BOXの内容を編集する

[Syntax]
INT WkbProgram2 (HWKBSYSTEM hwkbsys, ULONG flCtrl, const TCHAR FAR *pszPort, WKBSERIAL const FAR *pwkbsr, const VOID FAR *pvPgmData)

[内容]
このファンクションは、指定したWIBU-BOXのエントリを変更したり、WIBU-BOXのコンフィグレーションを編集します。

hwkbsys
WIBU-KEYサブシステムのハンドルを指定します。ローカルサブシステムでWIBU-BOXをプログラムする場合は、あらかじめ定義されているHWKB_LOCAL定数を使用できます。

flCtrl

WKB_PGM_CLEAR (0x0000)
既存のエントリを完全に消去する。

WKB_PGM_NEW (0x0010)
新規の基本エントリ（ベースエントリ）を作成する。

WKB_PGM_ADD (0x0020)
既存の基本エントリ（ベースエントリ）に追加エントリを追加する。

WKB_PGM_MODIFY (0x0030)
既存の基本エントリ（ベースエントリ）を修正する。修正できるのは、ユーザーデータエントリだけで、他のエントリを修正するためには、エントリを消去し再度作成する必要があります。

WKB_PGM_MODIFY_ADDED (0x0040)
追加エントリのデータ内容を修正する。

WKB_PGM_CONFIG (0x0080)
WIBU-BOXのコンフィグレーションデータを修正する。

WKB_PGM_LOCK (0x0090)
プログラムできないようにWIBU-BOXをロックする。

WKB_PGM_NULOCK (0x00A0)
ロックされたWIBU-BOXを解除する。

WKB_PGM_EXTMEM (0x00B0)
pvPgmDataパラメータにあるWKBEXTMEMストラクチャの内容をExtMemストレージページにプログラムする。

次のオプションを上記コマンドに追加することができます。

WKB_PORTNUMBER (0x0008)
ポート名の代わりにポート番号を指定します。

このパラメータはスタンダードファンクションコントロールフラグも含みます。

pszPort
１つのポートを指定します。WKB_PORTNUMBERフラグが設定された場合、ポインターとして解釈されないため、WIBU-KEY Version1と互換性のあるポートスペック番号を指定する必要があります。ポート番号からアドレスへの変換は、MAKEINTRESOURCEマクロを使って行います。

pwkbsr
WKBSERIALタイプのシリアル番号を指定し、プログラムするエントリのWIBU-BOXを指定します。

pvPgmData
flCtrlパラメータに次のオプションが設定されている場合は、WKBPROGRAMストラクチャを指定します。WKB_PGM_CLEAR, WKB_PGM_NEW, WKB_PGM_ADD, WKB_PGM_MODIFY, WKB_PGM_MODIFY_ADDED
flCtrlパラメータにWKB_PGM_EXTMEMが設定されている場合は、プログラムされるメモリーページが指定されたWKBEXTMEMストラクチャを指定します。

WKBPROGRAMストラクチャにはプログラムされるデータが含まれ、必要であればセキュリティシグニチャも含まれます。このデータのflCtrlメンバーにはプログラムされるWIBU-BOXのバージョンデータ（WKB_VERSIONx)も含まれます。

[リターン値]
処理が成功するとTRUEを返し、エラーが発生するとFALSEを返します。エラーの場合、WkbGetLastErrorファンクションが関連するエラーコードを返します。

[コメント]
WKB_PGM_MODIFY_ADDEDオプションで、リミットカウンタ値と40ビットデータ値を修正することができます。WKB_DATA_SERIALVARYとWKB_DATA_COUNTVARYフラグはWKB_PGM_ADDコマンドとのみ使用できます。
有効期限（Expiration Dates)の直接の追加や修正はこのファンクションではサポートされません。有効期限（Expiration Dates)の追加および修正は、このファンクションとは別に行います。
オフセット値によるリミットカウンタの修正（WKBBOXENTRYストラクチャのWKB_LIMIT_OFFSET定数を使用して）は、Version4以降のWIBU-BOXで可能です。

WKB_PGM_EXTMEMフラグを使用しての拡張メモリーページのプログラムは、WIBU-KEY API バージョン4.0以降が必要になります。

WkbSelect2

オープンされたWIBU-BOXエントリで暗号化のアルゴリズム変数を選択する。このファンクションはWKBAREAストラクチャでWKB_AREA_SELECTオプションが設定されたWkbCrypt2ファンクションと同じです。

[Syntax]
INT WkbSelect2 (HWKBENTRY hwkbe, ULONG flCtrl, ULONG ulSelectCode, VOID *pvCtrl)

[内容]
このファンクションは、暗号化・復号化のために、指定されたセレクションデータを使って、オープンされているWIBU-BOXエントリを選択します。

hwkbe
オープンされたWIBU-BOXエントリのハンドル。

flCtrl
ファンクションの実行をコントロールするフラグ。コマンドコードは、WKBAREAストラクチャのflSelCtrlメンバと同じです。

このパラメータは、スタンダードファンクションコントロールフラグを含みます。

ulSelectCode
セレクションコード（Selection Code)。
この値は、WKBAREAストラクチャのulSelectCodeメンバと同じです。

pvCtrl
選択するためのオプショナルコントロールデータのポインタ。
このパラメータは、WKBAREAストラクチャのpvCtrlメンバと同じです。

[リターン値]
処理が成功するとTRUEを返し、エラーが発生するとFALSEを返します。エラーの場合、WkbGetLastErrorファンクションが関連するエラーコードを返します。

[コメント]
このファンクションは、WkbUnSelect2ファンクションが実行されるまでドライバーを内部的にロックします。

WkbUnSelect2

選択されたWIBU-BOXエントリを非選択にします。

[Syntax]
INT WkbUnSelect2 (HWKBENTRY hwkbe)

[内容]
このファンクションは、指定したWIBU-BOXエントリを「非選択」状態にします。「非選択」後は、そのエントリは、他のオープンされているWIBU-BOXエントリで選択できます。

hwkbe
オープンされているWIBU-BOXエントリのハンドル。

[リターン値]
処理が成功するとTRUEを返し、エラーが発生するとFALSEを返します。エラーの場合、WkbGetLastErrorファンクションが関連するエラーコードを返します。

[コメント]
WkbUnSelect2ファンクションでエントリを非選択状態にしたあとは、WkbSelect2ファンクションで選択するまで、WkbCrypt2ファンクションはエラーになります。
もし、WKBAREAストラクチャでWKB_AREA_SELECTオプションを使ってWkbCrypt2ファンクションを実行した場合、WkbCrypt2ファンクションが自動的に非選択プロセスを行うため、WkbUnSelect2ファンクションを実行する必要がありません。
WkbUnSelect2は、実際に選択されていないエントリに対しても使用することができます。

WkbWriteMem

メモリアドレスから割り当てられたドライバーメモリブロックにデータを書く。

[Syntax]
UINT WkbWriteMem (HWKBMEM hwkbmem, VOID *pvSrc, UINT cbData)

[内容]
このファンクションは、WkbAllocMemファンクションで割り当てられたメモリブロックへ、指定したソースアドレスから指定した長さでデータを転送します。

hwkbmem
WkbAllocMemファンクションで割り当てられたメモリーブロックのハンドル。

pvSrc
メモリーブロックに書かれるデータのポインタ。少なくてもcbDataかそれ以上のバイト数を指定する必要があります。このパラメータには、別に作成されたメモリーブロックのハンドルも指定することができます。

cbData
転送されるバイト数。割り当てられたメモリーブロック以上のバイト数を転送することはできません。

[リターン値]
このファンクションは、実際に転送したバイト数を返します。もし値が0の場合は、エラーが発生したことになります。エラーの場合、WkbGetLastErrorファンクションが関連するエラーコードを返します。

WkxGetLastError

[Syntax]

WKEXT_API int WKAPIENTRY WkxGetLastError ( );

[内容]

拡張APIのエラーコードを返します。エラーコードは、WKX_ERROR_XXXで始まります。

WkxGetRemoteContext

[Syntax]
WKEXT_API int WKAPIENTRY WkxGetRemoteContext (const char *pszPortSpec, unsigned long flCtrl, unsigned int fuEntries, int fOnlyEmpty, int fAppend, const char *pszRtcFile);

[内容]
このファンクションは、リモートコンテキストファイル(Remote Context File)を作成する拡張APIファンクションです。pszPortSpecで指定されたWIBU-BOXを調べ、その内容をリモートコンテキストファイルpszRtcFileに書きます。

pszPortSpec
コンテキストファイルに保存するWIBU-BOXを指定する

flCtrl
現在使用しません。(将来のための予約パラメータ)

fuEntries
どのエントリの内容を保存するかを指定する

fOnlyEmpty
"空のエントリだけに保存する"フラグ

fAppend
"既存のファイルに追加する"フラグ

pszRtcFile
RTCのファイル名

[リターン値]
このファンクションは、処理が成功すると1を返し、失敗すると0を返します。エラーの場合、WkxGetLastError()ファンクションがその関連するエラーコードを返します。

WkxGetRemoteContext2

[Syntax]

WKEXT_API int WKAPIENTRY WkxGetRemoteContext2 (WKXRMT_CONTEXT *pwkxRtc, WKXSTATUS *pwkxStatus);

[内容]

このファンクションは、指定したパラメータとデスティネーションデータに関連するリモートプログラミングコンテキストデータを返します。

pwkxRtc

ファンクションパラメータを含むWKXRMT_CONTEXTストラクチャへのポインタ。

pwkxStatus

最後に行った処理の状態をリポートするWKXSTATUSストラクチャへのポインタ。

[リターン値]

ファンクションは、リモートプログラミングコンテキストデータとして返されたバイト数を返します。エラーの場合は0を返します。エラーの場合、WkxGetLastError()ファンクションがその関連するエラーコードを返します。

WkxSetLastError

[Syntax]
WKEXT_API void WKAPIENTRY WkxSetLastError (int idError);

[内容]
このファンクションは、拡張APIのエラーコードを設定します。エラーコードはWKX_ERROR_XXXで始まります。

idError
設定するエラーコード

WkxSetRemoteUpdate

[Syntax]
WKEXT_API int WKAPIENTRY WkxSetRemoteUpdate (const char *pszRtuFile, int fDelete);

[内容]
このファンクションは、リモートアップデートファイルを適用する拡張APIファンクションです。pszRtuFileからリモートアップデートファイルを読み込み、その内容をローカルシステムに存在する指定されたWIBU-BOXに書き込みます。

pszRtuFile
RTUファイル名

fDelete
"使用後、ファイルを消去する"フラグ

[リターン値]
処理が成功すると1を返し、エラーの場合0を返します。エラーの場合、WkxGetLastError()ファンクションがその関連するエラーコードを返します。

WkxSetRemoteUpdate2

[Syntax]

WKEXT_API int WKAPIENTRY WkxSetRemoteUpdate2 (WKXRMT_UPDATE *pwkxRtu, WKXSTATUS *pwkxStatus);

[内容]

このファンクションは、リモートプログラミングアップデート処理を実行します。

pwkxRtu

ファンクションパラメータを含むWKXRMT_UPDATEストラクチャへのポインタ。

pwkxStatus

最後に行った処理の状態をリポートするWKXSTATUSストラクチャへのポインタ。

[リターン値]

処理が成功すると1を返し、エラーの場合0を返します。エラーの場合、WkxGetLastError()ファンクションがその関連するエラーコードを返します。

サンカーラ株式会社
〒103-0013 東京都中央区日本橋人形町3-3-12 人形町103ビル2F
TEL: 03-3249-3421 / Fax:03-3249-3444
E-mail: support@suncarla.co.jp
www.suncarla.co.jp