WibuKey

Developer Guide

for Windows, Mac OS, Linux/UNIX
on IBM compatible PCs and Macintosh

Version 6.11, Edition February 2013

Wibu-Systems is member of:

PC Card	PCMCIA since 1993
USB	USB Implementers Forum since 1997
SD	SD Card Association since 2007
SIIA	SIIA Software & Information Industry Association since 1998
BITKOM	Bitkom, Bundesverband Informationswirtschaft, Telekommunikation und Neue Medien (German Association of Information Technology, Telecommunications, and New Media) since 2003
VDMA	VDMA, Verband Deutscher Maschinen- und Anlagenbau e.V. (German Engineering Federation) since 2008

and also member of the Developers programs of Autodesk, Apple, HP, IBM, Intel and Microsoft.

Microsoft GOLD CERTIFIED Partner OEM Hardware Solutions	Microsoft Gold Certified Partner
Windows® Embedded	Microsoft Embedded Partner

Table of Contents

Part I Preface

1 About this Guide

WibuKey is the technology of Wibu-Systems providing secure protection and effective license management of software and digital content. The *WibuKey* Developer Guide is divided into the following parts.

This preface gives you an overview of the Guide's structure, informs on typographic conventions used, and helps you in contacting the support team of Wibu-Systems.

Part II introduces into the concept of the *WibuKey* software protection and licensing, presents key features of *WibuKey*, sketches the *WibuKey* network support, describes how to use *WibuKey* and *WibuBoxes* in the field, and presents a list you should check when copy protecting your digital content (see page 16).

Part III informs you on installing the *WibuKey* Software (see page 62).

Part IV deals with protecting software and digital content and points to the automatic and individual integration of the protection into your software using *AxProtector* and *IxProtector* (WUPI) (see page 65).

Part V comprises the applications you have at hand for programming of *WibuBoxes* including remote programming (`WkList`, `WKCRYPT`) (see page 208).

Part VI explains how to implement network protection when using *WibuKey* (`WkLan` and *Huge File Management*) (see page 235).

Part VII informs on how to control and configure *WibuBoxes* (*WibuKey Control Panel Applet* and `wku`) (see page 272).

Part VIII follows with a description of deployment options: what does your customer need for running the protected software, and how to setup the *WibuKey* Runtime (see page 301).

Part IX overviews the *WibuKey Classic API* (see page 319).

Part X introduces the *WibuKey Interactive API Guide* (see page 338).

Part XI refers to the *WibuKey COM Control* (see page 348).

Part XII lists some more additional tips (see page 349).

Part XIII comprises several appendices with helpful information when using *WibuKey* (see page 353).

The Guide closes with an index.

2 Other Help Documents

In addition to this Developer Guide (accessible via **"Start | Program Files | WibuKey | Documentation"**), the following help files are available. You open them either in the respective applications, or you find them in the **"Start | Program Files"** menu after installing the SDK (*Software Development KIT*).

Help File	Accessible by
WibuKey User Help as compiled HTML online help	menu items or buttons in the application or by **"Start	Program Files	WibuKey"**
AxProtector Help and *Software Protection API* (WUPI) as compiled help files	**"Start	Program Files	AxProtector	Help"**
COM API online help	**"Start	Program Files	WibuKey| Development Kit	Documentation"**
COM-Control Online Guide	**"Start	Program Files	WibuKey| Development Kit	Documentation "**
Samples Overview & help	**"Start	Program Files	WibuKey| Development Kit	Documentation "**
WibuKey Extended Memory	**"Start	Program Files	WibuKey| Development Kit	Documentation "**
Wibu Error Messages	**"Start	Program Files	WibuKey| Development Kit	Documentation "**

3 Typographic Convention

This manual uses the following semantic markups, text emphases, and symbols.

Format Definition	Type of Information
Italics	*Product names*
SMALL CAPS	ESSENTIAL NOTIONS
SMALL CAPS ITALICS	*PROPERTY ELEMENTS*
"Bold Double Quote"	**Objects you select or use, e.g. menus, buttons, or notions in a list**
"BOLD SMALL CAPS DOUBLE QUOTE"	**COMMAND NAME**
`CAPITAL LETTERS COURIER NEW`	Key strokes on the keyboard, e.g. `SHIFT`, `CTRL` or `ALT`

Format Definition	Type of Information
`Courier New`	`Path specification or file names`

Pictogram	Description
	This symbol refers to important and essential instructions you should follow.
	This symbol refers to additional information of general interest.
	This symbol refers to an example which explains a feature.

4 Contacting Support

Our customers are supported by a professional team of exceptionally qualified staff. Our direct customer contact allows us to meet customer requests as fast as possible.

Enduser Support

Wibu-Systems provides a free-of-charge user hotline for your end customers.

Developer (Customer Support)

We are available in Germany workdays (Monday through Friday) from 8 a.m. to 5 p.m. per phone (+49-721-93172-14) or per e-mail (support@wibu.com). Wibu Systems USA support is available Monday through Friday from 8 a.m. to 5 p.m. PST by phone at 800-6-GO-WIBU (425-775-6900) or by e-mail (support@wibu.us).

Many of our distributors also provide support. Please contact your distributor to see if this service is available to you and your customers locally.

Please state your customer number which helps us to deal with your request as fast as possible.

Support Information

For best handling of your request we need the following information:

- type of protection implementation (automatic / individual)
- operating system
- version of *WibuKey* software installed
- *WibuKey* hardware you may use
- detailed error description

Support agreements with extended services on inquiry.

5 About Wibu-Systems

WIBU-SYSTEMS AG was founded in 1989 by Oliver Winzenried and Marcellus Buchheit with a mission to provide state-of-the-art solutions for protecting and licensing software and digital media. Products from Wibu-Systems support virtually all operating systems and come in a broad variety of form factors, including independency and the variety of form factors, including USB, PC Card, Express Card|34, Compact Flash Card, SD Card, MicroSD-Card, and ASIC. Applications include software for desktop PCs, servers, embedded systems, mobile, smart phones, and cloud computing.

Wibu-Systems is a privately-held corporation with a worldwide staff of 80, the majority in the headquarters facility in Karlsruhe, Germany. Subsidiaries are in Seattle (USA),Shanghai, and Beijing (China), with sales offices as well in Belgium, Great Britain, the Netherlands, Portugal and Spain, and distributors in more than 25 countries. Corporate efforts stress achieving world-class quality in the areas of security, reliability, durability, support, and customer service.

More than 6,000 independent software vendors (ISV) rely on *WibuKey, CodeMeter*, and *CodeMeterAct* to sell more products by reducing piracy and increasing the flexibility of their licensing models. Products include:

- *WibuKey*, a solution for software protection and digital content that is well proven since 1989. It is available in different form factors for nearly all interfaces
- *CodeMeter*, Wibu's newest architecture, allows for multiple ISVs to share a single dongle, easy online license transfers, and up to 8GB of flash memory
- *CodeMeterAct* is a software-based solution that protects software by binding to the characteristics of an individual PC
- *CodeMeter License Central* creates, managers, and delivers licenses with integration into sales and ERP systems
- *SmartShelter* allows for secure encryption of PDF documents
- *CodeMeter SDL* (Secure Data Layer) protects data files including audio, video, and database
- *Authentication Solutions* allow for easy and safe access to websites and hosted software applications (SaaS).

Wibu-Systems is a certified ISO 9001:2008 manufacturer and is an active member of BITKOM, VDMA, SIIA, and participates with standards organizations such as PCMCIA, USB Implementers Forum, and the SD Card Association. Additionally, Wibu-Systems is a Microsoft Gold Certified Partner, Windows Embedded Partner, and partner in developer programs of Apple, Adobe, Autodesk, Wind River, and others. Products from Wibu-Systems have received multiple industry awards including the SIIA CODiE Award for "Best Digital Rights Management" solution and the international iF Product Design Award. The company is leading different research project with universities and other companies, in parts funded by the German BMBF and BMWi. Examples include Pro-Protect with the aim of providing effective solutions to the manufacturing sector against product counterfeiting, VitaBIT for secure mobile solutions for health care, SumoDacs with solutions for mobile data access, and both MimoSecco and S4Cloud in the realm of cloud computing.

Contact information		
Germany	+49-721-93172-0	sales@wibu.de
USA	+1.425.775.6900	info@wibu.us
China	Shanghai +86-21-55661790	info@wibu.com.cn
	Beijing +86-10-82961560/61	info@wibu.com.cn
Great Britain	+44 (0)20 314 747 27	sales@wibu.co.uk
Ireland	+44 (0)20 314 747 27	sales@wibu.uk.co
Netherlands	+31 (0)74 75 01 495	sales@wibu-systems.nl
Belgium	+32 (0)3 400 03 14	sales@wibu.be
Spain	+34 (0) 91 414 8768	sales@wibu.es
Portugal	+34 (0) 91 414 8768	sales@wibu.pt
Other countries	+49-721-93172-0	sales@wibu.com

Part II Introduction

1 The *WibuKey* concept

Before you continue to read the following chapters you should be familiar with the terms concerning the *WibuKey* protection system and the field of copy protection. This part gives an explanation of important terms and protection methods. You will find a list of hardware and operating systems for *WibuKey* and also a brief introduction of different encryption methods.

1.1 What is *WibuKey*?

WibuKey is an intellectual property protection system. It works to prevent the use of illegal copies of software and data files. *WibuKey* consists of encryption and driver software, and a hardware component called the *WibuBox*. The principle of the protection is the encryption of the intellectual property. License checks and encryption are built into the software or data file to be protected. These access the *WibuBox*, which is connected to the computer. The *WibuBox* confirms the license checks and implements the decryption of the encrypted program or data file. Without the *WibuBox*, the protected software cannot be used and the protected data files cannot be read. This decryption cannot be replicated by a software process, because it relies on certain secrets which are built directly into the *WibuBox*. Wibu-Systems guarantees the non-reproducibility of the *WibuBox*. This way, no more copies of the program can be used than defined in the *WibuBox* hardware.

1.2 What is a *WibuBox*?

The *WibuBox* is the *WibuKey* hardware component. There are many different *WibuBoxes* form factors, all of which contain the same memory chip.

The *WibuBox* is available for PCs, Macs, and UNIX machines. Because all of the different *WibuBox* hardware variants contain the same memory chip type, they all operate identically. In fact, no special work is necessary to support any specific *WibuBox*. *WibuKey* automatically detects and utilizes an appropriately configured *WibuBox* no matter what type it is, or what port it is connected to. This gives you the greatest degree of flexibility, because you can deliver different *WibuBox* hardware to different customers depending on their requirements without writing any special code, or doing any extra work.

Table 1 lists the *WibuBox* form factors currently available (for more details see the data sheets in the download section on the Wibu-Systems website www.wibu.com.

Form Factor	Description

	Form Factor	Description
	*WibuBox/U+*	Universal Serial Bus (USB) for PC, Mac and UNIX, 10 entries, 16 kByte memory (8 kByte protected, 8 kByte freely accessible).
	*WibuBox/RU+*	Universal Serial Bus (USB) for PC and Mac, 1 entry, 16 kByte memory (8 kByte protected, 8 kByte freely accessible).
	*WibuBox/P+*	PC parallel (printer) port (LPT1 to LPT3), 10 entries, 16 kByte memory (8 kByte protected, 8 kByte freely accessible).
	*WibuBox/RP+*	PC parallel (printer) port (LPT1 to LPT3), 1 entry, 16 kByte memory (8 kByte protected, 8 kByte freely accessible).
	*WibuBox/ST*	Serial asymmetric port (RS-232).
	*WibuBox/M+*	PCMCIA (PC Card) slot for laptops. 10 entries, 16 kByte memory (8 kByte protected, 8 kByte freely accessible)
	*WibuKey ASIC*	Universal Serial Bus (USB). 10 entries, 16 kByte memory (8 kByte protected, 8 kByte freely accessible).

Table 1: *WibuKey* form factors

Detailed information about the use and handling of the extended memory can be found in the separate document `WibuKey ExtMem.doc` on your *WibuKey* CD-ROM.

In this documentation, the term *WibuBox* is used as a synonym for all these form factors. In the case of differences between the form factors the exact designation is given.

The main task of a *WibuBox* is to encrypt or decrypt incoming data byte-by-byte, and return the created data in the same fashion. For this purpose, a complex communication protocol is employed. The *WibuKey* user does not need to know this protocol, since powerful software drivers exist to handle the box communication. The application calls these drivers which automatically control the location of and the adaptation to the different *WibuBox* variants at runtime. You can use different ports on different computers without any additional coding or configuration.

In addition, each *WibuBox* possesses a unique SERIAL NUMBER which is used for identification purposes by the drivers. This unique serial number helps to realize the secure remote programming of a *WibuKey*. This feature lets you change the contents of a *WibuBox* at your customers.

The *WibuBox* hardware is programmed by the software developer. One *WibuBox* can hold up to 10 programmed entries consisting of a pair of numbers, called the FIRM CODE and the USER CODE. The *WibuBox/RP+* and *WibuBox/RU+* can hold only one entry. Both are explained in the next section. The *WibuBox* can contain other types of information, such as EXPIRATION DATES, LIMIT COUNTERS for software metering, secure configuration data, and user editable data. The fact that pre-programmed *WibuBoxes* cannot be procured from Wibu-Systems rules out the possibility of tampered orders.

Programming *WibuBox* hardware is fast and easy. The *WibuKey* software tool called **`WKLIST`** allows you to program *WibuBox* hardware using an intuitive GUI (see page 209). For faster production, it is possible to simultaneously program up to 20 *WibuBoxes* connected in series with the *WibuKey* commandline tool **`WKCRYPT`** (see page 216).

The *WibuBox* is designed to be transparent to other devices that need to use the hardware port. For example, printers can be connected behind the *WibuBox/P* and a mouse can be connected behind the *WibuBox/SP*. In addition, multiple *WibuBox* hardware can be connected in a series on the same port.

1.3 What are the FIRM CODE and the USER CODE?

The **FIRM CODE** and **USER CODE** are basisc constituents of the *WibuKey* protection system. Both of them are 24-bit numbers equivalent to values between `1` and `16,777,215`. These numbers are used to create unique *WibuBox* hardware. They are used mathematically in the encryption/ decryption process.

The standard entry in a *WibuBox* is the FIRM CODE/ USER CODE entry. It is called a **BASE ENTRY**. It is indicated by a square ☐ in the *WibuKey* software (see figure below).

The FIRM CODE and USER CODE are the two most important numbers in the *WibuKey* system, and absolutely necessary for the protection of programs or data.

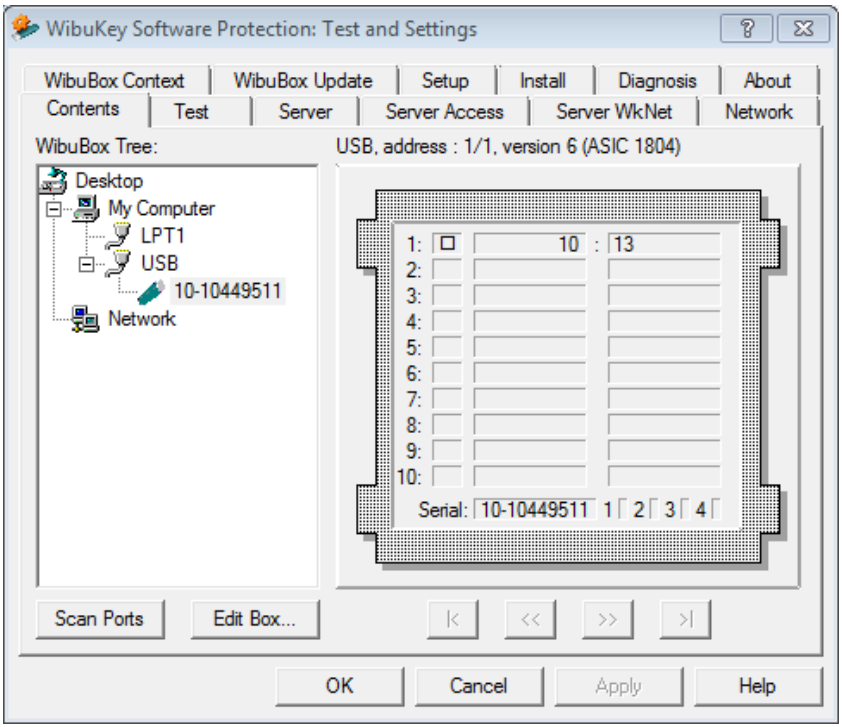


Figure 1: *WibuKey Control Panel Applet* (Advanced Mode) showing a FIRM CODE/USER CODE entry

The FIRM CODE is a unique number assigned to each Wibu-Systems customer. The software developer receives the FIRM CODE by ordering the *WibuKey* FIRM LICENSE.

***WibuKey* FIRM LICENSE**

The FIRM LICENSE consists of a disk with a FIRM SEQUENCE stored in the **WKFIRM.WBC** file and a **FIRM SECURITY BOX** (FSB). To use a FIRM CODE with *WibuKey*, the proper FIRM SECURITY BOX and the corresponding FIRM SEQUENCE from the disk need to be available. With these it is possible to program and alter

entries in the *WibuBox* that contain the corresponding and unique FIRM CODE. Wibu-Systems guarantees that each FIRM CODE is one-time assigned only.

Special **FIRM CODES**

FIRM CODES higher than `100` are allocated to software developers who have purchased a FIRM LICENSE. For demonstration and test purposes, Wibu-Systems has reserved the FIRM CODE `10`. This is not suitable for software protection. It is available to anyone who purchases a Protection Kit from Wibu-Systems. Nevertheless, with the FIRM CODE `10` you can explore and test all features offered by a regular FIRM CODE.

The **USER CODE** is not confidential and can therefore exist multiple times. Its purpose is to allow different software products or modules to be issued with different encryption and differently programmed *WibuBoxes*. Another option is to deliver a differently programmed *WibuBox* to each customer (see page 52). The allocation of different USER CODES to different customers prevents the *WibuBox* hardware from being exchanged between customers. This way an expensive software version with more features becomes not accessible via the box of an inexpensive version.

The FIRM CODE and the USER CODE are programmed into the *WibuBox* by the software developer supported by the **`WKLIST`** program (see page 209). One *WibuBox* can contain up to 10 FIRM CODE/USER CODE pairs. These entries may even be programmed into one *WibuBox* by different software developers. This guarantees an efficient use of the *WibuBox* for both, software developer and customer.

Reserved USER CODE

A USER CODE in the range from `15,728,640` (`0xF00000`) up to `16,777,215` (`0xFFFFFF`) is reserved for the *Huge License Management* (see page 259). Such a USER CODE should not be used for application protection, because it cannot be used for network protection.

1.4 *WibuKey* protection by encryption

This section shows the principle of encryption and decryption with *WibuKey* and the different protection methods. *WibuKey* protects intellectual property through a sophisticated encryption process. This encryption process is what makes *WibuKey* especially secure.

As noted above, the FIRM CODE and USER CODE are used in the encryption process. For each software developer, the encryption produced by *WibuKey* is different because of the different FIRM CODES.

SELECTION CODE

The USER CODE and an additional run-time code called a **SELECTION CODE** offer each software developer almost unlimited possibilities of encryption variation

because each variation of any of these numbers produces a completely different encryption sequence. To work with an encrypted program it is necessary that the encryption and decryption algorithms of the protected program are compatible to the algorithms in the *WibuBox*. To be compatible the algorithms must possess the same FIRM CODE, USER CODE and SELECTION CODE. Should any of these parameters not comply the program cannot be decrypted for use at the end user's side.

AUTOMATIC PROTECTION

WibuKey offers two styles of encryption, *AUTOMATIC PROTECTION* and *INDIVIDUAL PROTECTION*. The ***AUTOMATIC PROTECTION*** using *AxProtector* creates an encrypted version of a program or data file from its original. It does not require any source code modification.

INDIVIDUAL PROTECTION

The ***INDIVIDUAL PROTECTION*** involves using the *Wibu Universal Protection Interface* (Wupi) as part of *AxProtector* and adding function calls to the *WibuKey API* into your application. Individual.

With both *AUTOMATIC PROTECTION* and *INDIVIDUAL PROTECTION* *WibuKey* uses a sophisticated 3-step encryption process that is based on the FIRM CODE, USER CODE, and SELECTION CODE as well as numeric constants built into the *WibuKey* ASIC.

The following diagram shows the flow of the encryption process in the *WibuBox*:

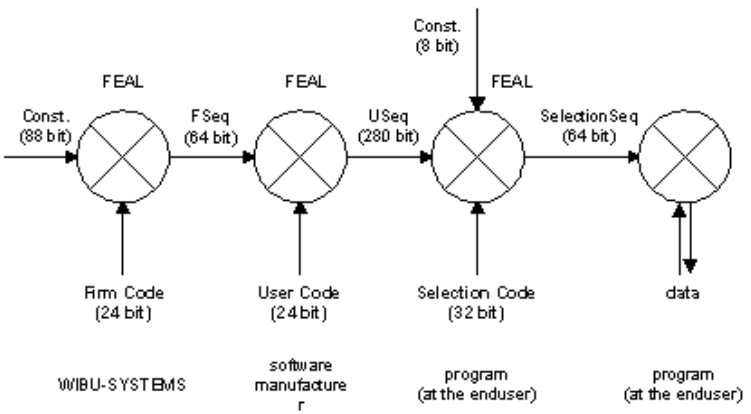


Figure 2:*WibuKey* encryption algorithms

Any encryption performs three steps:

- Constants and the FIRM CODE are used to calculate a FIRM SEQUENCE (Fseq) at the initialization of the encryption. The FEAL algorithm (NTT, Japan) with a round factor of 8 is used. This is an accepted symmetric algorithm, a block chiffre procedure with a 64 bit key length.

- The USER SEQUENCE (USeq) is calculated from the FIRM SEQUENCE through a FEAL encryption with the USER CODE.
- The USER SEQUENCE is used for calculating the SELECTION SEQUENCE (SSeq) with an additional FEAL encryption together with another constant and the SELECTION CODE. This SELECTION SEQUENCE initializes a random number generator.

After the initialization the data is encrypted byte-by-byte with this encryption sequence.

Version	Variants	External Memory	Algorithms	Mask Byte
1	P, SP	no	1, 2	1
3	P, SP	no	1, 2	3
4	P,M,U	no	1, 2	4
4a	ST	no	1, 2	5
4b	P,M,U	no	1, 2	6
4c	P, U	no	1, 2, 4	8
5	RP, RU	no	3	7
5a	RP, RU	no	3, 5	9
6	P+, U+	yes	1, 2, 4	8, 10
7	RP+, RU+	yes	3, 5	9, 11

Table 2: *WibuBox* variants and corresponding algorithms

If you have already shipped *WibuBoxes* to customers, please make sure that you use algorithm 1 or 2 for software updates. If you deliver new software with new *WibuBoxes*, please always use the highest available algorithm for highest security!

In addition to being a part of the encryption process, the SELECTION CODE IS used to activate additional *WibuKey* features such as network licensing and software metering. The highest bits – bit 28 and STATION SHARE and bit 31 – have special meanings:

- If bit 29 is set to 1, the EXPIRATION DATE feature is activated.
- If bit 30 is set to 1, the Network License Management feature is activated.
- If bit 31 is set to 1, a *WibuKey* LIMIT COUNTER feature is activated.

SELECTION CODE

All 32 bits of the SELECTION CODE affect the encryption. An encryption set to change the LIMIT COUNTER creates a different encryption sequence than the encryption that does not change the LIMIT COUNTER even if the values in bit `0` to `30` are identical. This makes it extremely difficult to tamper with *WibuKey* features.

In the case of the **direct encryption**, each byte which is to be decrypted, is sent to the *WibuBox* for decryption and returned to substitute the non-encrypted byte. Hence, when `1,000` bytes are to be encrypted, `1,000` bytes must be sent to the *WibuBox* and collected again. This can lead to delays which are possibly undesired in practice.

For this reason, the additional **indirect encryption** has been introduced where the encryption process is started in the *WibuBox* and completed in system memory for faster performance. Using the indirect encryption mode, it is possible to employ different encryption algorithms for a variety of run-time encryption needs.

1.5 Summary

The high degree of protection with *WibuKey* is of a triple hierarchical nature:

- The FIRM CODE: Each software developer receives his/her own FIRM CODE. S/he exclusively receives it in the form of a file named `WKFIRM.WBC` and his/her own FIRM SECURITY WIBUBOX. These two objects together allow *WibuKey* to create encryption algorithms for that specific FIRM CODE.
- The USER CODE: The software developer can issue his/her software with different USER CODES for different programs, modules, or even for different users. When using different USER CODES for different users, each customer receives a version of the program which is encrypted differently and consequently requires a different decryption. For each customer, the decryption must naturally be done by a connected *WibuBox* which has been programmed with his/her USER CODE. Different customers cannot exchange software copies. Different USER CODES are important for databases and similar applications.
- The SELECTION CODE: Because a number of sections may be differently encrypted and decrypted within a program or data file, the 32 bit SELECTION CODE allows more than 4 billion different encryption and decryption algorithm variants to be used for each FIRM CODE and USER CODE combination.

WibuKey Security

This triple code hierarchy defines the encryption and decryption and warrants a high flexibility in conjunction with a high degree of security. The *WibuKey* protection system is not based on a simple password check which is open forever once it is cracked. With *WibuKey* the encryption sequence is never the same.

1.6 Operating systems for *WibuKey*

WibuKey is a cross-platform copy protection system. It is available for a wide variety of operating systems and programming environments. This cross-platform functionality includes software and drivers for multiple operating systems and a common API (Application Programming Interface) for copy protection regardless of the operating system being utilized.

For individual protection, *WibuKey* has been designed with a portable API. The *WibuKey API* is independent of language, operating system and hardware used. This makes it possible to learn the system once, and then use it on any of the operating systems supported by *WibuKey* for truly cross-platform software protection.

Table 3 lists the operating systems supported by the various *WibuKey* form factors.

	Form Factor				
Operating system	**/P+ /RP+**	**/U+ /RU+**	**/M /P36**	**/ST**	**/SP**
Windows 98/Me/2000/XP/Vista/Windows 7/Server 2003, 2008 [32-bit/64-bit](IBM PC)	✓	✓	✓	✓	✓
Windows 98/Me/2000/XP/Vista/ Server 2003/Windows 7/Server 2003, 2008 [32-bit/64-bit] (Nec PC)	✗	✓	✓	✗	✗
Windows 95/NT4 (IBM PC)	✓	✗	✓	✓	✓
Windows 95/NT4 (Nec PC)	✗	✗	✓	✗	✗

	Form Factor				
Operating system	**/P+ /RP+**	**/U+ /RU+**	**/M /P36**	**/ST**	**/SP**
Mac OS 8 & 9	✗	✓	✗	✗	✗
Mac OS X	✗	✓	✗		✗
LINUX [32-bit/64-bit]	✓	✓	✗	✓	✗
UNIX-Derivate [32-bit/64-bit]	✗	✓[1]	✗	✓	✗

Table 3: *WibuBox* : Supported operating systems

The *WibuKey* drivers, the 32-bit version as well the 64-bit version, for Windows XP, Windows 2003. 2008 Server and Windows Vista, Windows 7 are certified by Microsoft.

Due to that the installation process is much easier, because no hardware wizard will pop-up.

The support for DOS, Windows 3.x, 95, OS/2 is no longer maintained, but older *WibuKey* versions can be used to protect and to use protected software.

The *WibuKey API* for explicit encryption of programs supports Windows 3.x/95/98/ NT/2000/XP,Vista, Windows 7, DOS, Mac OS/OS2, and UNIX/Linux.

The *WibuKey* software for automatic protection is currently available for Windows 3.x/95/98/NT/2000/XP, Vista, Windows 7 and DOS.

The *WibuKey* communication drivers are available for all of the operating systems listed above.

[1] *WibuKey* supports ARM/PPC in form of the current source driver. However, it is not available as download package ready to install but has to be compiled separately. In the NDA variant the source driver supports *WibuBox/U* and WKLAN. Please contact Wibu-Systems Support for more detailed information.

The *WibuKey* cross-platform features extend into the area of network protection as well. All *WibuBox* hardware operates the same. *WibuKey* uses a single, common API across all operating systems. Therefore the *WibuKey* network licensing features can be used in a heterogeneous network environment without requiring any extra work. This makes it possible to protect all software copies running on a network of Macs, PCs, and UNIX machines from a single *WibuBox*.

In addition, Wibu-Systems provides the ANSI C source code to the *WibuBox* communication driver. It is possible to adapt the *WibuKey* copy protection system to custom interfaces and operating environments while maintaining the strength of *WibuKey*'s common API design.

WibuKey's consistent design ensures "upwards compatibility" in the ever changing software world. Because of its consistent design, *WibuKey* can be expanded to support additional operating systems transparently to applications. For example, a DOS program protected with *WibuKey* in 1989 will run today in a DOS box on Windows NT, even though Windows NT did not exist in 1989. This design ensures that your investment in copy protection is safe for the future.

2 WibuKey features

After getting familiar with the *WibuKey* basics, this chapter explains some more advanced *WibuKey* features. These features extend the encryption possibilities offered by the FIRM CODE/USER CODE combination, and expand *WibuKey* from a standard copy protection system into a full-featured software management system.

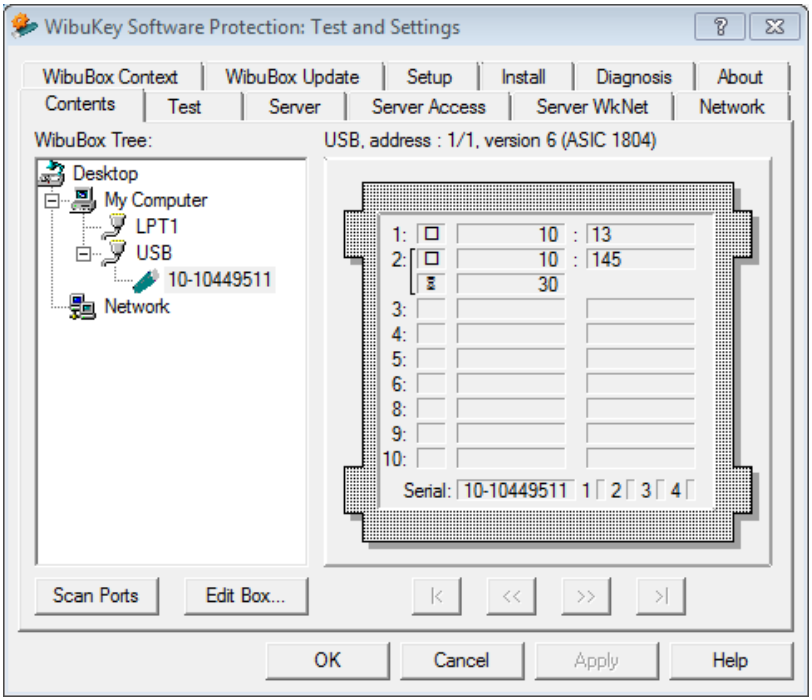


Figure 3: BASE ENTRIES with ADDED ENTRY (*WibuKey Control Panel Applet* Advanced Mode*)*

BASE ENTRY and ADDED ENTRY

There are a number of different options. The **BASE ENTRY** can be replaced or complemented by an **ADDED ENTRY**, which is bundled to the BASE ENTRY. An ADDED ENTRY can be bundled to any entry in the first 5 rows of the *WibuBox*. The ADDED ENTRY is physically stored in an entry with an index of 5 more than the corresponding BASE ENTRY. This means that if there is a bundled entry, it is always stored within rows 6 to 10 of a *WibuBox*. The visual effect in the *WibuKey* software is shown below.

The figure below shows the *WibuKey Control Panel Applet* with two BASE ENTRIES: The first is `10:13`, the second `10:145`. The second BASE ENTRY has an ADDED ENTRY (also called a bundled entry). It occupies the row 2+5 = 7. Entry 7 does not exist anymore as a free space for a BASE ENTRY because it is bundled to entry number 2.

The advanced features of *WibuKey* open up many licensing possibilities such as MASTER KEY functions, protection by module or program, software metering, LIMIT COUNTER, EXPIRATION DATE, pay-per-use, and storage of sensitive and

non-sensitive data within the *WibuBox*. In the following sections these features are explained in detail.

WibuBox/RP and *WibuBox/RU* only support a BASE ENTRY with one optional ADDED ENTRY.

2.1 Storing data in the *WibuBox*

USER DATA entries 🗁.

In some cases, data which does not need to be protected might need to be stored in a *WibuBox*. This can be user defined parameters, software settings, or any other data that is not used for program or data protection. In these cases, the user of the software with the *WibuBox* should be able change the data without needing any security information.

USER DATA ENTRIES provide this type of unprotected data storage in the *WibuBox*. They are indicated by the symbol of an open folder 🗁.

Each USER DATA entry can store up to 5 Bytes of data. The 5 bytes are broken into a lower value of 4 bytes and into an upper value of 1 byte.

A high value between `240` and `255` is reserved for *WibuKey* specific operations and should not used by software developers.

Data in a USER DATA entry are not protected. The security integrity of other entries in the same *WibuBox* is not reduced by the presence of a USER DATA entry.

To change USER DATA entries, use:

- the end user program **WKU** (see page 293),
- the *WibuKey Control Panel Applet* in the control panel (see page 272), or
- the program **WKLIST**.

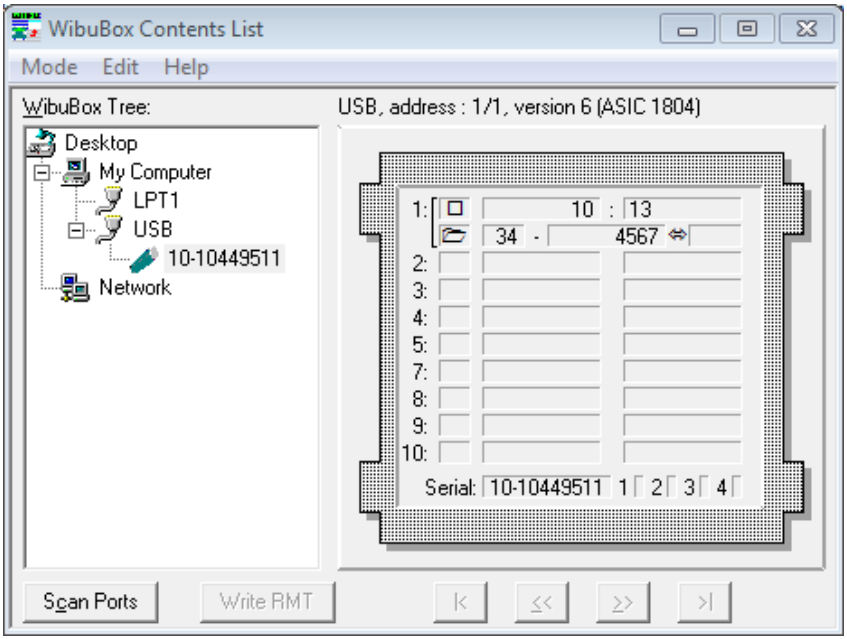


Figure 4: BASE ENTRY and USER DATA entry (`WKList`)

The figure shows one BASE ENTRY (`10:13`) and one USER DATA entry. The USER DATA entry has 34 for the high value and 4567 for the lower value. As shown, the USER DATA is an independent entry and has no effect on the BASE ENTRY in row 1.

2.2 Storing sensitive data

ADDED DATA

Often it is necessary to store sensitive data. This kind of data is not intended to initialize an encryption or decryption process, but nevertheless needs to be protected. **ADDED DATA** entries provide this type of protected data storage in the *WibuBox*. They are bundled entries and are indicated by the symbol of an open folder 🗁. ADDED DATA entries are connected to a BASE ENTRY. They can be used to store up to 5 bytes of data. The 5 bytes are broken into a lower value of 4 bytes and into an upper value of 1 byte.

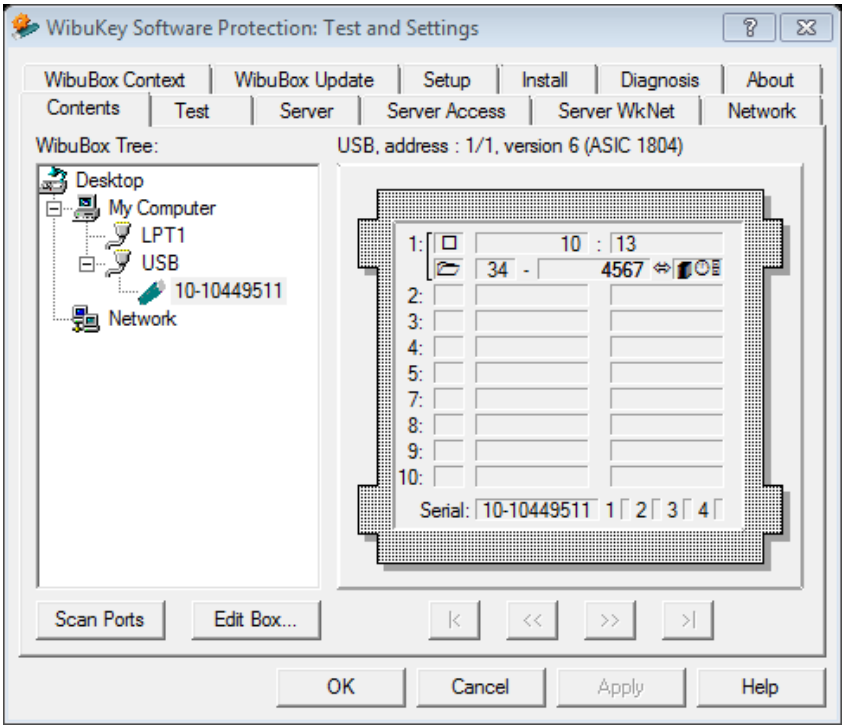


Figure 5: BASE ENTRY with ADDED DATA entry (*WibuKey Control Panel Applet* Advanced Mode)

Figure 5 shows one BASE ENTRY (`10:13`) and an ADDED USER entry which is bundled to the BASE ENTRY. This Added Entry occupies the entry 1+5 = 6 of the *WibuBox*. Visually the entry 6 is not available anymore. It is bundled to the first entry of the box.

ADDED DATA entries are by design not editable or changeable by the end user. The *WibuKey* end user tools such as the *Control Panel Applet* do not allow modification of ADDED DATA entries. Because they are bundled to a BASE ENTRY, in order to change the contents of an ADDED DATA entry, a programming sequence is needed which is based on the FIRM CODE and USER CODE of the bundled BASE ENTRY.

A high value between 240 and 255 is reserved for *WibuKey* specific operations and should not used by software developers.

In addition the programming sequence can optionally depend on

- the serial number of the *WibuBox* (identified by the 🗐 symbol),

- the internal programming counter of the *WibuBox* (identified by the ⏱ symbol), and
- the actual data contents (identified by the 🗄 symbol, requires *WibuBox* hardware version 4 or higher).

These options must be specified when an ADDED DATA entry is first created. The modification of the options cannot be made to an existing entry. If none of these additional options is set, the programming sequence for setting the ADDED DATA entry is identical for all *WibuBoxes* that contain the same BASE ENTRY. Setting these parameters offers additional security when changing the contents of an entry by *Remote Programming*.

The symbols are shown on the right side of the ADDED DATA entry in a separate field. The programming sequence in case of the last figure depends on the serial number of the *WibuBox*, the internal programming counter and the actual data contents.

Serial Number option (🗐)

If – as another example - the serial number option (🗐) is specified the programming sequence is fixed for a specific *WibuBox* because each *WibuBox* has a different serial number.

Counter option (⏱)

If the internal programming counter option (⏱) is specified, the programming sequence changes with each programming operation.

Data Contents option (🗄)

The data contents option (🗄) makes the programming sequence depend on the actual data to be protected.

2.3 Using the additional memory

The newest versions of the *WibuBox/P*, */U*, */RP* and */RU* provide 16 kByte of additional memory, which is divided into an 8 kByte unprotected and an 8 kByte protected area. The areas itself are separated in pages of 32 bytes each.

The reading and writing of the PROTECTED DATA depends on the FIRM / USER CODE combination in entry 1 of the *WibuBox*. It also it depends on the dependencies set in a possible ADDED DATA entry of this entry. The 8 byte signature for accessing the protected memory must be calculated by **`WKCYRPT`**.

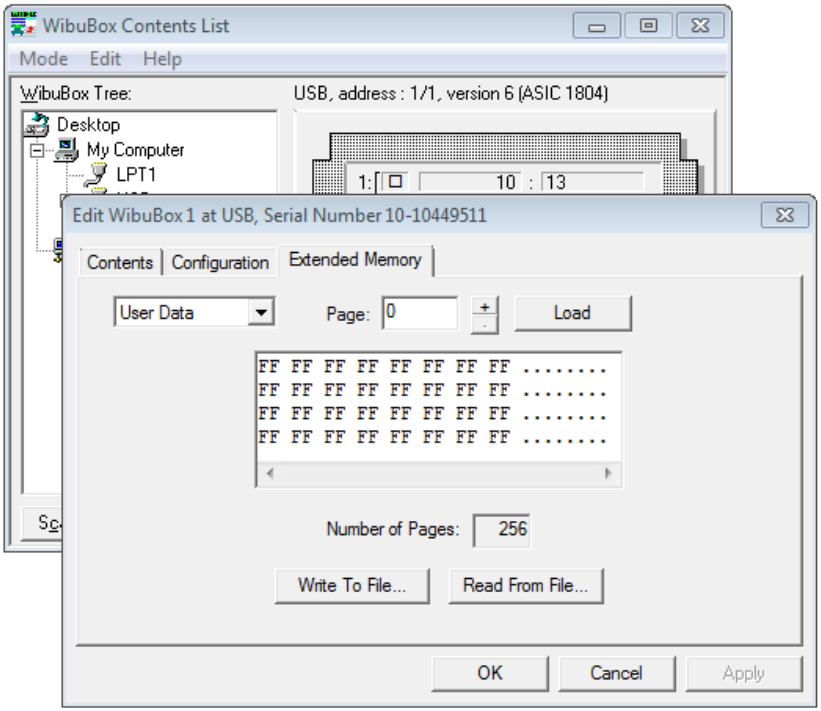


Figure 6: Unprotected USER DATA (`WKList`)

The additional memory e.g. can be used for storing application data during runtime (unprotected data), data that is used by the application to specify the software configuration, or to initialize algorithms. Also company or user specific information, e.g. address, can be stored.

2.4 Software metering

Software metering is a fair and profitable way of software distribution. It is a method of accounting for the use of software based on any number of factors such as time used, number of executions, number of printings, or even the number of times a certain button has been pressed. The customer pays only for the amount he really works with an application. This opens new marketing opportunities for all software developers.

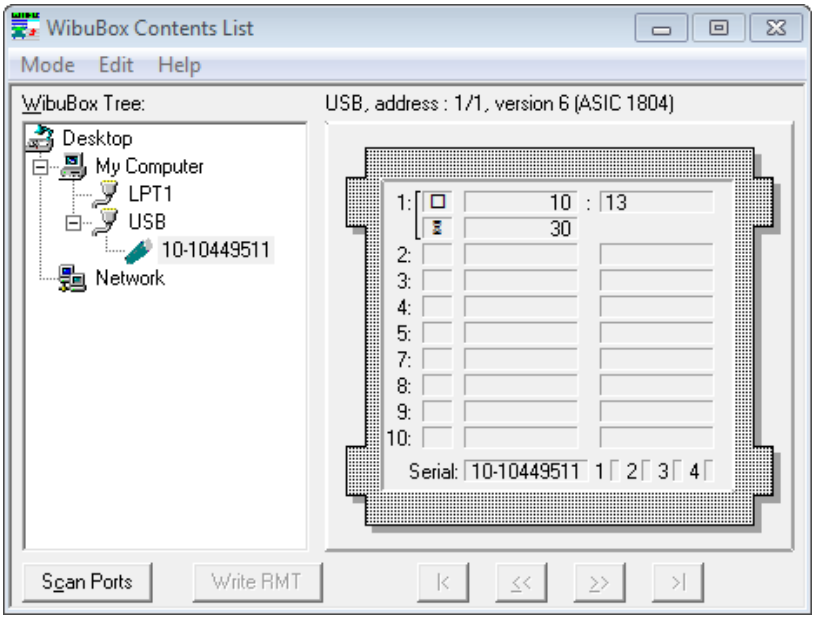


Figure 7: BASE ENTRY with bundled LIMIT COUNTER (`WKList`)

Limit Counter ⌛

With *WibuKey*, software metering can be realized by the implementation of a **LIMIT COUNTER**. This is a bundled entry and symbolized by an hour glass ⌛. A LIMIT COUNTER contains a counter value between `0` and `1,048,575` (20 Bits).

If the BASE ENTRY is selected for an encryption or decryption, the value of the LIMIT COUNTER is checked. If the value is `0`, the operation fails. This entry then cannot be used for any encryption or decryption operation. Otherwise the SELECTION CODE being used is analyzed: If the highest bit (bit `31`) is set, the LIMIT COUNTER is automatically reduced by 1 within the *WibuBox* ASIC chip. If bit `31` is set to `0`, no changes occur to the LIMIT COUNTER. This behavior allows you to use such an entry for encryption or decryption operations that reduce the LIMIT COUNTER, and also to use it for operations that do not reduce the LIMIT COUNTER. The SELECTION CODES for both variants differ by bit `31`. Because these different SELECTION CODES generate different encryption or decryption sequences, the LIMIT COUNTER mechanism cannot be simply removed by resetting bit `31` of the SELECTION CODE, as this would alter the necessary decryption and consequently make it unusable.

- *WibuKey* LIMIT COUNTERS provide a very safe way to implement software metering. Moreover, all of the reductions of the LIMIT COUNTER are done within the *WibuKey* ASIC in the *WibuBox*. There is no data transfer between the protected software and the program itself. Resetting the LIMIT

COUNTER is only possible by reprogramming the *WibuBox*. Therefore a FIRM CODE and USER CODE specific encryption sequence is needed. Furthermore, the encryption sequence differs with each *WibuBox* because it depends on the serial number of the *WibuBox*. It differs with each resetting of the LIMIT COUNTER. The internal programming counter is increased with each programming operation. The value of this programming counter modifies the necessary programming sequence. Saving the programming sequence of a specific LIMIT COUNTER reset operation to re-use the same programming sequence later for another change of the LIMIT COUNTER is not possible.

- The programming counter of a *WibuBox* version 3 or 4 is stored in 8 bits which limits the rate of a repeating programming sequence to 256 programming operations of a single *WibuBox*. Because this number is too less for a continued updating of a LIMIT COUNTER with a fixed FIRM CODE/USER CODE in the same *WibuBox*, each LIMIT COUNTER entry contains its own local programming counter which is updated by each modification of the LIMIT COUNTER. To modify a LIMIT COUNTER value, also the local programming counter influences the calculation of the required sequence. Moreover each modification increases only this local programming counter until its 1-byte-value overruns after 256 modifications. Only in this case the global programming counter of the *WibuBox* is increased by 1 and the local counter is set to 0 again. This bundling of the two counters results in a repeat rate of 65,536 for the resetting of a LIMIT COUNTER value to a specific value.

WibuKey does provide a secure *Remote Programming* facility that allows the software developer to securely reprogram a *WibuBox* at the end user's site. It is possible to reset a LIMIT COUNTER without physically exchanging the *WibuBox*. An order can be made over the phone. Then the necessary information to reset the LIMIT COUNTER is provided. The customer is instantly up and running with his newly purchased software. *WibuKey Remote Programming* is explained in detail on pages 48 and 226.

2.5 EXPIRATION DATE

Another way to distribute software by time is to set an **EXPIRATION DATE**. An EXPIRATION DATE is an absolute date, for example January 8th, 2004.

EXPIRATION DATE ⌦

The symbol for an EXPIRATION DATE in the *WibuKey* software is a cross in an arrow ⌦. EXPIRATION DATES allow the entry to be used for encryption and decryption only until this date.

The date will only be checked if bit 29 of the SELECTION CODE is set. In comparison to the LIMIT COUNTER, the advantage of the EXPIRATION DATE is the absolute time limit. A disadvantage is that the time span depends on the computer system date which can be modified by the user. Therefore it is

recommended to always use an EXPIRATION DATE together with a LIMIT COUNTER or an ADDED DATA entry containing the system date from the last execution. These are stored inside the *WibuBox* and cannot be changed by the end user. Nevertheless, changing a system's date can cause big problems especially in a network. Therefore the security provided by the EXPIRATION DATE alone is sufficient for most applications. EXPIRATION DATES are stored as bundled entries. The *WibuKey* drivers, starting with version 2.30, recognize EXPIRATION DATES.

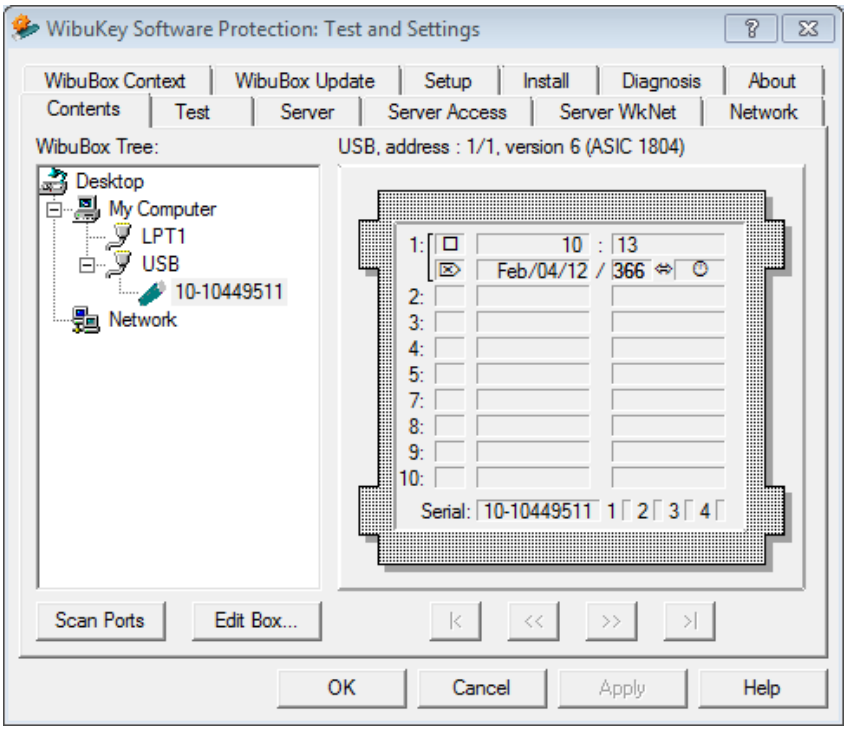


Figure 8: BASE ENTRY with bundled EXPIRATION DATE (*WibuKey Control Panel Applet* Advanced Mode*)*

The *WibuKey* software will display an EXPIRATION DATE the following way:

WWYYMMMDD/nnn	
WW	for the day of the week, i.e. Su, Mo, Tu for Sunday, Monday, Tuesday, etc.
YY	for the year in the range from 70...99 for 1970 until 1999 and 00...69 for the range from 2000 to 2069.

	WWYYMMMDD/nnn
MMM	for the month, i.e. Jan, Feb, Mar, etc
DD	for the day with the month in the range from 01...31.
nnn	for the number of days between now and the EXPIRATION DATE: 0 for today, 1 for tomorrow, 2 for in two days, etc. If the EXPIRATION DATE is in the past the text "---" will be displayed instead of a negative number.

Graphical Windows applications like the *WibuKey Control Panel Applet* or **WKList** show the EXPIRATION DATE in a similar way. The date might be arranged slightly differently depending on the current country settings in Windows. If the EXPIRATION DATE is in the same year as the current year, the day of the week (WW) is shown instead of the year (YY).

The *WibuBox* entry can be used until the day that the EXPIRATION DATE specifies. On Windows NT the date is compared to the UTC (Greenwich) time. In this case there might be differences of a few hours.

2.6 Protecting modular software

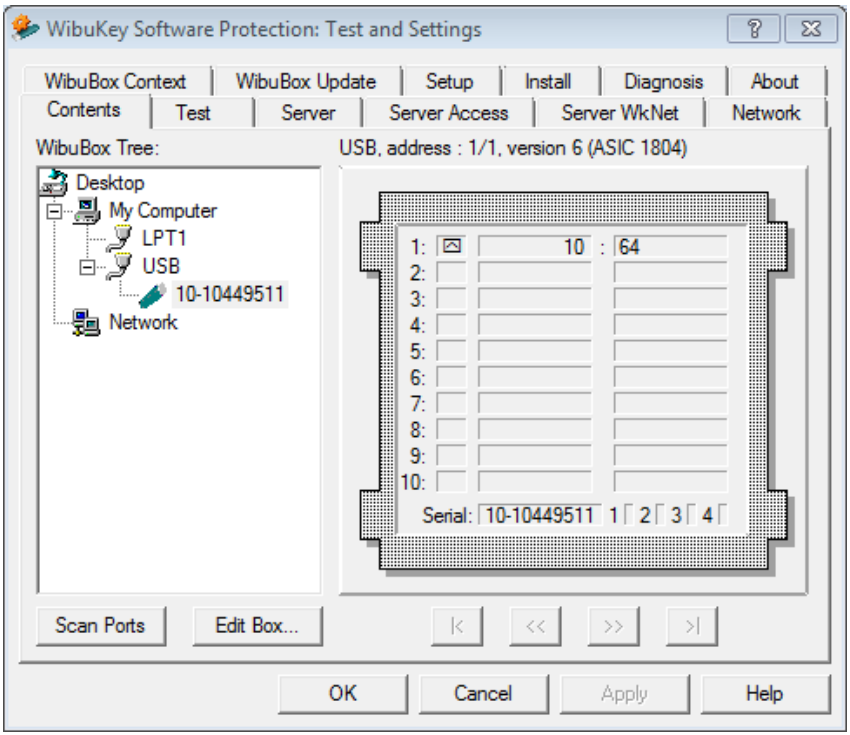


Figure 9: MASTER ENTRY (*WibuKey Control Panel Applet* Advanced Mode*)*

Complex software is often realized as a large application divided into different modules. These modules can be sold separately to offer user customization. This distribution method requires separate protection of different modules. With *WibuKey* it is possible to give each customer the whole software package and control access to each module individually. Depending on his demands some modules will be open for access and others will stay closed. Configuring software in this way saves work, time and money for the software developer. In addition, when a customer wants to purchase a new module, the *WibuBox* can be updated very easily and securely by *Remote Programming*, giving the user instant gratification for his purchase.

A normal FIRM CODE/USER CODE entry can generate an encryption or decryption sequence for one specific USER CODE and therefore protect one application or one module.

MASTER ENTRIES

WibuKey **MASTER ENTRIES** can simulate a class of several USER CODES within one *WibuBox* entry. With MASTER ENTRIES the protection of up to 180 different software modules is possible with one single *WibuBox*. A MASTER ENTRY consists of a FIRM CODE and a **MASTER CODE**. MASTER ENTRIES are a different kind of BASE ENTRY. They are not bundled entries, but may have bundled entries like LIMIT COUNTER, EXPIRATION DATE, or ADDED DATA. The roof symbol ⌂ is used to indicate a MASTER ENTRY.

The principle of MASTER ENTRIES is much like the concept of a master key in an apartment building. Individual USER CODES are used to protect each module and they are compared against the MASTER ENTRY to enable the protection. Each USER CODE that remains unchanged after a bit-by-bit *and*-combination with the MASTER CODE can generate an encryption or decryption sequence.

For example, if a module is protected with a USER CODE 4 (binary = `0100`) and a *WibuBox* is delivered with the MASTER CODE 5 (`0101`), the bit-by-bit *and*-combination results in `0100`. The USER CODE of the module remained unchanged as a result of the *and*-combination. The *WibuBox* enables access to that particular module. If another module is protected with a USER CODE 2 (`0010`), the result of the combination is `0000`. In this case the result is unequal to the USER CODE of the protected module. The module then cannot be opened (see table below).

USER CODE of protected Module	`0100`	`0010`
MASTER USER CODE in *WibuBox*	`0101`	`0101`
Result of bit-by-bit *and*-combination	`0100`	`0000`
Use of protected module	possible	impossible

By using different USER CODES for each module, it is possible to grant a specific user access to the application on a module-by-module basis. Providing the user with a *WibuBox* holding the appropriate MASTER ENTRY allows access to the desired modules. Another advantage is that the end user can purchase additional modules at any time. All there is to do to deliver the new modules is to update his *WibuBox* via the *WibuKey* Remote Programming facility. The new MASTER ENTRY includes access to the newly purchased modules.

The *WibuBox/R* does not support MASTER ENTRIES.

3 *Network support* – WibuKey Server

This chapter explains the *WibuKey* network concept. *WibuKey* includes complete network support built into all standard *WibuBox* hardware, except *WibuBox/R* which only supports a maximum of one user. With *WibuKey* it is not necessary to purchase different network keys or additional license management

software. With *WibuKey* it is possible to easily control the number of software copies that can be used at the same time in a network.

3.1 The *WibuKey* network concept

A *WibuBox* does not need to be attached to the local computer where the protected software is started. The software protection in a network is handled centrally by a *WibuBox* on a computer called the *WibuKey Server*. This means that in a network, one single, standard *WibuBox* can protect an application that runs on multiple computers simultaneously. This even works in heterogeneous networks consisting of PCs and Macintosh computers. The *WibuBox* is the same as the one used for applications that run at a single workstation.

For software protection in a network a server process is installed on the network computer to which the *WibuBox* is attached. *WibuKey* offers two different methods to protect an application in the network called ***WkNet*** and ***WkLAN***. To get more information on the characteristics see page 235.

Please note that this guide will not deal in detail with ***WkNet*** although this method is still supported by *WibuKey* and mentioned sporadically in the document.

However, for new implementing *WibuKey* using this older method is not recommended.

3.2 Network license management

Besides the actual protection of the software in a network, it is possible to define the number of software licenses a customer can use in his network at the same time. This feature is called **network license management**. *WibuKey* offers its customers a high degree of flexibility in the realization of network license management.

3.2.1 Licensing a single application

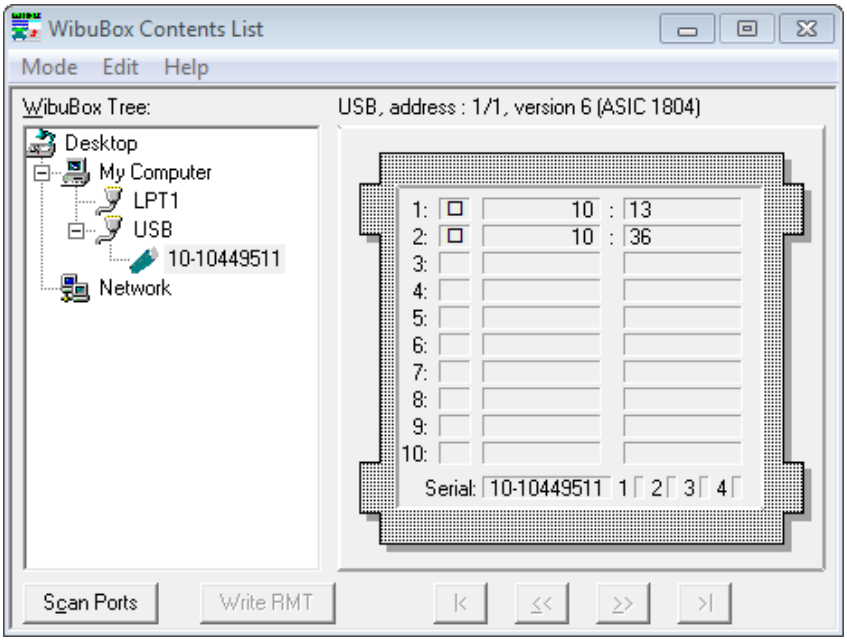


Figure 10: *WibuBox* showing a 23-user network license (*WKList*)

The *WibuBox* on the *WibuKey Server* contains a *WibuKey* network entry. This entry consists of a FIRM CODE/USER CODE. It is used for the encryption and decryption of the protected software. If a *WibuBox* contains only one entry the software can be used once on any network computer. The USER QUANTITY is the number of software copies that can run simultaneously in a network. It is implemented by a second network entry in the *WibuBox*.

The FIRM CODE must be the same for both entries. The USER CODE of the second entry must have a higher value than the USER CODE of the first entry. The difference between the USER CODES defines the number of software copies that can be used at the same time in a network.

If there are many entries with the same FIRM Code in the *WibuBox*, the smallest value that happens to be larger than the first USER CODE is employed.

USER QUANTITY calculation

Protected Application: FIRM CODE 10 - USER CODE 13

WibuBox

Entry 1: 10:13

Entry 2: 10:36

WibuKey calculates the difference between the two USER CODES to determine the USER QUANTITY (36 – 13 = 23). In the network 22 copies of the application encrypted with FIRM CODE 10, USER CODE 13 can run simultaneously.

3.2.2 Licensing modular software

In the case of modular software the first network entry is a FIRM CODE/MASTER CODE. All applications or modules that fit to the MASTER CODE can run in the network. The applications or modules that fit to the MASTER CODE share the USER QUANTITY. It is defined by the network count entry.

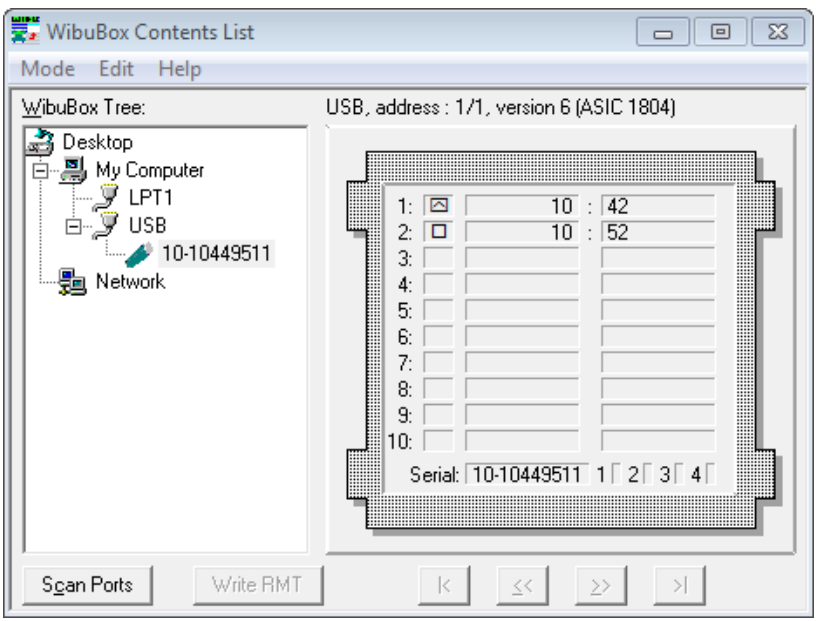


Figure 11: *WibuBox* showing a 10-user license for modular software (WKList)

This entry cannot contain a MASTER CODE. As soon as one application is running, the number of software copies which can be started is reduced by one for all other applications and modules.

Licensing modular software

Protected Basic Application: FIRM CODE: `10` - USER CODE: 2

Protected Module A: FIRM CODE: `10` - USER CODE: 8

Protected Module B: FIRM CODE: `10` - USER CODE: 32

WibuBox:

Entry 1: M`10:42`

Entry 2: `10:52`.

The network USER QUANTITY is 52 – 42 = 10.

10 copies of the basic application, module A or B can run at the same time. If a client starts the basic application and module A, it is possible to start 8 more copies. How many of each is not defined as long as no more than 10 copies run.

A FIRM CODE/MASTER CODE entry can be used as **OVERFLOW CLUSTER.** This is a way to give a customer an additional number of licenses. He can use these licenses for all those applications or modules that are compatible to the specified MASTER CODE. This demonstrates the high flexibility *WibuKey* offers even end users of protected software.

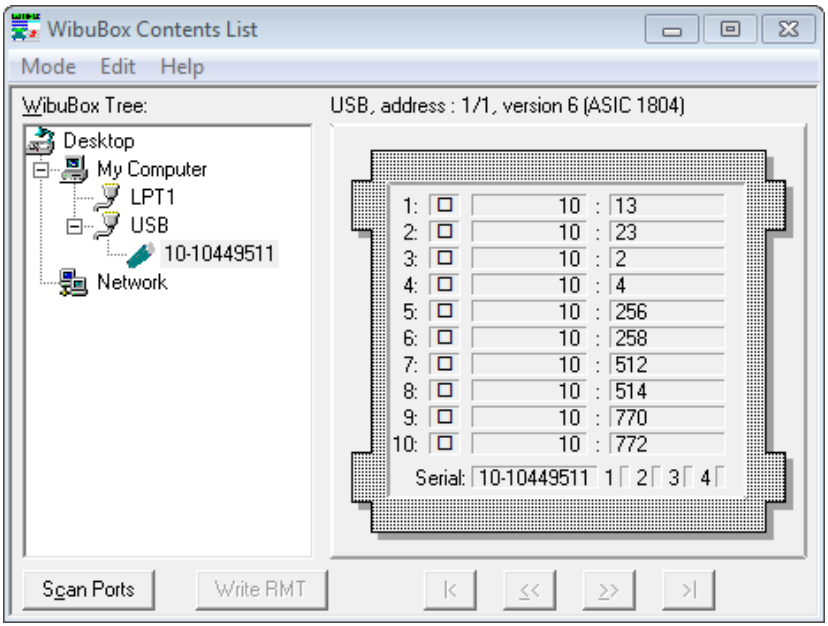


Figure 12: *WibuBox* showing an overflow entry for modular software (`WKList`)

Overflow Entry

Protected Basic Application: FIRM CODE `10` - USER CODE `13`

Protected Module A: FIRM CODE `10` – USER CODE `2`

Protected Module B: FIRM CODE `10` – USER CODE `256`

Protected Module C: FIRM CODE `10` – USER CODE `512`

OVERFLOW CLUSTER: FIRM CODE `10` – MASTER CODE `770`

WibuBox:

Entry 1: `10:13`

Entry 2: `10:23` (10 copies)

Entry 3: `10:2`

Entry 4: `10:4` (2 copies)

Entry 5: `10:256`

Overflow Entry

Entry 6: `10:258` (2 copies)

Entry 7: `10:512`

Entry 8: `10:514` (2 copies)

Entry 9: `M10:770`

Entry 10: `10:772` (overflow cluster: 2 overflow licenses).

On the network it is possible to run 10 copies of the basic application, 2 copies of module A, 2 copies of module B, 2 copies of module C and 2 additional copies of module A, B or C. The additional copies are the result of the OVERFLOW CLUSTER.

3.2.3 Huge license management

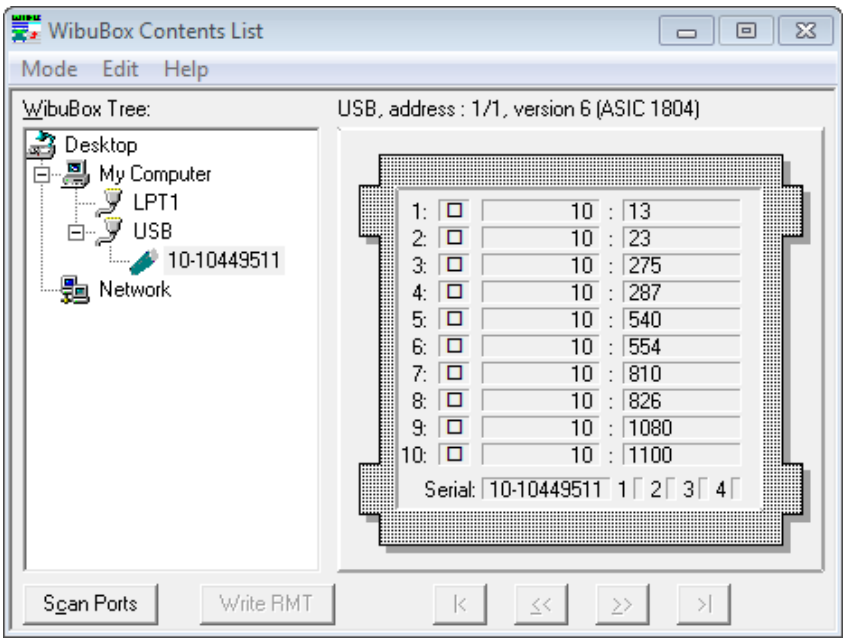


Figure 13: *WibuBox* showing showing a `10`, `12`, `14`, `16` and `20` user license (`WKList`)

One *WibuBox* offers slots for 10 entries. This leads to at most 5 network entries with different USER QUANTITIES. The entries need to differ by at least 251 to avoid the wrong entry being taken to calculate the licenses.

With *Huge License Management* there is no restriction to the number of different USER QUANTITIES for a large number of different applications and modules.

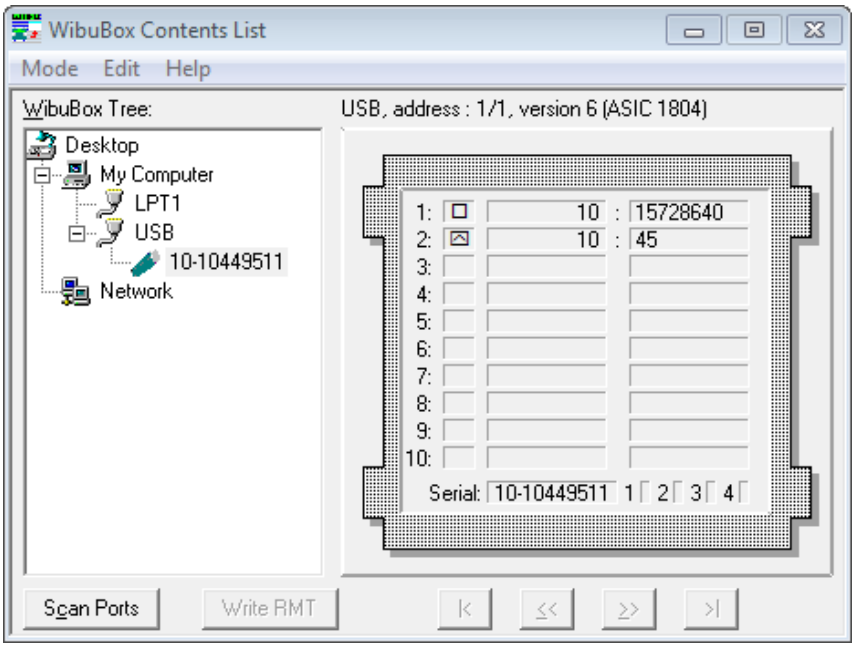


Figure 14: *WibuBox* showing entries for Huge License Management (*WKList*)

Example: 5 different User Quantities in one *WibuBox*

Protected Application A: FIRM CODE `10` – USER CODE `13`

Protected Application B: FIRM CODE `10` – USER CODE `270`

Protected Application C: FIRM CODE `10` – USER CODE `530`

Protected Application D: FIRM CODE `10` – USER CODE `790`

Protected Application E: FIRM CODE `10` – USER CODE `1050`

WibuBox:

Entry 1: `10:13`

Entry 2: `10:23` (10 copies)

Example: 5 different User Quantities in one *WibuBox*

Entry 3: `10:275`

Entry 4: `10:287` (12copies)

Entry 5: `10:540`

Entry 6: `10:554` (14 copies)

Entry 7: `10:810`

Entry 8: `10:826` (16 copies)

Entry 9: `10:1080`

Entry 10: `10:1100` (20 copies)

Five different applications are protected with individual network entries. The USER QUANTITY allows each application an individual number of simultaneous copies to run in the network.

To assign more than 5 different user limits to a large number of applications or modules it is necessary to use a special file.

This file is called **Huge License Management (HLM) Control File**. The software developer generates this file via **WKCRYPT** (see page 261). It contains all network entries and the corresponding number of simultaneously running copies in the network.

The HLM file itself is protected by a *WibuBox* entry. In the *WibuBox* two entries are necessary. The first entry with a FIRM CODE/USER CODE protects the HLM file. A special range of USER CODES is reserved for the protection of HLM files. The second entry with a FIRM CODE/MASTER CODE defines which applications and modules listed in the HLM file can be used. Only those applications and modules that are compatible to the specified MASTER CODE can run in the network.

With HLM there is no restriction to the number of different USER QUANTITIES for a large number of different applications and modules.

The HLM file can also be stored in the EXTENDED MEMORY of the /+ variants of the *WibuBox*.

3.3 Advantages of *WibuKey* network license management

Before you implement a network license management system you should precisely plan the assignment of the different USER CODES to the different license variants. This way you get all the advantages of *WibuKey* now and in the future.

- The definition in the *WibuBox* of the maximum number of simultaneous usage licenses within a network can be easily changed at a later date with the *WibuKey* remote programming feature. This allows for an extremely simple method for selling and installing additional licenses to customers.
- The software developer is in control of the network license limits. He can set the limit to be the exact number that he needs. For example, a 7-user network license is easily set with *WibuKey*. There are no requirements to use pre-defined, pre-set license limits, nor are there any costs to update the license limits in the field.
- With *WibuKey* there is no difference between a key for a single workstation and a key for a network. This means that any *WibuBox* can be used on a single workstation as well as in a network. You can change the settings in a *WibuBox* easily by remote programming.

4 *WibuKey* hardware in the field

WibuKey enables you to get the most out of your investment in copy protection hardware by offering a number of features that allow you to reuse the same *WibuBox* while it is in the field. No key swapping or shipping is necessary. Here are a few examples of the enormous benefits of this feature:

- Protect software updates or additional program modules with the same *WibuBox* that was delivered with the original software order.
- Protect multiple applications within a single *WibuBox*, even if the applications are from different software developers.
- Make a demo version by using an EXPIRATION DATE in the *WibuBox*. With *WibuKey* it is possible to extend the trial period for a customer very easily as well as upgrade the trial to a full version.
- Upgrade a single user installation to a network license manager, or upgrade a network installation to add more concurrent users when you sell additional licenses of your software.
- Add additionally required licenses for existing clients in their *WibuBox*.

4.1 Delivering customer-specific updates

This is the easiest method of delivering updates. It does require a special kind of software distribution in which each application is delivered with a specific FIRM CODE - USER CODE entry for each customer. It is important that no two customers have the same USER CODE programmed in their *WibuBox*. If a customer orders an update of the application, it is encrypted with the FIRM CODE - USER CODE of the *WibuBox* the customer already possesses. In this way a *WibuBox* can be reused as often as software updates are delivered.

In the case of a huge number of clients and of standard software products Wibu-Systems recommends using product specific entries, see page 52.

4.2 *WibuBox* remote programming

WibuKey makes it possible to reprogram entries in a *WibuBox* while the box is located at the customer's site. This option is called ***remote programming***. Remote programming is accomplished by exchanging encrypted data files. These data files can be exchanged by any medium such as e-mail, files on floppy disks, etc. Because the data is readable, it is possible for the necessary remote programming data to be sent via a letter, fax or even by phone. *WibuKey* offers two remote programming tools for the end user,

- the interactive ***Control Panel Applet***,
- the commandline tool ***WKU/WKU 32*** or
- easily by clicking in the Windows Explorer with the right mouse button.

and one additional tool for software developers:

- the commandline mode of **WKCRYPT**

which generate and read the remote programming data files. To simplify the analysis of these data files, each remote programming data file contains a header. Each line, which starts with a semicolon, is treated as a comment.

A remote programming data file might look like this

```
[WIBU-SYSTEMS Control File]
Guid={00070000-0000-1100-8002-0000C06B5161}
Specification=WibuKey Remote Programming Context File
Version=1.00
[Contents]
; 1 entry of WibuBox with Serial Number 4-803057:
X9NNw NXZFh NV9By V4Ce8 ZX4e0 18CN5
eyZ44 K
```

The first two lines of the example contain header information.

The following lines contain the encrypted entries of a specific *WibuBox*.

The native remote programming data is a byte sequence that is packed by an enhanced Lempel-Ziv algorithm and converted into a readable alphanumeric sequence. They contain text sequences with characters from `A` to `Z` except `O`, `D`, `I` or `J` and numerals from `0` to `9`. Lower and upper cases are not interpreted differently. The data is made up of the state information about the specific *WibuBox* as well as added management information. This information includes

the *WibuBox* serial number, an internal programming counter and a 16-bit CCITT-CRC value to detect errors in the remote programming data.

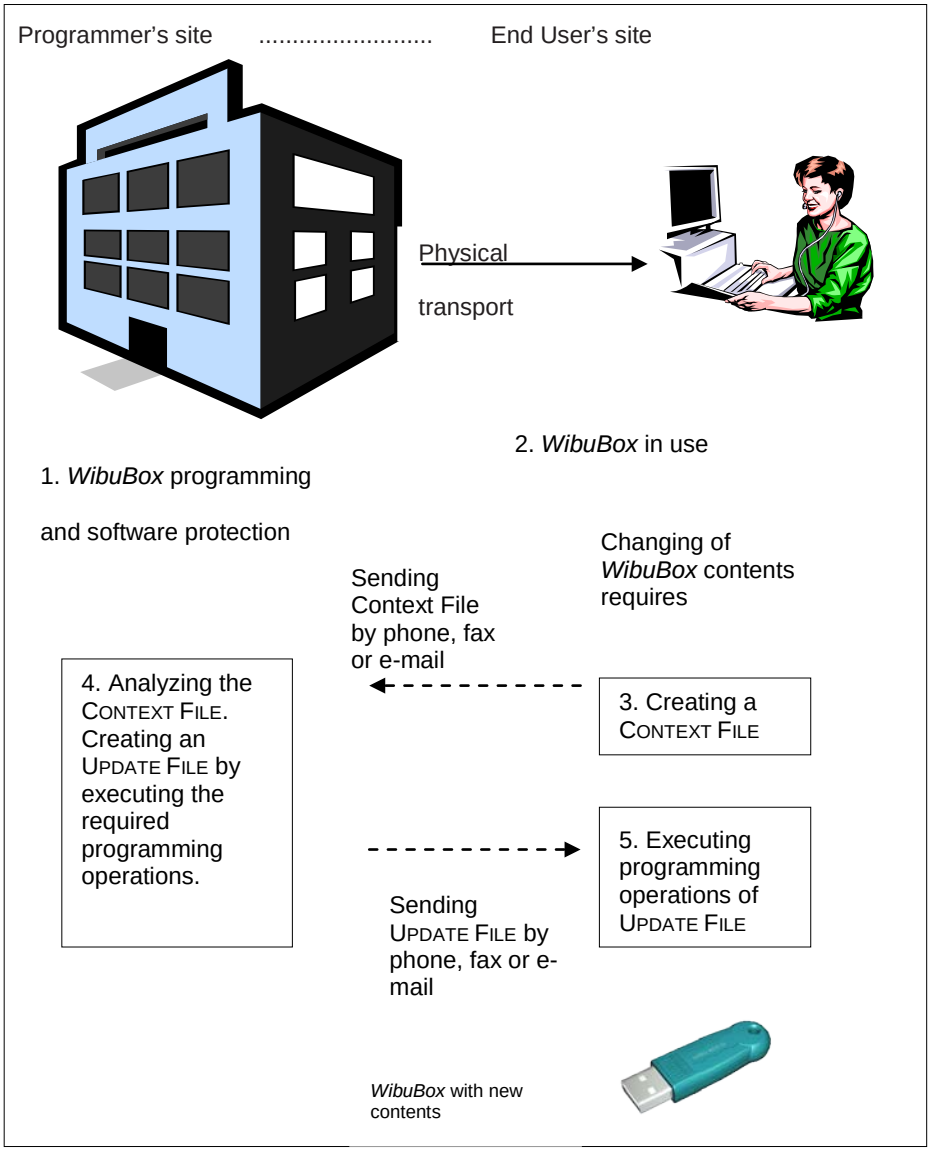


Figure 15:*WibuKey* remote programming concept

In the beginning a *WibuBox* is at the developer's site. The software developer delivers a programmed *WibuBox* along with the software.

The customer gets the *WibuBox* with the software s/he ordered and uses it to legally access the software. At some point in the future, an event such as an EXPIRATION DATE is reached, a LIMIT COUNTER reaches a value of `0` or the software developer offers an update or additional modules or applications. At this point the customer is interested in updating the software.

The customer generates a **CONTEXT FILE** of the *WibuBox*. This is done interactively in the *WibuKey Control Panel Applet* or with the **WKU/WKU 32** commandline tool, or by clicking the right mouse button in the Windows Explorer. This CONTEXT FILE contains all information about the entries of the *WibuBox*. It is necessary to change these contents via the *WibuKey* remote programming process. The customer sends this file to the developer by fax, disk or e-mail. Even dictating the characters by phone is possible and absolutely secure because nobody other than the software developer with the correct FIRM SECURITY BOX (FSB) and `WKFIRM.WBC` file can create the correct UPDATE FILE to this CONTEXT FILE.

The programmer can then change the entries in the *WibuBox* as desired or on customer request. The programmer uses the *WibuKey* software **`WKCRYPT`** and references the provided CONTEXT File as the *WibuBox* to program. If there are entries in the box with different FIRM CODES, the programmer can only change the entries with the FIRM CODE for which he has the corresponding FSB and `WKFIRM.WBC` file. This makes the *WibuKey* remote programming an absolutely secure operation. The new file is called the **UPDATE FILE**.

It contains the appropriate management information for the specific *WibuBox*, as well as the commands that implement the changes specified by the software developer. This file is then sent back to the customer.

The customer double-clicks the UPDATE FILE or reads it into his *WibuBox* interactively with the *Control Panel Applet* or with the **`WKU/WKU 32`** commandline tool. Only the customer with a matching *WibuBox* can use the UPDATE FILE because of the management information embedded into the update sequence. The *WibuBox* now contains the new entries. It is ready for use.

It is even possible to shorten this procedure by eliminating step 3. The software developer can save a CONTEXT FILE before the *WibuBox* is sent physically to the customer. Because the CONTEXT FILES are 'on file' at the developer's site, the customer does not have to send a CONTEXT FILE to the programmer. The programmer saves a **Modified Context File** for the reprogrammed *WibuBox* each time he changes entries. It is best to keep at last the latest two CONTEXT FILES on file.

5 *Checklist for copy protection*

This chapter is designed to help software developers define a good concept for the requirements of a copy protection system. As with any new system or process, it is a good idea to make some decisions about what the needs of the system are. This helps to avoid surprises and sets the stage for success. The information in this chapter provides a checklist to help identify the copy protection requirements for each software application.

5.1 Degree of protection

The first question that needs to be addressed is the degree of protection that is required. *WibuKey* specializes in providing the highest quality and highest reliability copy protection available. The true encryption method of copy protection that *WibuKey* utilizes is the most secure type of copy protection that you can buy. In addition, Wibu-Systems has an ISO 9001 certification. This quality assurance system guarantees quality workmanship through every aspect of the product's life from R&D through production, marketing, sales and support.

5.2 Automatic or individual protection

The next question regarding copy protection requirements is whether the automatic encryption or the individual protection process is most suitable for the protection of the program or data file at hand.

Automatic protection works without requiring any changes to the source code of the software. Most of the *WibuKey* features can be used with automatic encryption. This is the fastest way to protect a program or data file and is realized within minutes.

Individual protection requires changes to the source code of the software. The integration of calls to the *WibuKey API* into the source code is necessary.

The automatic protection is necessary if

- the software developer wants to access the *WibuBox* at any part of the source code in his program.
- data is protected inside the program.
- the same program is distributed as demo version and full version or as modular software.
- the **WkNet** network protection is used.

5.3 Product or customer-specific protection

The entries in a *WibuBox* can be used in different ways:

Product specific entries encrypt each application with a specific USER CODE entry. This means that one application is always encrypted with the same entry and two different applications are always encrypted with different entries. Product specific encryption is recommended for mid- to high quantities of distributed programs. It is independent from customers and supports the large quantity duplication of programs for CD-ROM or Internet-based distribution.

Customer specific entries encrypt a product for a specific customer. Each customer gets a personalized *WibuBox* with a customer-specific USER CODE. One application is protected with different entries depending on the customer. Two different applications can be encrypted with the same entry as long as they are licensed for the same customer. The advantage is that each customer gets a personalized *WibuBox* and needs basically just one entry. All licensed programs are delivered with the customer's USER CODE. This makes sense for smaller quantities of high priced software. Keeping track of each customer is necessary.

A combination of both methods. The encryption of the software is handled on a product specific basis. In addition, a customer specific entry is programmed into the *WibuBox*. With this entry a customer specific `*.INI` file or customer specific data can be protected. Consider a software package for use in a doctor's office. The program is protected by product specific entries in the *WibuBox* which allow for simple duplication. Patient data are protected by the doctor's specific customer entry in the *WibuBox*. With this approach, all programs and modules can be replicated on CD-ROM. Nevertheless, the proprietary data of each doctor cannot be decrypted and used by an unauthorized person.

5.4 Protection with a local *WibuBox* and/or across the network

With the increasing use of networked computing, it is recommended to plan any copy protection system so that it can be used at a single workstation as well as in a properly licensed manor across a network. *WibuKey* supports this perfectly with its universal network solutions which can even be used in heterogeneous networks.

The advantages of a network implementation are the following:

- The protected program can use a locally connected *WibuBox* as well as a *WibuBox* in the network.
- For network licensing, the *WibuBox* can be connected to any workstation or to a Novell, Windows or Macintosh server.
- A license for a single workstation can be used immediately as a single floating license in the network with the *WibuKey Server*. This means that a customer can work with the protected program on any single computer on the network.

- An upgrade for multiple network user licenses is possible without changes to the program. Only the *WibuBox* entry that defines the number of simultaneous users needs to be changed.

The USER CODES which are used for applications or modules must differ by a minimum of 256 at least to be used with individual numbers of network licenses per application or module. For example, application App1 is protected by USER CODE 100, App2 should be protected by USER CODE 356 or higher.

For automatic encryption, the only thing that needs to be done to implement the *WibuKey* network licensing is to set the "**WkLAN**" option in the **WKCRYPT** program.
For individual protection, the function **WkbAccess2()** is implemented in the source code in order to get a handle to the network system.

If the use at a single workstation only is desired without floating license capabilities, a *WibuBox* is connected locally to the computer on which the software is to run. For automatic protection, "**WkLAN**" and "**WkNet**" options are not selected. This excludes the floating license features from the encryption. For individual protection, the implementation does not use the **WkbAccess2()** function and the **HWKB_Local** option is specified in the call to **WkbOpen2()**.

5.5 Protection of modular software with a single *WibuBox*

Secure sales and distribution of different programs or modules requires them to be individually protected. However, from the customer's perspective, a single hardware key should be sufficient. With *WibuKey*, up to 190 modules or programs can be protected with one *WibuBox* with completely independent encryption for each module or program.

If each program or module is protected with an individual USER CODE, this would allow for 10 programs which can be protected with the 10 entries of one *WibuBox*. The special BASE ENTRY - called a **MASTER ENTRY** – has a similar function as a master key in a big building. The USER CODE is used as a bit mask instead of a single value. Each USER CODE that stays the same after a logical bit-by-bit *and*-combination with the MASTER ENTRY bit mask can generate an encryption or decryption sequence.

To enable or disable specific modules or programs it is necessary that the USER CODES for the protection of the program be a two's complement, i.e. 1, 2, 4, 8,.... From the outset – even if there are initially only a few programs or modules – this must be considered and documented in order to use MASTER ENTRIES. If a program later consists of a large quantity of modules, it is no problem to protect those with one *WibuBox*.

With automatic encryption, the different modules or programs need to be independent from each other. Each program or module is protected separately

and the documentation contains the USER CODE for each of this programs or modules.

Explicit encryption offers the option to encrypt different functions of one program with different USER CODES. In this way access to specific features within a program or module can be controlled individually.

5.6 Simulation of multiple USER CODES with MASTER ENTRIES

A standard FIRM CODE/USER CODE entry generates encryption and decryption sequences for one specified USER CODE. The SELECTION CODE can be used to vary this sequence. A variation of the USER CODE itself is not possible.

MASTER ENTRIES, displayed by ⊠, can simulate a class of USER CODES: Each USER CODE, which remains unchanged after a bit wise AND operation with the master USER CODE value can be simulated with that master entry.

A MASTER ENTRY with the mask 11 (11 = 1011 binary) simulates the USER CODES 1 (0001), 2 (0010), 3 (0011), 8 (1000), 9 (1001), 10 (1010) and 11 (1011), but not 4 (0100) or 12 (1100).

MASTER ENTRIES may be also combined with a LIMIT COUNTER or ADDED DATA like normal FIRM CODE/USER CODE entries. MASTER ENTRIES are very useful to simulate a large class of *WibuBoxes* by one master *WibuBox* for service purposes or for the safe protection of up to 190 different modules of software by different encryption/decryption variants with one single *WibuBox*. The following principle is used:

Each module is protected by an own USER CODE. Each USER CODE is a power of 2 (value 1,2,4,8,...) till the maximum value of 2^{19} (524288). Using this technique, a MASTER ENTRY can protect up to 20 different modules through a bit wise combination.

A MASTER ENTRY with the value of 5 protects the module with USER CODE 1 and the module with USER CODE 4, but not the modules with USER CODE 2 or 8. This is because 2 or 8 is ANDed with 5 not 2 or 8.

To protect more than 20 different modules with one *WibuBox*, more than one entry is needed. It is not sufficient, to use the first n bits of an entry to implement a table.

In a four bit coding of lines in a table, an entry in line 3 (0011) would also enable line 1 (0001) and line 2 (0010)! With MASTER ENTRIES the bits, which are set, are important if an entry could be found or not. It must be assured, that master or USER CODE entries are distinguished by at least one set bit. One solution would be the coding according to the following table:

Entry	Coding for up to 190 modules	Coding for up to 120 modules	Coding for up to 63 modules
1	00011	0011	011
2	00101	0101	101
3	00110	0110	110
4	01001	1001	
5	01010	1010	
6	01100	1100	
7	10001		
8	10010		
9	10100		
10	11000		

Table 4: Table with different possibilities of encoding

For a protection of up to 63 modules, the following table would be used:

		Coding			Table columns				
	Entry	23	22	21	20	19	…	1	0
Table lines	1	0	1	1	X	…	…		X
	2	1	0	1		…	…		
	3	1	1	0	X	…	…		X

Table 5: Table for protection of up to 63 different modules

The first entry in the *WibuBox* contains the first line of the table above. Bit 23,22 and 21 are used to encode the line (011 in this case), bit 20...0 are used to store the columns of the first line. The next entry contains the second line of the table and the third entry the last line of the table. Each line is distinguished by a code from the coding table, that means in each line is at least one bit set, which is not set in any other line.

Totally this is a table with 3 lines and 21 columns, which allows protecting 63 different modules. The maximum number of different modules, which could be protected by one *WibuBox*, is 190.

Please note: The protection is not realized by a simple check, if a bit in the *WibuBox* entry is set or not. The protection is based on a USER CODE specific encryption and decryption. This makes it not possible to break the protection with a debugger easily.

If the maximum number of different protected modules is less than 190, you can specify additionally a LIMIT COUNTER or EXPIRATION DATE to be stored in the *WibuBox* (maximum 5 entries). This LIMIT COUNTER could be used differently and enables you to limit the usage of different modules. The same applies to the EXPIRATION DATE entries.

6 WibuBox reprogramming

One of the big advantages of *WibuKey* is user-friendly support of additional business with existing customers. It is only necessary to send a *WibuBox* once per customer. Buying additional programs, user licenses or licensing additional program options can be controlled by simply reprogramming the *WibuBox* at the customer's site. This is convenient for both the software developer and the end user.

There are various ways to reprogram a *WibuBox*:

- Via **online** access with TCP/IP directly to the customer PC. The reprogramming of the *WibuBox* connected at the customer's site is the same as the programming of a *WibuBox* in the local network. The access occurs via WkLAN, *WibuKey's* TCP/IP based network system. At the customer's site a WibuKey server process needs to be started. This method offers possibilities to automate the reprogramming. For example, a customer orders an upgrade on the WWW server and **WKCRYPT** is called at the software developer's site to reprogram the *WibuBox* at the customer's site. No distribution or support effort is required for the software developer! This is the ultimate in convenience for the software developer and for the end user. Plus it maintains all of the advantages of a secure copy protection system.

- **Offline** via remote programming. The customer gets an encrypted REMOTE UPDATE FILE by disk, e-mail, fax or phone. An encrypted REMOTE CONTEXT FILE that describes the state of the customer's specific *WibuBox* is necessary to generate a REMOTE UPDATE FILE. The REMOTE CONTEXT FILE can be generated by the customer using the *WibuKey Control Panel Applet* or **WKU** commandline utility or by clicking in the Windows Explorer with the right mouse button. It is possible to store REMOTE CONTEXT FILES at the software developer's site at the time of the original programming to simplify this process. Because of the encryption and the specific state information utilized, remote programming is very secure, even when the data is transferred across insecure methods such as email. Remote

programming is an excellent way to simplify business with existing customers without creating additional costs.

- With **web remote programming** you can reprogram the *WibuBoxes* at the customer's site directly via the Internet. Since the *WibuKey* COM Control is used on the server and the client, both stations have to run on Windows 95/98/Me/NT/2000/XP/Vista/Windows 7. The *WibuKey* drivers must be installed on the server and the client, additionally on the server needs to be installed the Development Kit.
  Web remote programming works like remote programming: The *WibuBox* content is read on the client and sent to the server where the update data is calculated and sent back to the client. The *WibuBox* is programmed on the client's site.
  This procedure needs an ASP made available by the IIS on the server and by the Internet Explorer on the client.
  Web remote programming can be used for ESD (Electronic Software Distribution) since the *WibuBox* can be reprogrammed automatically on the customer's site.

The *WibuKey* **customer administration** (**`WKCAdmin`**) makes your work easier, stores all client relevant data and *WibuBoxes*. You can generate at any time an update file for your client. This application uses the *WibuKey* COM Control and is available on the *WibuKey* CD together with the source code. It has an open database interface (ODBC) and therefore can be linked with own data.

7 Additional WibuKey features

7.1 Pay-per-use, software leasing, software metering

These are all new ways of selling software that make sense for many industries. The cost of the software is no longer associated with the actual use of the software by the customer. *WibuKey* offers the LIMIT COUNTER to implement these features.

Using automatic encryption, the reduction of the LIMIT COUNTER occurs at the program start or during runtime of a program. At runtime, the LIMIT COUNTER is reduced at periodic intervals while checking the *WibuBox*. The software developer defines the interval. With this feature, the run-time of a program can be easily measured and accounted.

With explicit encryption in addition to the above features, metering can be attached to specific functions or features within the application. For example, each page printed or analysis in a calculation program reduces the LIMIT COUNTER.

7.2 EXPIRATION DATE

The use of EXPIRATION DATES makes it possible to accomplish a time limit for demo or test versions of software or to implement time-based software leasing.

With automatic encryption, the EXPIRATION DATE is compared with the system date as a part of the decryption. In most of the cases this is not a security issue, as the comparison happens deep in the *WibuKey* driver. It cannot be eluded very easily. If the user to changes the system date, this would create misleading print outs, backups and cause an uncontrolled chaos in most of today's networked systems. To increase the security there are the following possibilities:

- In addition to the EXPIRATION DATE, a LIMIT COUNTER is set. This is realized with two identical FIRM CODE/USER CODE entries. One has an EXPIRATION DATE as ADDED ENTRY the other a LIMIT COUNTER. The LIMIT COUNTER is set such that it will last at least as long as the EXPIRATION DATE. Resetting the system date one can work with the application for perhaps a bit longer than the EXPIRATION DATE, but only until the LIMIT COUNTER reaches zero.
- The date of the last program start is saved in an ADDED DATA entry inside the *WibuBox*. At the next program start it can be checked to see if the system date has been manipulated. If so, the program can abort.

7.3 Saving data in the *WibuBox*

Saving data in the *WibuBox* is possible. There are two different storage places for data:

- A standard entry like a USER DATA entry and ADDED DATA entry. However, the storage space for data is limited to 5 bytes per entry, or a total of 50 bytes per *WibuBox*. It is recommended to save large text in files encrypted through *WibuKey*. There are no restrictions to the file length. The data is stored in a secure way.
- In a *WibuBox* two kinds of data can be stored:
- A USER DATA entry which can be reprogrammed by anybody without any programming sequence. It can contain user customizable configuration data.
- An ADDED DATA entry. This entry can be changed only with the appropriate sequence, which can optionally depend on the serial number of the *WibuBox*, the data itself and the internal programming counter. In any case, the sequence depends on the FIRM CODE/USER CODE of the corresponding BASE ENTRY. The dependence is defined by the software developer. The dependence on the serial number and the internal programming counter guarantee the maximum security.

ADDED DATA entries can be used to save a check sum, programming parameters or even the last date of program use. USER DATA entries can save a random value (e.g. system clock tick) and to check this value during runtime.

- In the ADDITIONAL MEMORY which is separated in two areas of 8 kBytes USER DATA and 8 kByte PROTECTED DATA. The USER DATA can be accessed like a USER DATA entry without any restrictions. The PROTECTED DATA is secured by a 8 byte signature, which depends on the first entry in the *WibuBox* and, if used, the dependencies flags of an ADDED DATA entry of the first entry.

The ADDITIONAL MEMORY only is available at *WibuBoxes/P* and */U* of version 6 and at *WibuBoxes/RP* and */RU* of version 7.

Detailed information about the use and handling of the EXTENDED MEMORY can be found in the separate document `WibuKey ExtMem.doc` on your *WibuKey* CD-ROM.

7.4 Data protection

With *WibuKey* data can be protected. This is done inside the program and requires the implementation of explicit encryption in the source code. Different encryption algorithms can be used to get an optimum solution in security, speed and data type. The encryption via permutation is ideal to protect `C` strings. It guarantees that the protection result contains only originally used characters. No `0` value shortens the `C` string.

It is also possible to encrypt complete files with the program **FCRYPT32**. It can be called from other programs and offers the excellent possibilities to protect programs in which no API function calls can be integrated or which cannot be automatically protected.

The EXPIRATION DATE and the LIMIT COUNTER can be used in conjunction with data encryption as long as the corresponding bits are set in the SELECTION CODE.

7.5 Use of the EXTENDED MEMORY

The EXTENDED MEMORY of the *WibuBox/+* variants can be used for various purposes. Except of using it for storing the *Hugh License Management* information for protecting multiple licenses in the network, information e.g. about the detailed software version or customer address information can be stored.

Another way of how to use the EXTENDED MEMORY is to store data that is used by the software itself. This could be initialization constant values for algorithms, data calculated at runtime or unique data depending on the user's configuration.

7.6 Web authentication

With *WibuKey* you can safely authenticate a *WibuBox* user. This is done via a two-way-encryption, similar to the Kerberos technique. This method is realized a lot easier and cost-effective than the installation of a public key method with certificates.

8 Protection checklist

The software requires

- Automatic protection
- Individual protection (see Programmer Guide)

The protection of the software is

- Product-specific (for medium- and high distribution quantities).
- Customer-specific (for a small number of high priced software).
- a combination of both methods.

The network support

- is implemented even if the *WibuBox* is initially used at a single workstation. In this case, adding network licensing later is super easy.
- is not implemented, because the program will run only on single workstations.

Protection of modular software

- is planned with automatic protection. More than 190 separate programs or modules for the same customers can be protected with one *WibuBox*.
- is planned with individual protection. This allows the distribution of demo or test versions with certain functions enabled for one program.

The reprogramming of *WibuBoxes* is realized

- online via TCP/IP access.
- offline via remote programming.

The implementation of *WibuKey* features include

- Pay-per-use, software leasing, software metering, expiration dates
- Saving data in the *WibuBox*, data encryption

Part III Installing *WibuKey* Software

All *WibuKey* software components and required drivers are delivered on CD-ROM. The *WibuKey* set up handles the installation process. This insures that all files are installed at the right place no matter which operating system is used.

Software updates can be downloaded any time at http://www.wibu.com.

1 WibuKey *CD-ROM*

The *WibuKey* CD-ROM is delivered with the *WibuKey* Protection Kit and the *WibuKey* FIRM LICENSE.

All customers who have an active Support Contract with Wibu-Systems automatically get the newest version of the *WibuKey* CD-ROM as it is released.

The ***WibuKey* CD-ROM** contains:

- *WibuKey* software and drivers
- *WibuKey* tools and libraries,
- Example programs in different programming languages
- *WibuKey* manuals (online editions)
- The latest *WibuKey* News

Wibu-Systems offers a *Quick Start Guide*. It contains instructions on how to install *WibuKey* software on its first page. This guide is available in many different languages including English, German, French, Dutch, Chinese, and Spanish. It ships with all Protection Kits.
Ask for the *Quick Start Guide* in your language!

2 Installation process

The set up program makes the installation of the *WibuKey* software an easy and fast procedure. It automatically installs all *WibuKey* drivers that are necessary on the operating system in use. Just follow the on-screen instructions and use the default settings to complete the installation.

Before you start to install *WibuKey* software, Wibu-Systems recommends to close all other programs that are running on the computer, especially any programs that use the *WibuKey* protection system.

For DOS there is no special set up program. DOS users copy all necessary files directly from the CD-ROM to folders on the hard disk.

- Insert the *WibuKey* CD-ROM into the CD-ROM drive. The installation program automatically starts on Windows

95/98/Me/NT/2000/XP/Vista/Windows 7. Otherwise run the `cdstart.exe` program from the root directory.

- Click the Software Setup button.
- Select Software:
  Runtime Setup installs the *WibuKey* Runtime Kit with all *WibuKey* drivers and tools (Network Server for Windows 95/98/Me/NT/ 2000/XP, Vista, Windows 7 and Novell Netware, the Server Monitor **`WKSVMON.EXE`**, **`WKU.EXE`**, **`WKU32.EXE`**, and the corresponding help files).
  The *WibuKey* Runtime Kit also has to be installed on the client's site. Thus the *WibuBox* can be called properly.
- Runtime Files opens an html file with 6 current Runtime Kit and Driver Update versions in multiple languages. Here you can select the appropriate Runtime Kit for integration in your installation program. The current versions of the Runtime Kit can always be found on www.wibu.com.
- DevKit Tools Setup installs all developer's tools (**`WKCRYPT`**, **`WKLIST32`**). You need the programs and files for the integration of WibuKey into your software.
- DevKit Samples Setup installs all samples. In case you want to use your program explicitly (e.g. you integrate the WK-API into your source code), you will find implementation samples here.
- Follow the instructions and use the default settings from the set up to complete the installation.
- Connect the *WibuBox* to the corresponding interface on the computer. The arrows on the *WibuBox* point in direction of the computer. For PC hardware, use the convenient thumbscrews to fix the connection. Connect additional devices at the same interface behind the *WibuBox*.

You are ready to work with *WibuKey* now. Wibu-Systems wishes you much success with your software and is looking forward to a long-term partnership with you.

3 Installation on Macintosh

There is a set up in the *WibuKey* folder of the CD-ROM for installation. In the folders `SAMPLES`, `INCLUDE` and `LIB` are samples and libraries for integration of the *WibuKey API* in own programs. The `BIN` folder contains:

	Mac OS 9
`README.TXT`	Latest release information

	Mac OS 9
`WkMacList`	Application for programming and remote programming of *WibuBoxes*
`WkSvMac`	*WibuKey* Server processes
`WibuKey.INI`	`*.ini` file with settings
`WkMACLIB`	Library and driver to access ADB and USB ports for system extension folder
`USBWibuKeyDriver`	Class driver for WIBU-BOX/U at USB for system extension folder
`USBWibuKeyDriver2`	Class driver for W IBU-BOX/RU at USB
`Wkfirm.wbc`	License file for software developer
	Mac OS X
`README.TXT`	Latest release information
`WkMacLibX.framework`	Library and driver to access USB ports for system extension folder
`WkMACLIB`	Library and driver to access ADB and USB ports for system extension folder

4 Installation on Linux

On the installation CD the file `index_en_devel.html` in subfolder Linux contains all information needed for an installation on Linux. This includes

- information on *WibuKey* driver supporting the *WibuBox /U /U+ /RU /RU+ /P /P+ /RP /RP+ /ST* and *CmCard/M* and requirements
- installation packages for SuSE, Red Hat, etc. in form of RPM packages, and for Debian derivates in form of DEB packages
- *WibuKey* Unix source driver
- Information on integrating the *WibuKey* driver into your own install package.

Part IV Protecting software and digital content

Programs can be easily and quickly protected using the *WibuKey* automatic and individual protection.

- With *AxProtector* you have a tool on hand that can automatically protect already complied executables where the source code of your application remains unaltered (see page 65).
- With *IxProtector* you have a protection technology on hand, which allows you to define and protect individual parts (segments) in the source code while developing software. Then during runtime, these segments are linked to different license entries.
  With the *IxProtector* protection technology allows you to define 'real' single segments (modules, functions) in the source code when developing an application, encrypt them, and then link them to license entries at runtime by using index-based placeholders. The *Software Protection API* WUPI (*WIBU Universal Protection Interface)* assists you in this process (see page 191).

The obsolete tool WKCRYPT is described in detail in the external document *WibuKey - Guide for WKCRYPT*.

1 AxProtector: Automatic software protection

No programming skills required

With *AxProtector* you have a tool on hand that can automatically encrypt already complied executables. *AxProtector* allows you to integrate *WibuKey* into your application – quickly and smoothly – without the need to alter your source code. It is so easy to use, that integration can take place without any programming skills.

In just a few minutes, *AxProtector* encrypts and protects your application for a variety of project types:

- Windows 32-bit/64-bit and DLLs
- Mac OS X applications
- Java applications
- .NET applications and DLLs (Framework 2.9, 3.0, 3.5, 4.0)

The .NET 4.0 Framework is also supported. The new commandline variant `AxProtectorNet4.exe` is able to handle .NET 4.0 assemblies. The AxProtector for .NET 2.0 automatically starts AxProtector for .NET 4.0 on the attempt to encrypt an .NET 4.0 assembly.

- Linux applications (see page 128)
- files.

AxProtector provides a standard graphical user interface (GUI) for the automatic integration of software protection and license strategies for all existing Wibu-Systems copy protection systems: *CodeMeter*, *WibuKey*, and the software-based system *CodeMeterAct*. *AxProtector* is also available as a commandline tool for Windows 32-bit / 64-bit, .NET, Linux, and Java applications (see page 172). Using the *AxProtector* GUI is a simple way to generate a commandline that can be extended and used further to accomplish automatic protection.

Table 1 summarizes project types, operating systems and runtime environments available for *AxProtector*.

Project type	Operating System	Run-time	Win GUI	Win Com-mandline	Mac Com-mandline	Linux Com-mand-line
Win 32/64 PE	Windows	✓	✓	✓	✗	✗
.NET Assembly	Windows	✓	✓	✓	✗	✗
Java	Windows	✓	✓	✓	✓	✓
	Linux	✓	✓	✓	✓	✓
	Mac	✓	✓	✓	✓	✓
	Sun	✓	✓	✓	✓	✓
Linux	Linux	✓	✗	✓	✗	✓
Mac OS	Mac OS X	✓	✓	✓	✓	✗

Table 1: *AxProtector* – Project Types, operating systems and runtime environments

AxProtector:

- supports the encryption of all existing *WibuKey* license options. Thus all necessary license information is integrated into the encryption, for example, network licenses, or license checks at runtime.

- features functions to identify debugger use: in the case a debugger is detected, a *WibuBox* can be locked.
- provides the feature of "on-demand-decryption", i.e. parts of the protected application (source code and resources) are decrypted only when accessed. This "on demand decryption" effectively protects against memory dumping and the extraction of unprotected versions.
- offers the use of freely customizable user message dialogs including the creation of individual texts for purchasing options or errors and also the embedding of company logos.

1.1 Structure and Navigation

You access *AxProtector* by the **"Start | Program Files | AxProtector"** system menu item.

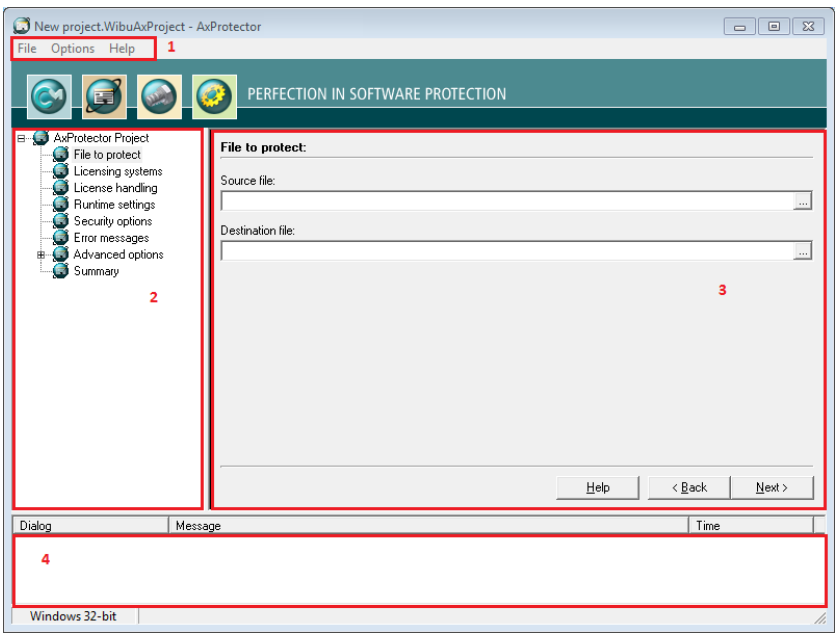


Figure 1: *AxProtector* – Interface and Navigation

The graphical user interface of *AxProtector* consists of four separate areas:

- Menu Bar (**1**)
- Navigation Window (**2**)

- Input Window (**3**)
- Error and Warning Window (**4**)

1.1.1 Menu Bar

File Menu

New Project

To create a new project, please proceed as follows:

❶ Select the **"File | New Project"** menu item. Alternatively, press the <CTRL+N> key combination.
The **"New Project"** dialog opens for selecting the project type.

Open Project

To open an existing project, please proceed as follows:

❶ Select the **"File | Open Project"** menu item. Alternatively, press the <CTRL+O> key combination.
The **"Open"** system dialog opens from which you can select the desired project file.

❷ Select the project file name to be opened, and click the **"Open"** button.

Save Project

To save a created or edited project, please proceed as follows:

❶ Select the **"File | Save Project** menu item. Alternatively, press the <CTRL+S> key combination.

Save Project As

To save an opened project using another project name, please proceed as follows:

❶ Select the **"File | Save Project As"** menu item.

❷ Select a destination folder in the **"Save as"** dialog, and specify the new name of the project file.

In the case this file already exists, *AxProtector* prompts with an overwrite confirmation dialog. Click on the **"No"** button and save the project using a different name, to keep the existing project file.

Export Wbc-file

Selecting this menu item exports the protection settings into a `*.wbc file` you are free to name and save. Later you may use this file in the *AxProtector* commandline tool (see page 189).

This menu item is active only after the project has passed all necessary checks.

Exit

Select the **"File | Exit"** menu item to close *AxProtector*. Alternatively, close the *AxProtector* by the **"x"** control or the <ALT+F4> key combination. Before exiting *AxProtector* you are prompted to save the changes you have made to a project.

Options Menu

Language

AxProtector provides you with different language version for the graphical interface. Select from eight different language settings: Chinese, German, English, Spanish, French, Japanese, Dutch, and Portuguese.

? (help) Menu

Content

Select this menu item to open the *AxProtector* online help.

About

Select this menu item to open a window holding *AxProtector* version information.

1.1.2 Navigation Window

For every project type, the navigation window displays the single protection steps in a tree view. The navigation allows you to access each single step.

The options you may set in single protection steps vary for the different project types.

1.1.3 Input Window

For each protection step, the input window provides for specifying protection options using corresponding fields and controls. You navigate through the single steps by using the **"Next >"** or **"< Back"** buttons at the bottom of each window.

This symbol informs you that you have set additional protection options using the **"Advanced"** button.

1.1.4 Error and Warning Window

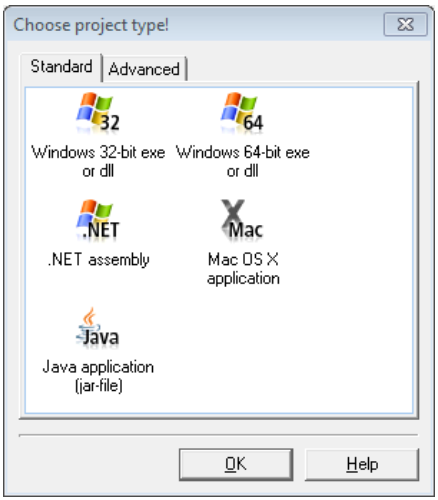
This window displays information, errors or warnings using symbols. You also see the symbols in front of each protection step within the tree view.

Symbol	Description
	When setting an option an error occurred. The protection step involved is not executed. A text informs you about what the error might be. Then you have the option to check your input.
	Please note a warning related to the options you set when protecting your application.
	All settings are correct. This protection step is will be executed.

With a double-click on the ! and ⊗ symbols you will automatically access the protection step to which the information relates.

1.2 Project Dialog

When you open *AxProtector*, or create a new project in *AxProtector*, a project dialog opens where you make the selection from different project types.



You receive help by clicking on the **"Help"** button.

1.3 Project Types

AxProtector features the following project types:

Symbol	Project Type
	Windows (32-bit) applications and libraries (see page 71)
	Windows (64-bit) applications and libraries (see page 71)
	.NET Assemblies (see page 97)
	Mac OS X applications (Universal Binaries) (see page 117)
	Java applications (see page 129)

Symbol	Project Type
	IxProtector only, i.e. project without encryption of the complete application (see page 160)
	File encryption (see page 145)

2 Windows Applications (32-bit/64-bit)

AxProtector protects executable files (applications *.exe and libraries *.dll) in PE format (Portable Executable): The executable files may be created by established compilers, for example, (C, C++; Delphi, VB 6.0, FORTRAN, ...), or by authoring tools (Adobe Flash, etc.).

2.1 File to Protect

To safely encrypt an executable file using *AxProtector*, first select the file you want to protect.

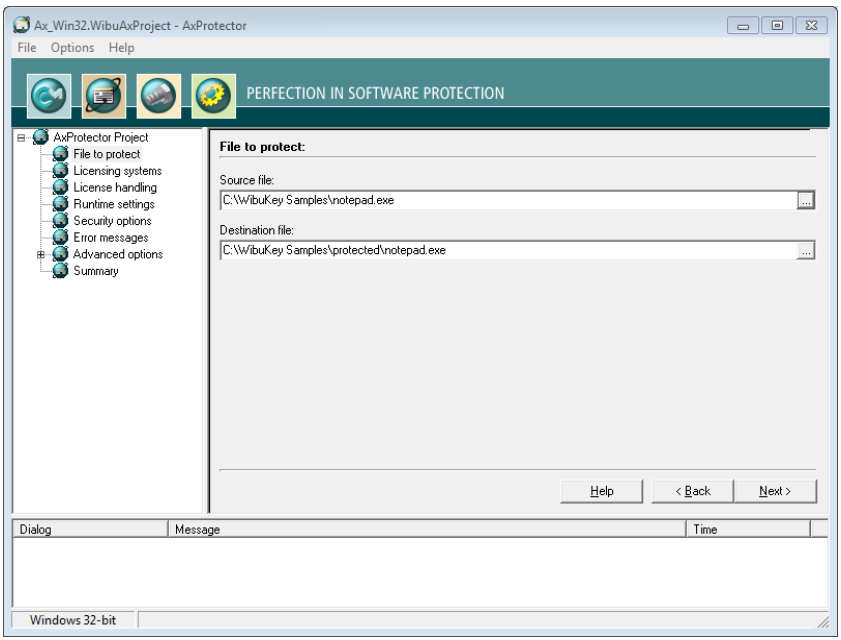


Figure 2: *AxProtector* - Windows "File to Protect"

Source file

Click on the **"..."** button and select the file to protect using the system dialog **"Open"**. Alternatively, manually specify the path and name of the file in this field.

As alternative to the "**...**" button, you may also directly drag & drop the source file from Windows Explorer into the source file field.

Destination File

After you selected the source file, *AxProtector* automatically creates a subfolder `[..\protected\..]`. You may change this default by manually specifying the path and name of the destination file. Then the destination file corresponds to your protected application.

2.2 Licensing systems

After you select the file to be protected, the "Licensing systems" fields will appear in the input window. This is where you can select which protection schemes will be used. Depending on your requirements, you can select one or all of the check boxes.

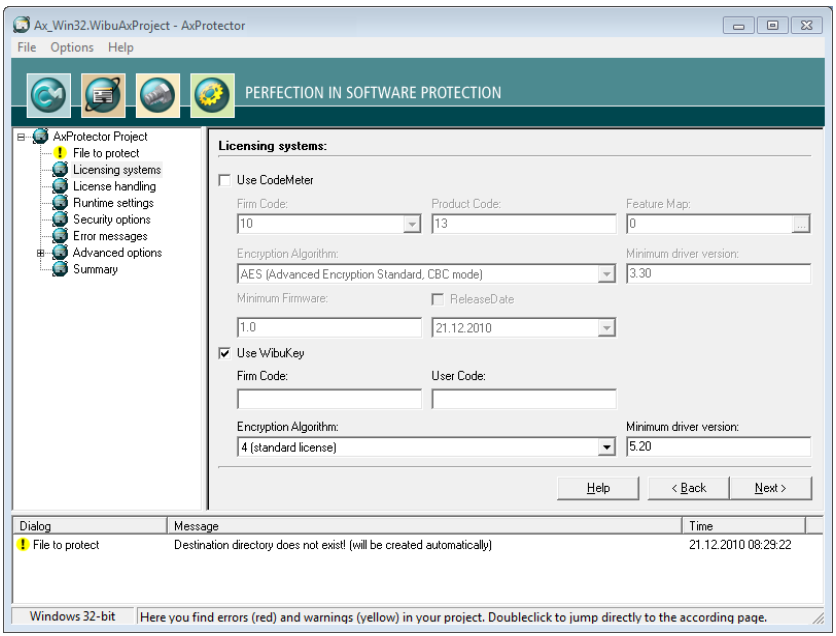


Figure 3: *AxProtector* - Windows "Licensing systems"

If you are switching from *WibuKey* to *CodeMeter*, please activate both hardware platforms.
In this way, you are able to ship updates and upgrades to existing customers who already have a *WibuBox* without the need to replace the hardware. New end-users will be the ones to receive a *CmStick* together with the protected application.

You may also choose to protect using *CodeMeterAct*, the software-based copy protection system. For more information on *CodeMeterAct* visit the Wibu-Systems homepage.

For *WibuKey* the following settings are available:

Firm Code

Specify the FIRM CODE to be used for encrypting the software.

The FIRM CODE 10 used in Figure above is the Evaluation-FIRM CODE found in the *WibuKey* Software Development Kits (SDK).
In real life you would not use a FIRM CODE of 10, since this would be insecure. As a registered licensor, you will be issued your own unique FIRM CODE.

User Code

Enter the USER CODE which defines the encryption of a specific product. You can freely choose this identifier, e.g. for a separate module of a software application, or for a single application.

Encryption Algorithm

Select the algorithm to encrypt your software. By default, *WibuKey* currently supports Algorithm 4. Algorithm 5 is for reduced licenses. Use Algorithms 1 to 3 for downward compatibility only. .

Minimum Driver Version

Enter the minimum *WibuKey* driver version. When setting the minimum driver version to 5.20 the session handling for terminal servers is automated. This means that *AxProtector* automatically handles sessions of the protected software, and each session is allocated one of the available licenses.

Setting the driver version is also required when, for example, you wish to use new features for the encryption of an application. Older driver versions will not support these new features, and will trigger error messages when starting the protected software.

For setting *CodeMeter* / *CodeMeterAct* options, see the separate *CodeMeter* Developer Guide.

2.3 License Handling

This input window lets you to define whether the protected application is to search for existing licenses locally in *WibuBoxes*, in the network, or both. Moreover, you can define the license allocation mode.

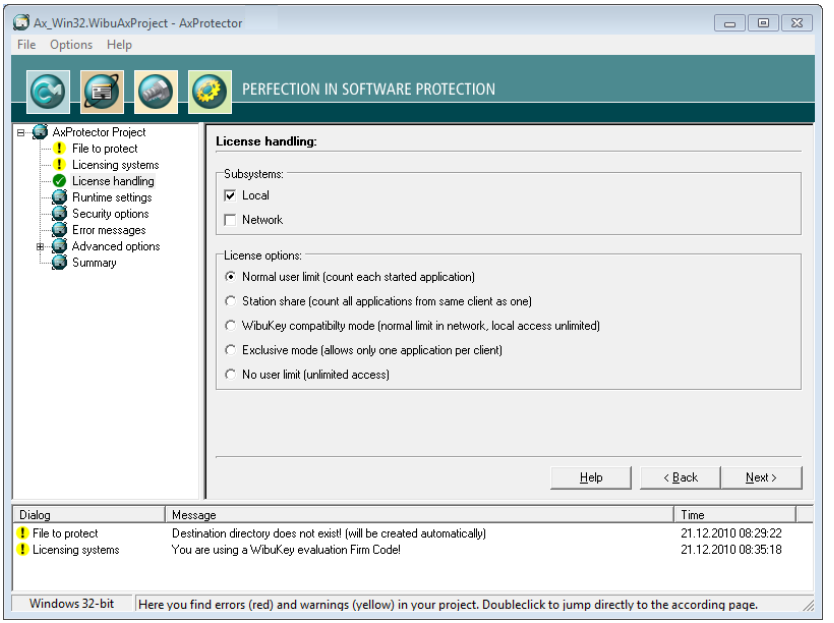


Figure 4: *AxProtector* - Windows "License Handling"

Subsystems

Here you can define in which subsystem (local or network) the protected application is to search for matching license(s).

Local

This setting determines if the protected application searches exclusively for licenses located on the same PC, or allocated to the same VMware.

Network

This setting determines that the license of the protected applications is to be sought in the network, i.e. only PCs are accessed where *WibuKey Server* runs.

On selecting both subsystems at the same time, the license is first sought locally and then subsequently on the network.

License Options

In this group you define how started instances of the protected applications perform together with the allocation of licenses.

Normal user limit

Here each started instance allocates a single license. It does not make a difference if the *WibuBox* was found locally, or on a network.

Station Share

Here multiple instances can be started on a single PC. But, allocate only a single license.

You use this setting, for example, when you want to provide the end-user with the option of starting the application several times. On a terminal server each session allocates a license. In virtual machines each machine allocates a license.

WibuKey Compatibility Mode

Here each started instance in the network allocates a license (normal user limit) but the local access is unlimited (no user limit).

This allocation option exists only because of compatibility issues with *WibuKey*. Wibu-Systems recommends the setting 'normal user limit' and 'station share'.

Exclusive Mode

Here a protected application can be started only once on a PC.

No user limit

Here any number of instances of the protected application can be started locally or in a network, and no additional licenses are allocated. Allocated licenses in this mode can be re-used.

2.4 Runtime Settings

This input window lets you define the application's runtime settings, e.g. license checks in *WibuBoxes*, issue warnings, etc.

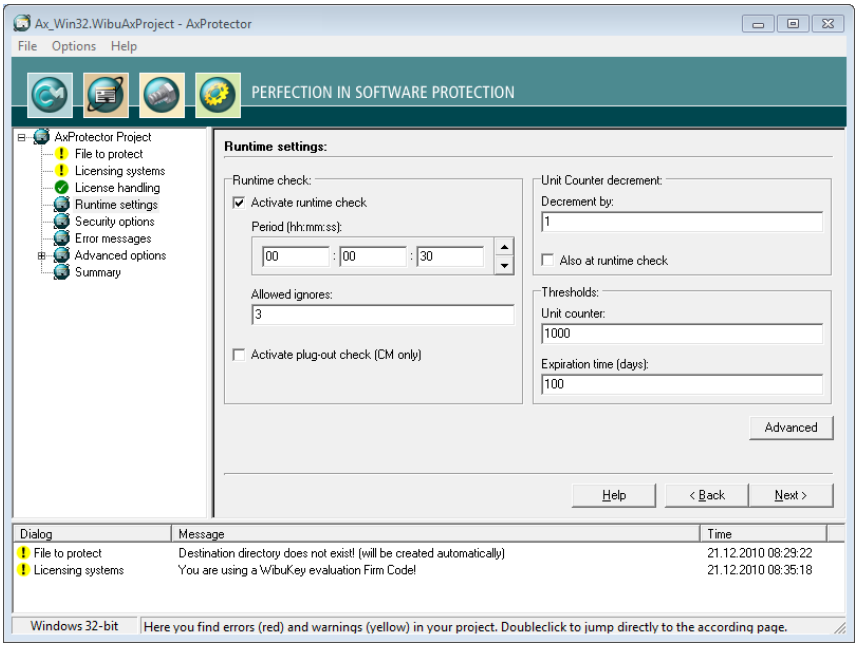


Figure 5: *AxProtector* - Windows "Runtime Settings"

Runtime Check

In this group you define whether and how often the protected application checks the license at runtime.

Runtime Check Activates or deactivates the check at runtime.

Period Defines the period between two checks. You specify this time interval in the format: hours: minutes: seconds.

Allowed Ignores Defines how often the end-user is able to ignore a failed check.

If the connection to a *WibuBox* should fail or the license cannot be accessed, you can assign a reasonable number of “ignores” allowing the end-user to continue working without a license access.

Activate Plug-out Check (*CodeMeter* only) This option closes the protected application when a *WibuBox* is removed while the application is running. Immediately, an error message is issued. This option is valid for *CodeMeter* only.

Unit Counter Decrement

Decrementing an *UNIT COUNTER* can serves to establish the validity of licenses in a *WibuBox*. This group allows you to define this behavior.

Decrement by

Defines the value by which the *UNIT COUNTER* is decremented. This option causes a decrement of the counter when the protected application starts.

When the **"Also at Runtime Check"** option is activated and the specifications are set as shown in Figure 5, every 30 seconds (see the defined period) a set UNIT COUNTER is decremented by a value of 1.

Also at Runtime Check

Decrements the UNIT COUNTER also at runtime of the protected application.

This option works only when the **"Also at Runtime Check"** option in the **Runtime Check** group is activated.

Thresholds

In this group you define when a message is issued to give information on the validity of a license.

Unit Counter

Where the defined threshold falls short, a warning message is issued.

Expiration Time (days)

When the specified EXPIRATION TIME (in days) is achieved within the defined threshold, a warning message is issued.

2.5 Advanced Runtime Settings

This input window lets you define further settings at the runtime of an encrypted application.

With the exception of the UNIT COUNTER and EXPIRATION TIME check at runtime and the "Advanced option" area all other settings are valid only for the licensing systems *CodeMeter* or *CodeMeterAct*.

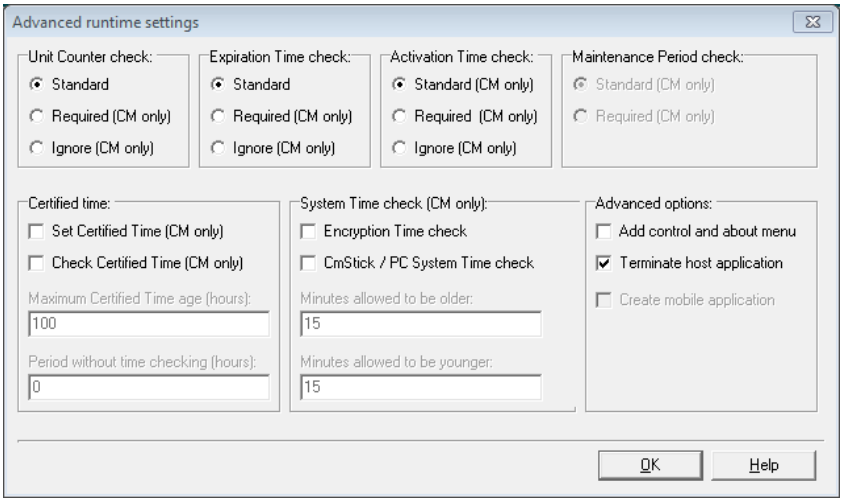


Figure 6: *AxProtector* - Windows "Advanced Runtime Settings"

Unit Counter Check

Defines the handling of a *UNIT COUNTER* when set in a license.

Standard

Decrements at runtime and/or start time an existing *UNIT COUNTER* entry in a license by the value defined on the previous page. When the *UNIT COUNTER* reaches 0 (null) the encrypted application does not start.

Expiration Time Check

Defines the handling of an *EXPIRATION TIME* set in a license.

Standard

Checks for an existing *EXPIRATION TIME* entry in a license. However, the application also starts when no *EXPIRATION TIME* entry exists, or the current date precedes the *EXPIRATION TIME*.

Advanced Options

This group allows to set further options.

Add Menus

Adds the "About" and "Control" menu items to your application.

Terminate Host Application

When no valid license is found, in the case of protected DLL application files the calling *.exe is terminated.

Create Mobile Application

[pending to be implemented]

2.6 Security Options

This input window lets you select from different mechanisms and methods for protecting your application. You are able to scale the degree of security for

yourself, for example, how intensive the search for debugger is to be, or whether a *WibuBox* is locked.

When the options you set here turn out to be incompatible with your protected application, you are also able to separately deactivate single security options.

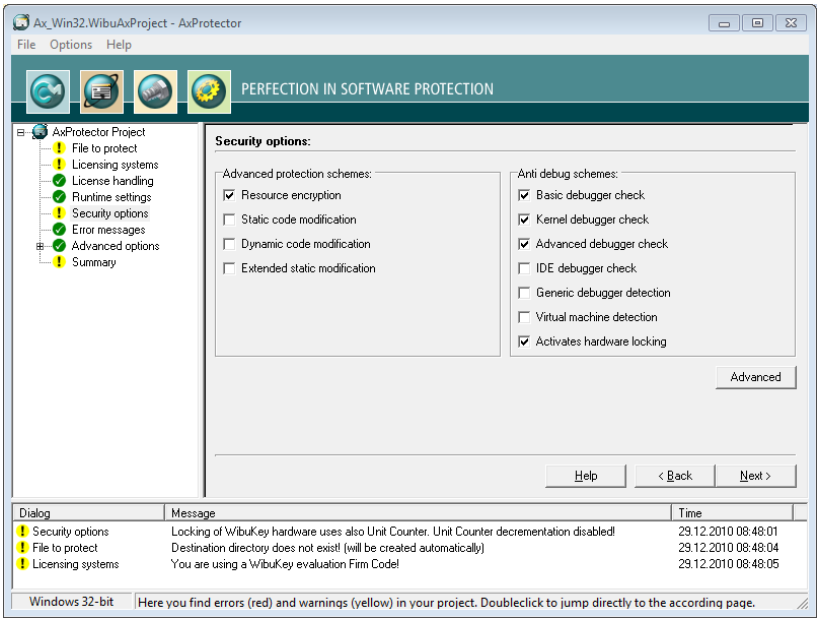


Figure 7: *AxProtector* - Windows "Security Options"

Advanced Protection Schemes

The advanced protection schemes deeply intervene into your application. In some cases, this may mean that some single mechanisms will not work due to compatibility reasons.

Resource Encryption

Also encrypts the resources of your protected application. After the start of your application, the resources located in the PC memory and are decrypted "on demand".

Static Code Modification

Your software is modified in a way so that it is protected against debugging, dumps and reverse engineering. These modifications are added to your application before the encryption.

Dynamic Code Modification

The source code of the application to be protected is modified dynamically at runtime of the application.

Extended Static Modification

This option adds extended static modifications to your application.

You can choose either static or extended static modification. Both types of modification take place before the encryption of your application.

Anti-Debug Schemes

Debugger programs serve an honest role in searching for error and finding bugs. But they may also be used by hackers to analyze software. In this group you determine how to react to debugger programs.

Basic Debugger Check

Checks whether a debugger is attached to your application. In the case a debugger is found, your application will not be started or exited.

Kernel Debugger Check

Additionally checks for Kernel debugger programs, such as, SoftICE. In the case a debugger is found, your application will not be started.

The next two mechanisms comprise methods for detecting specific debugger programs and tools.

Advanced Debugger Check

Checks in an advanced search for debugger programs which may run parallel to your application, also cracker tools, such as, ImpREC, are detected. In the case a debugger is found, your application will not be started.

IDE Debugger Check

Checks for all debugger programs. With this option, debugger programs are not allowed at all, i.e. even within developer environments, e.g. Visual Studio, Delphi. In the case a debugger is found, your application will not be started.

Generic Debugger Detection

Adds a mechanism to the application preventing the attachment of a debugger program to the application at runtime.

Virtual Machine Detection

Detects whether the application is to be started on a virtual machine, and prevents this.

Activates Hardware Locking

This option locks the used *WibuBox* as soon as a debugger program is detected. In this case the LIMIT COUNTER is set to a value of `0`.

Wibu-Systems recommends to deactivate this option when using the *WibuKey* feature LIMIT COUNTER. This because the *WibuBox* will be locked when the LIMIT COUNTER reaches a value of `0`.

The owner / end-user of the locked *WibuBoxes* must contact the software vendor for unlocking codes. How and how often unlocking is granted depends on the vendor's policy.

Subsequently, the developer is able to unlock the *WibuBox* via remote programming.

2.7 Advanced Security Options

This input window lets you define further settings.

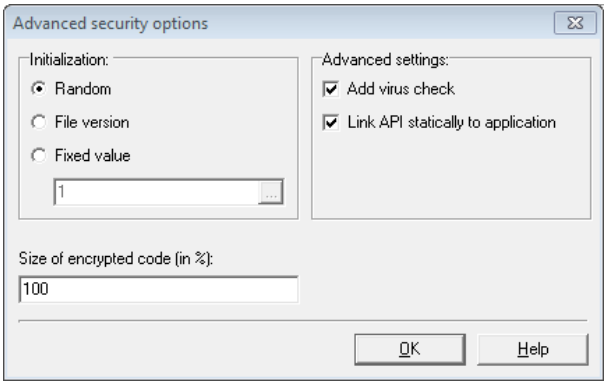


Figure 8: *AxProtector* - Windows "Advanced Security Options"

Initialization

For the initialization of the encryption of a protected application several options exist.

Random

Uses random data to initialize the encryption.

File Version

Uses the file version as an integral part in initializing the encryption.

Fixed Value

Uses as fixed value to initialize the encryption.

Use a fixed value for initialization in order to get binary-compatible results each time you encrypt the same application.

For example, this makes sense when you want to regenerate an encrypted application from an earlier project version. This guarantees an exact match to an earlier encrypted and delivered version.

Advanced Settings

This group allows the setting of further options.

Add Virus Check

Adds a virus check to the protected application by using a check sum.

Link API statically to Application

The *WibuKey* Core API is statically linked to the protected application. This option increases security but also increases the sizes of the executable file.

Size of encrypted Code (in %)

Specifies the portion of the code to be encrypted stated as percentage number.

2.8 Error Messages

This input window lets you define the messages displayed if errors occur. You define whether a user message DLL with a separate error display is used, or whether you use default error message windows.

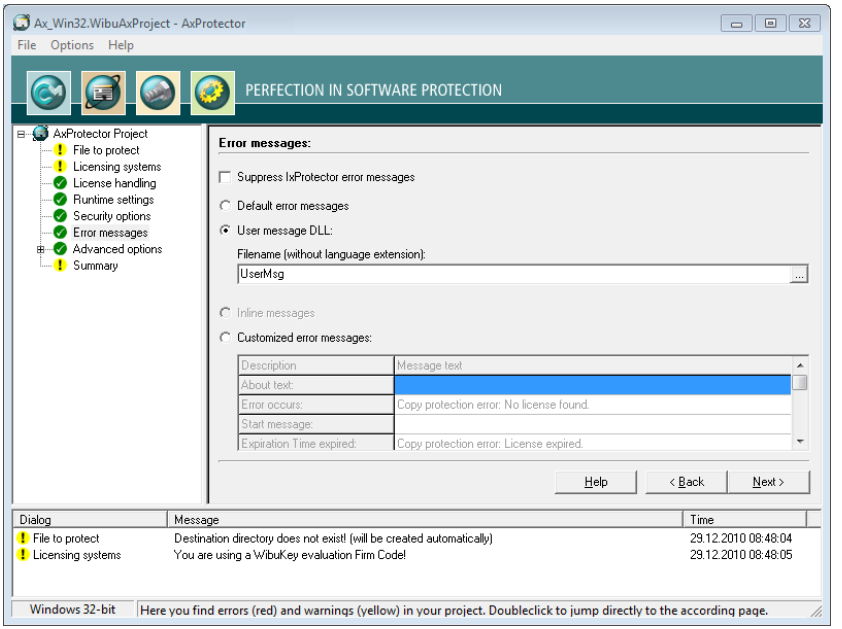


Figure 9: *AxProtector* - Windows "Error Messages"

Suppress *IxProtector* Error Messages

The output of *IxProtector* error messages is suppressed.

If you do not activate this option, when using *IxProtector* errors, additional message windows are displayed along with the messages you program in the project.

Default Error Messages

All errors occurring at the runtime of a protected application display default error messages.

User Message DLL

The ability to use the `User Message DLL` is activated. Error messages can be localized to different languages using `*.ini` files. In addition, you have the option to integrate your own designs to this file, for example, by using separate logos or text.

The `*.ini` files with the respective country suffix and the `Dll` program library are automatically saved to the directory where the application locates the files protected by *AxProtector*.

File Name

Enter the file name without specifying path and language file extension.

The `UserMsgDll` is copied from the directory `%Program Files%\WIBU-SYSTEMS\AxProtector\DevKit\bin\UserMessage`. The corresponding `*.ini` files are also saved to this directory.

Inline Messages

Links for .NET projects, with an inline assembly which can also be configured by `*.ini` files.

This option is available for the encryption of .NET applications only.

Customized Error Messages

Activate this option to enter customized error messages displayed in the message boxes below.

2.9 Advanced Options

This input window lets you set further options for the encryption using *IxProtector* and for the project type file encryption.

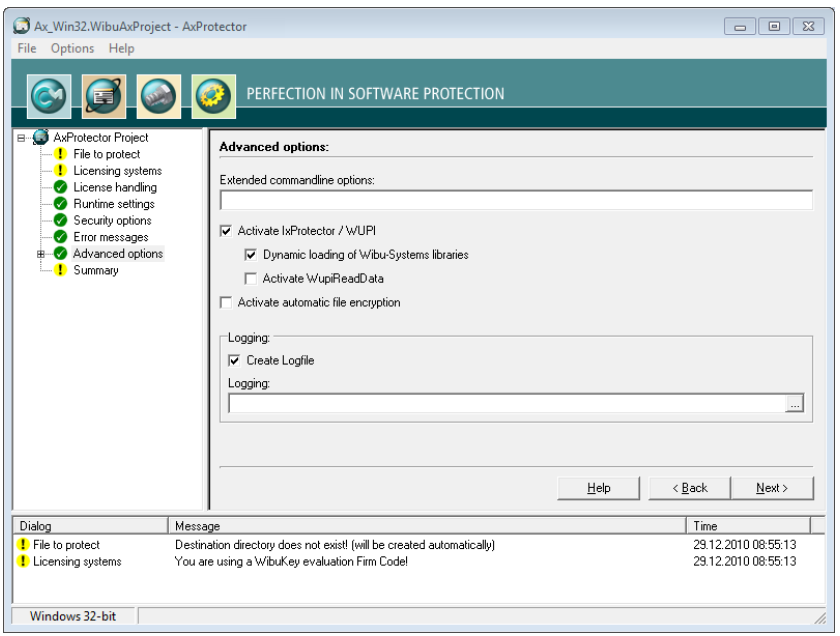


Figure 10: *AxProtector* - Windows "Advanced Options"

Extended Commandline Options

Here you are able to directly enter extended options or new feature functions using *AxProtector* commandline parameters.

For more information please contact support at Wibu-Systems.

Activate IxProtector / WUPI

Activate this checkbox to allow for the later creation and editing of license lists and function lists. These you need to protect using *IxProtector* via the *Software Protection API (WUPI)* (see page 172).

Dynamic loading of Wibu-Systems libraries

When activated this checkbox results in a special, more time-intensive process in the case of VB6 applications or on dynamic loading of Wibu-Systems libraries.

Activate WupiReadData

When activated this checkbox results in reading data from the *WibuBox* which has been previously stored at a defined location (see page 195).

Activate Automatic File Encryption

Activate this checkbox to trigger the automatic decryption of files by the *AxProtector* engine.

Create Logfile

Activate this checkbox to create file logging for the activities of *AxProtector*.

Logging

Specify the path and file name of this log file.

If you specify the name of the file only, by default, this file is saved to the directory `%\Program Files\WIBU-SYSTEMS\AxProtector\DevKit\bin`.

2.9.1 License Lists

This menu item lets you manage license lists. Those you need to protect using *IxProtector* via the *Software Protection API (WUPI)* (see page 196).

License lists consist of a unique identifier (**ID**), a **Description** and hold specifications on **Items** and **Item Details**.

This **ID** corresponds to the index number you require when addressing a license using most of the WUPI commands (see page 196).

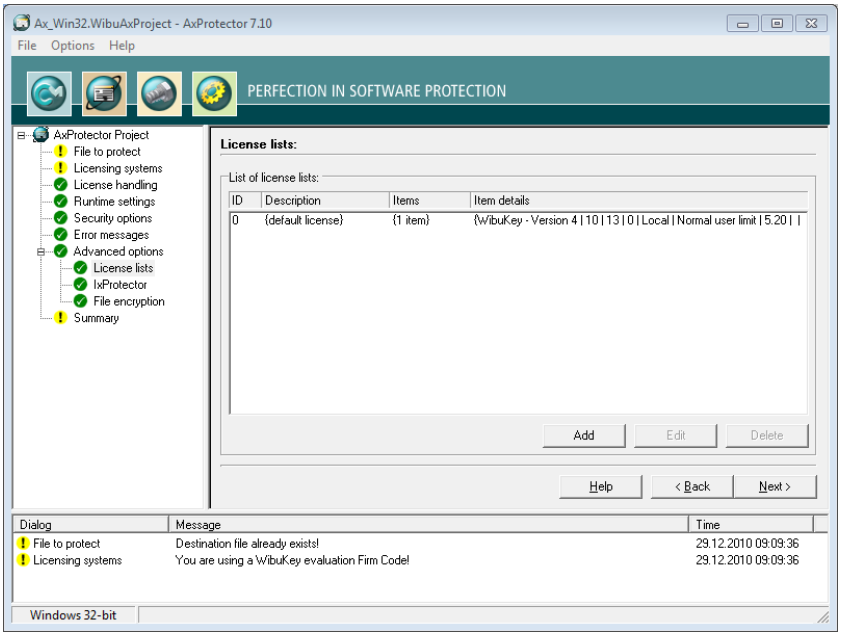


Figure 11: *AxProtector* - Windows License Lists

2.9.2 Add License Lists

This menu item lets you create license lists. Please proceed as follows:

❶ Click the **"Add"** button.

❷ Assign an **ID** in the *LICENSE LIST* group and complete the **Description** field.

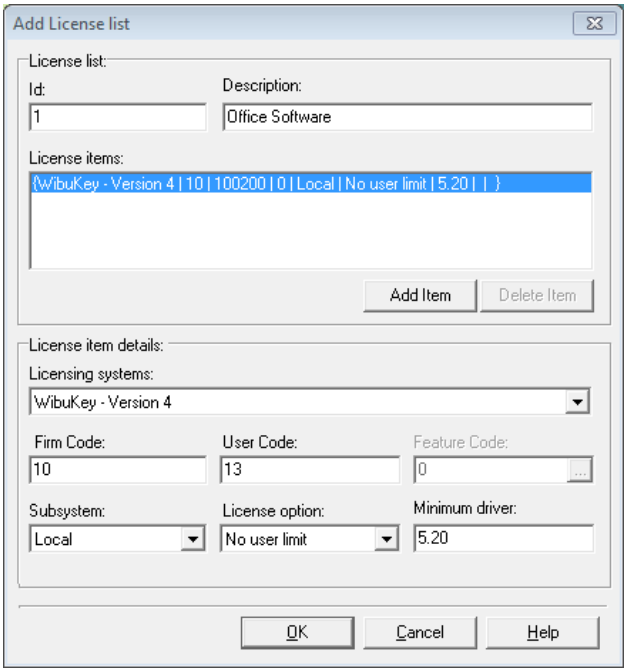


Figure 12: *AxProtector* - Windows Add License Lists

Id

This ID uniquely identifies a license list and serves for referencing.

By default, an **ID** of 0 is initially set by the selection of the copy protection system. Following, you are able to add license list entries starting with an **ID** of 1.

Description

Here you will describe a license list with text.

❸ Define the license by completing the fields in the *LICENSE ITEM DETAILS* group.

Licensing systems

Select the copy protection system used for protection of the license (*CodeMeter, CodeMeterAct* or *WibuKey*).

Firm Code

Enter the FIRM CODE used for the protection of the license.

User Code

Enter the PRODUCT CODE used for the protection of the license.

Feature Code

This option is valid only for the licensing systems *CodeMeter* or *CodeMeterAct*.

Subsystem

Select the subsystem in which the protected application is to search (local or network), and define the search order.

License Option

Select the options for license allocation:

- *NORMAL USER LIMIT*
- *STATION SHARE*
- *WK COMPATIBILITY MODE*
- *EXCLUSIVE MODE*
- *NO USER LIMIT*

Minimum Driver Version

Specify the required minimum driver version for the protected application.

❹ Click on the **"Add"** button in the *LICENSE LIST* group.
The summary of your specifications are displayed in the license item list.

❺ Click the **"OK"** button.
The new license data is added to the license list.

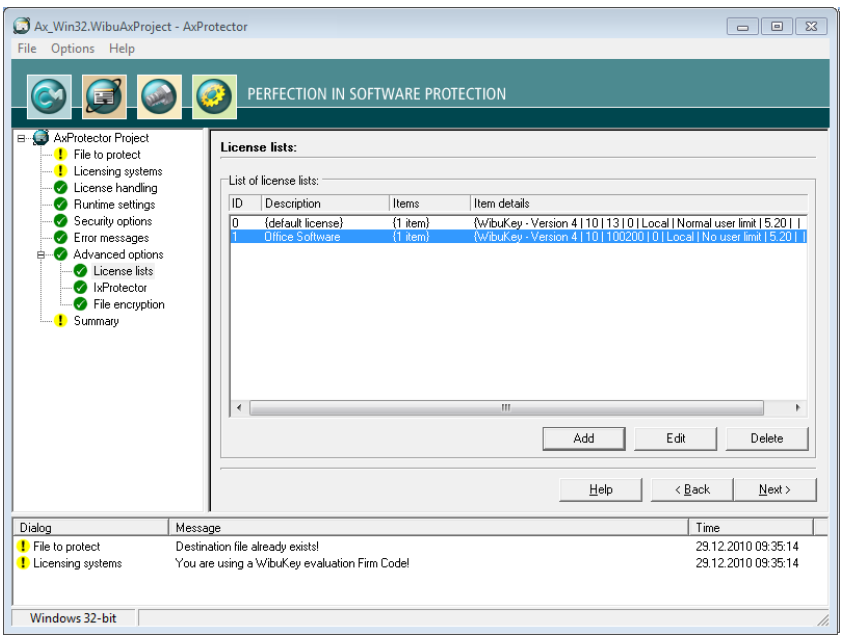


Figure 13: *AxProtector* - Windows Specified License List

2.9.3 *IxProtector*

This menu item lets you define single modules or program functions of the protected application (see page 172ff).

Even when you use *IxProtector* without any further options, i.e. only the explicit encryption of functions, you nevertheless obtain more security for your application.

In this case, *CodeMeter* and *WibuKey* API calls, using the dynamic library (*.dll) are redirected to the corresponding statical libraries and appended to the application. Since the dll interface is left out, the security increases without making any changes to your application.

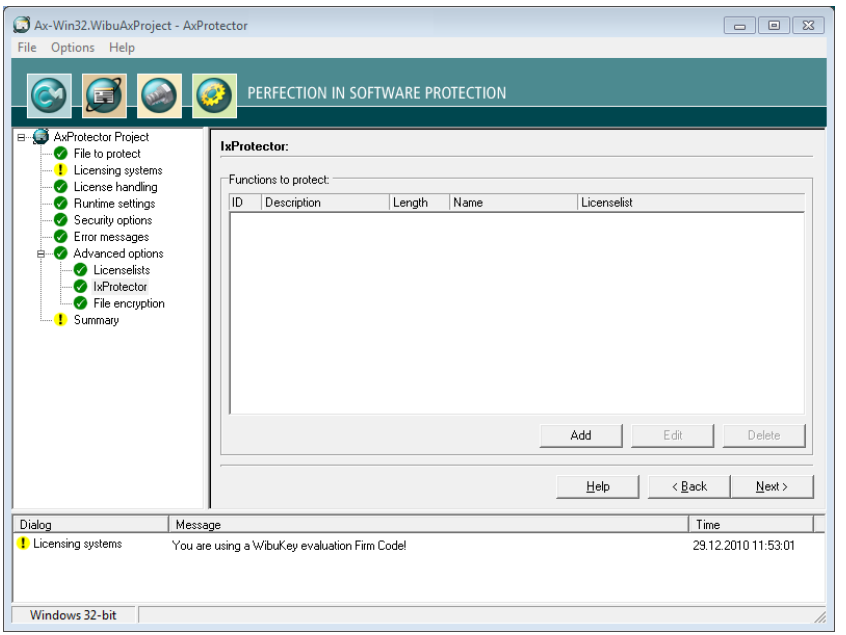


Figure 14: *AxProtector* - Windows *IxProtector* – Function List

Functions to protect

Lists all specified function lists, including all properties.

2.9.4 Add Function Lists

This menu item lets you create function lists. Please proceed as follows:

❶ Click the **"Add"** button in the *IxProtector Options* group.

❷ Define the function by completing the fields in the *FUNCTION* group.

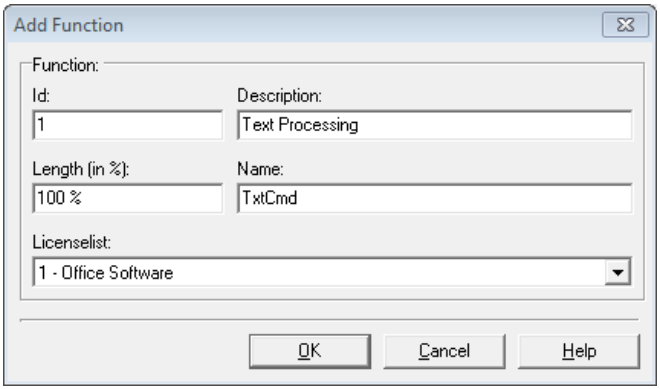


Figure 15: *AxProtector* - Windows *IxProtector* – Add Function

ID

Uniquely identifies the function.

This **ID** corresponds to the identification you use when calling the WUPI commands **`WupiDecryptCode`** and **`WupiEncryptCode`** (see page 196).

Description

Enter a description of the function with text.

Length

The length of the array to be encrypted for the function is specified here. You enter the length, in percent, anywhere from 0 to 100%. If you want this number to represent percentage, you must enter the percent character (%). Alternatively, you are able to specify the length by number of bytes. Then *AxProtector* automatically calculates the length.

Name

Specify the name of the function to be encrypted.

The function name must exactly match the name used in the export list of the linked map file.
Please note the correct spelling (case sensitive, underline, etc.).

In order to obtain the exact function name from the executable file, you may, for example, use the application Microsoft Dependency Walker.

Microsoft Dependency Walker shows dependencies between 32- or 64-bit Windows PE files. A tree view shows all linked modules and imported and exported functions are displayed in tables. Dependency Walker is part of Windows XP SP2 Support Tools and also part of Microsoft Visual Studio including version 8.0 (Visual Studio 2008, i.e. version 9.0 no longer provides Dependency Walker).

License List

Selects an existing license to which the function is assigned. Then this license list is used for the encryption of the function.

❸ Click the **"OK"** button.
The new functions are added to the function list.

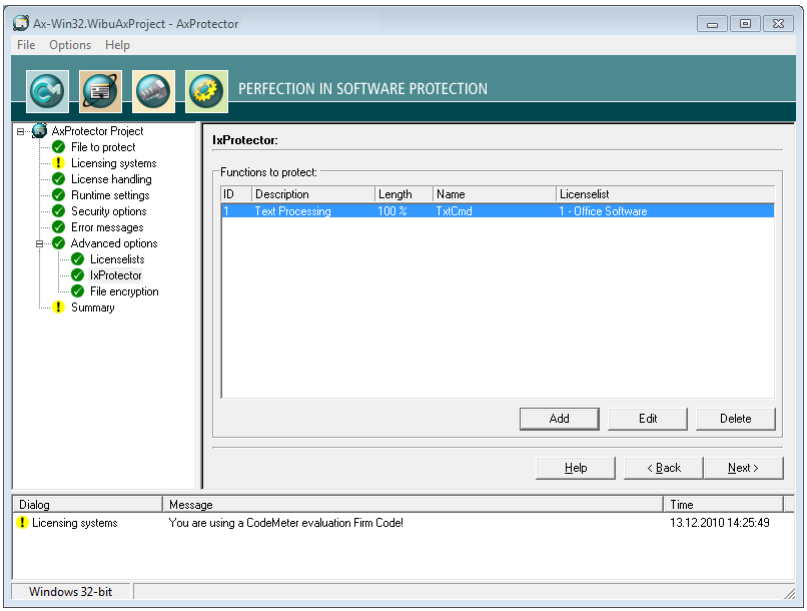


Figure 16: *AxProtector* - Windows *IxProtector* – Specified Function List

2.9.5 File Encryption

This menu item lets you define the rules on how an application accesses the encrypted files. In addition, you have the option to define those rules in a list for

different file types. You can add as many file types as possible. For a file only one file type is required.

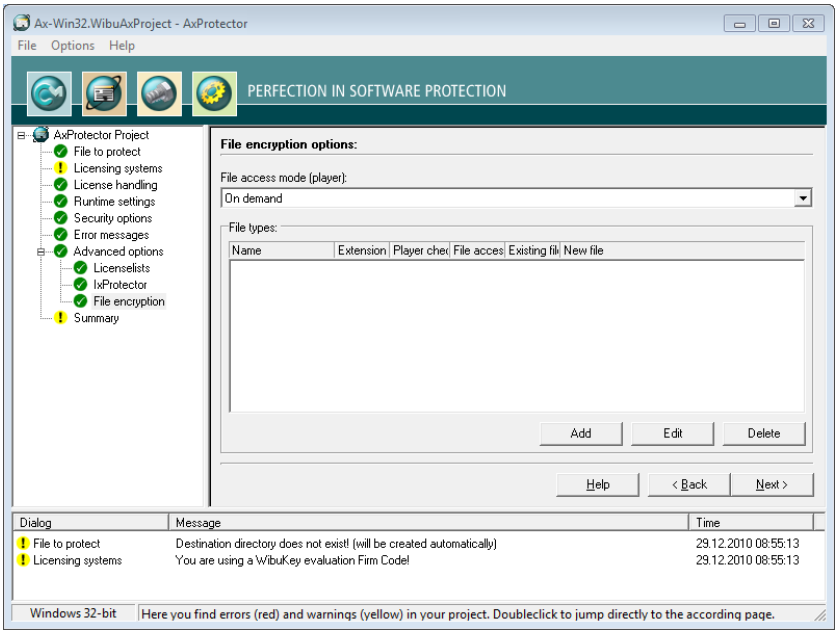


Figure 17: *AxProtector* - Windows File Encryption

Add File Types

Click on the **"Add"** button to add a new file type.

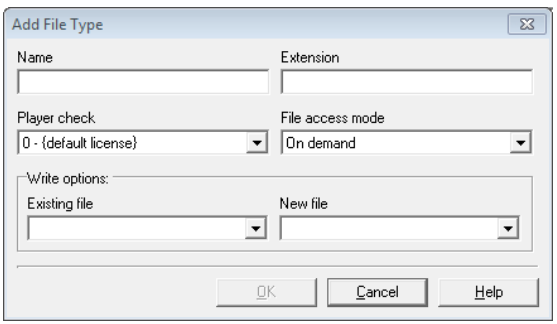


Figure 18: *AxProtector* – File Encryption "Add File Type"

Name

Enter a describing descriptive name for the file type. This name has no impact on the encryption.

Extension

Enter the file extension of the file type you create, e.g. `txt` for text files.

Player Check

In this group you define whether the license options of the accessing application (player) are checked when the encryption takes place.

Option	Description
`License list`	The player has to be encrypted using a license from this license list. For example, this allows you to define that a specific file type is accessed exclusively by the application you encrypted.
`No player check`	No check of the accessing application is performed.

File Access Mode

The File Access Mode defines how the player is prepared for the access of protected files. This mode allows you to configure the memory required and the runtime behavior.

The selection of a suitable mode depends on the type of the player and the size of the file. For example, when working with video files you should select "Huge file mode (read only)". In the case of smaller files (configuration files) you may access several times, the mode "At once" is preferable.

Since the selection of different runtime settings for the player and the data are possible, at runtime the more restrictive settings apply.

On demand

The player reserves RAM space for the complete file to be read; but reads only the required part – strictly speaking all 4 Kbyte blocks are holding this part – and decrypts these blocks. For further accesses to the protected file, more required blocks are loaded (on demand) and decrypted. When the required part is located in blocks already loaded, the decrypted image in the memory is used. In this way, step-by-step the player builds up a complete memory image of the required file.

This mode requires a lot of memory (the same size as the file to be loaded). However caching the decrypted data provides for good performance at runtime when accessing already decrypted blocks. This mode is available for read and write access.

At once

The player reserves RAM space for the complete file to be read; completely reads it, and completely decrypts it. Further accesses to the protected files, use the decrypted memory image.

This mode requires a lot of memory (the same size as the file to be loaded). However, caching the decrypted data provides a good performance at runtime. Compared to the "on demand" mode, this mode requires more time for first access (the file is completely loaded and decrypted). The performance of each additional access is increased because the file resides completely in memory, in a decrypted form. This mode is available for read and write access.

Huge file mode

The player reads the currently required parts of the protected file and decrypts them. This data is not cached in the memory.

This mode requires no additional memory. Multiple accesses to the same data means that the data has to be read and decrypted each time. This mode is available for read access only.

Write Options

Existing File

In this group you define the settings on how changes to an existing file are saved.

Option	Description
`Original`	Changes are allowed. Where the file was encrypted, it is re-encrypted. Unencrypted files are saved with no decryption.
`No writing`	Write actions are not allowed. Just read-only access is allowed.
`License list`	Changes are only encrypted using the license options defined in the selected license list.

New File

In this group you will define the settings on how new files are saved.

Option	Description
`Plain`	New files are only saved unencrypted.
`No writing`	New files cannot be saved.  A new file is saved, however no data is saved to this file.
`License list`	New files are only encrypted using the license options defined in the selected license list.

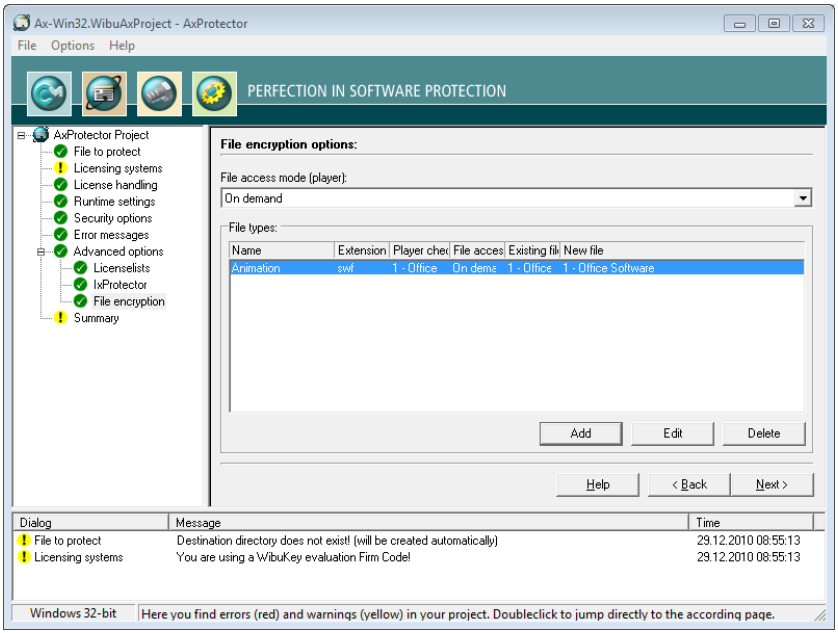


Figure 19: *AxProtector* - File Encryption "Specified Option List"

2.10 Summary

This input window shows you a summary of all the settings you defined for the automatic protection of your application, and allows you to start the encryption process.

For subsequent use, the contents of this page can be copied to a `*.wbc` file (WIBU Configuration file). Copy the content into a text file, and change the file extension to `*.wbc`.

Alternatively, you may also use this file to protect your application using the *AxProtector* commandline tool. In the commandline type `AxProtector.exe @*.wbc` (see page 189).

Alternatively, using the **"File | export wbc-file"** menu item, you can also create the corresponding `*.wbc` file.

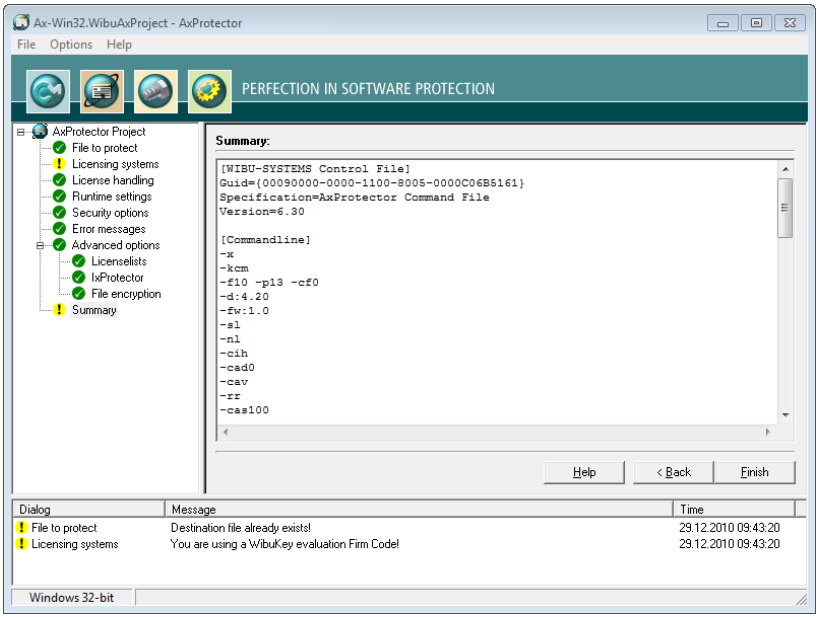


Figure 20: *AxProtector* - Windows "Summary"

Finish

Starts the encryption using *AxProtector* applying the settings you previously defined.

2.11 Protection Result

The result of the encryption with all relevant settings is displayed in a separate window.

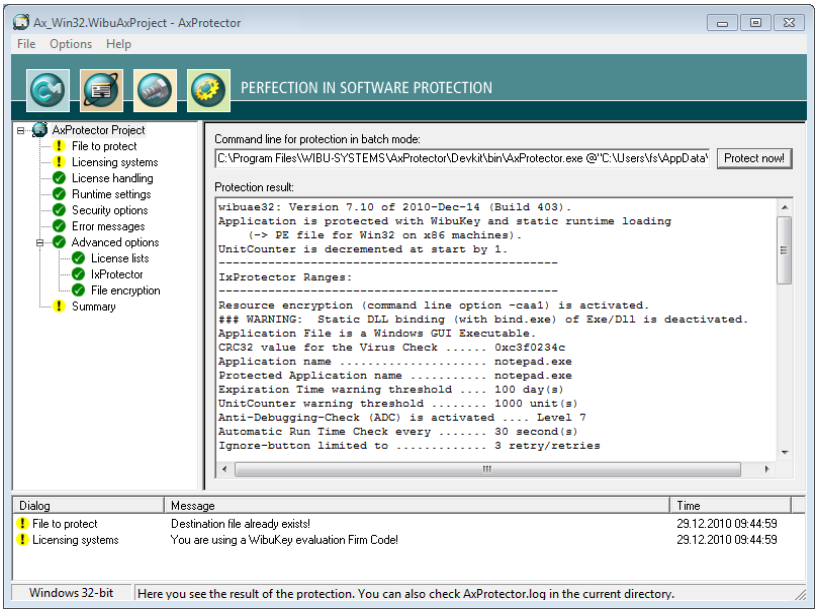


Figure 21: *AxProtector* - Windows "Protection Result"

Protect Now

When you need to repeat the encryption operation, click the **"Protect Now"** button. Then the *AxProtector* commandline is executed in batch mode.

You are also able to copy the *AxProtector* commandline for the batch mode to the clipboard and insert it in the commandline input. Subsequently, you can edit it and apply any desired changes.

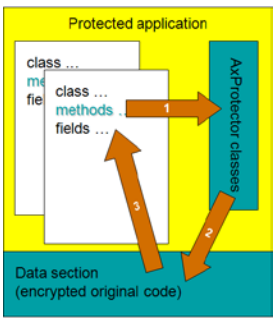
3 .NET Applications

In principle, a .NET assembly is an open book to hackers: using capable tools, e.g. Reflector, disassembling of your code and thus reverse engineering is quite simple. In order to prevent unauthorized analysis or modification, your executable code should always be encrypted before delivery. In doing so, *AxProtector* proceeds as follows:

How does it work?

- Your assembly is disassembled by the tool *AxProtector* for .NET.
- Classes, methods and fields are extracted from the original assembly.
- A new assembly is created.
- Classes are created with the same names, methods and fields.
- The newly created methods, however, do not hold the original code but instead make calls to the *AxEngine*.
- The original code is encrypted by the license you select, and is appended to the data section.

At the first call of the encrypted method, the code inserted by *AxProtector for .NET* calls the *AxEngine*. The *AxEngine* decrypts the original code stored in the data section, and calls the encrypted code. Because the original methods keep their original names, you are still able to call them from outside. Even the parameters (type and description) stay the same.



However, disassembling the encrypted code is not possible.

You can define for yourself which methods are encrypted, and which locate unencrypted in the assembly. This you define optionally for a complete name space, a complete class, or a single method.

A definition at the method level overrules definitions at the class level. The same holds for the class and name space level.

At the same time, you determine whether encryption takes place using the default license, not at all, or separate license lists are used.

With the latter option you automatically realize modular software protection.

3.1 File to protect

To safely encrypt an executable file using *AxProtector*, first select the file you want to protect.

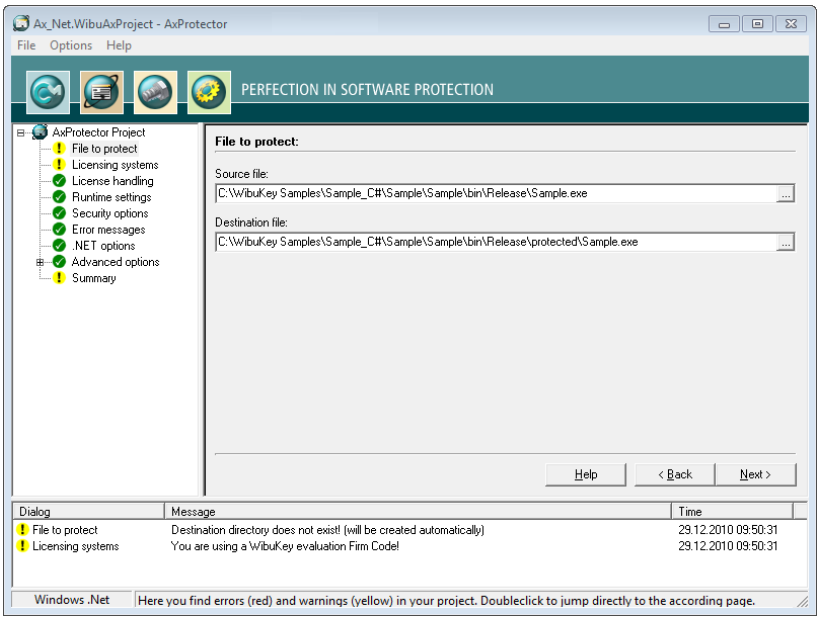


Figure 22: *AxProtector* - .Net "File to protect"

Source File

Click on the **"..."** button and select the file to protect using the **"Open"** system dialog. Alternatively, manually specify the path and name of the file in this field.

As alternative to the **"..."** button, you may also directly drag & drop the source file from Windows Explorer into the source file field.

Destination File

After you selected the source file, *AxProtector* automatically creates a subfolder `[..\protected\..]`. You may change this default by manually specifying the path and name of the destination file. Then the destination file corresponds to your protected application.

3.2 Licensing Systems

After you select the file to be protected, the "Licensing systems" fields will appear in the input window. This is where you can select which protection schemes will be used. Depending on your requirements, you can select one or all of the check boxes.

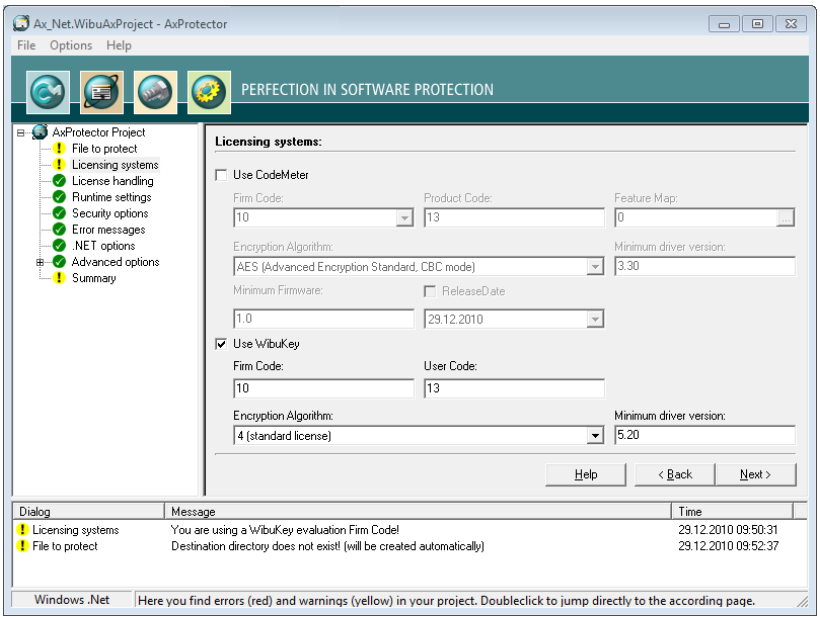


Figure 23: *AxProtector* - .Net "Licensing systems"

If you are switching from *WibuKey* to *CodeMeter*, please activate both hardware platforms.
In this way, you are able to ship updates and upgrades to existing customers who already have a *WibuBox* without the need to replace the hardware. New end-users will be the ones to receive a *CmStick* together with the protected application.

You may also choose to protect using *CodeMeterAct*, the software-based copy protection system. For more information on *CodeMeterAct* visit the Wibu-Systems homepage.

For *WibuKey* the following settings are available:

Firm Code

Specify the FIRM CODE to be used for encrypting the software.

The FIRM CODE 10 used in the figure above is the Evaluation-FIRM CODE found in the *WibuKey* Software Development Kits (SDK). In real life you would not use a FIRM CODE of 10, since this would be insecure. As a registered Licensor, you will be issued your own unique FIRM CODE.

UserCode

Enter the USER CODE which defines the encryption of a specific product. You can freely choose this identifier, e.g. for a separate module of a software application, or for a single application.

Encryption Algorithm

Select the algorithm to encrypt your software. By default, *WibuKey* currently supports Algorithm 4. Algorithm 5 is for reduced licenses. Use Algorithms 1 to 3 for downward compatibility only.

Minimum Driver Version

Enter the minimum driver version required for the installed *WibuKey* driver.

Setting the driver version is also required when, for example, you wish to use new features for the encryption of an application. Older driver versions will not support these new features, and will trigger error messages when starting the protected software.

3.3 License Handling

This input window lets you to define whether the protected application is to search for existing licenses locally in *WibuBoxes*, in the network, or both. Moreover, you can define the license allocation mode.

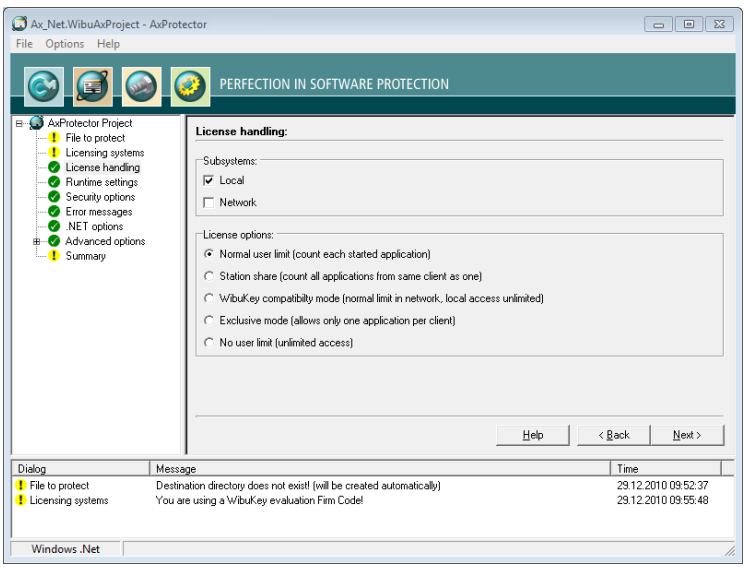


Figure 24: *AxProtector* - .Net "License Handling"

Subsystems

Here you can define in which subsystem (local or network) the protected application is to search for matching license(s).

Local

This setting determines if the protected application searches exclusively for licenses located on the same PC, or allocated to the same VMware.

Network

This setting determines that the license of the protected applications is to be sought in the network, i.e. only PCs are accessed where *WibuKey Server* runs.

On selecting both subsystems at the same time, the license is first sought locally and then subsequently on the network.

License Options

In this group you define how started instances of the protected applications perform, together with the allocation of licenses.

Normal user limit

Here each started instance allocates a single license. It does not make a difference if the *WibuBox* was found locally, or on a network.

Station Share

Here multiple instances can be started on a single PC. But, allocate only a single license.

You use this setting, for example, when you want to provide the end-user with the option of starting the application several times. On a terminal server each session allocates a license. In virtual machines each machine allocates a license.

WibuKey Compatibility Mode

Here each started instance in the network allocates a license (normal user limit) but the local access is unlimited (no user limit).

This allocation option exists only because of compatibility issues with *WibuKey*. Wibu-Systems recommends the setting 'normal user limit' and 'station share'.

Exclusive Mode

Here a protected application can be started only once on a PC.

No user limit

Here any number of instances of the protected application can be started locally or in a network, and no additional licenses are allocated. Allocated licenses in this mode can be re-used.

3.4 Runtime Settings

This input window lets you define the application's runtime settings, e.g. license checks in *WibuBoxes*, issue warnings, etc.

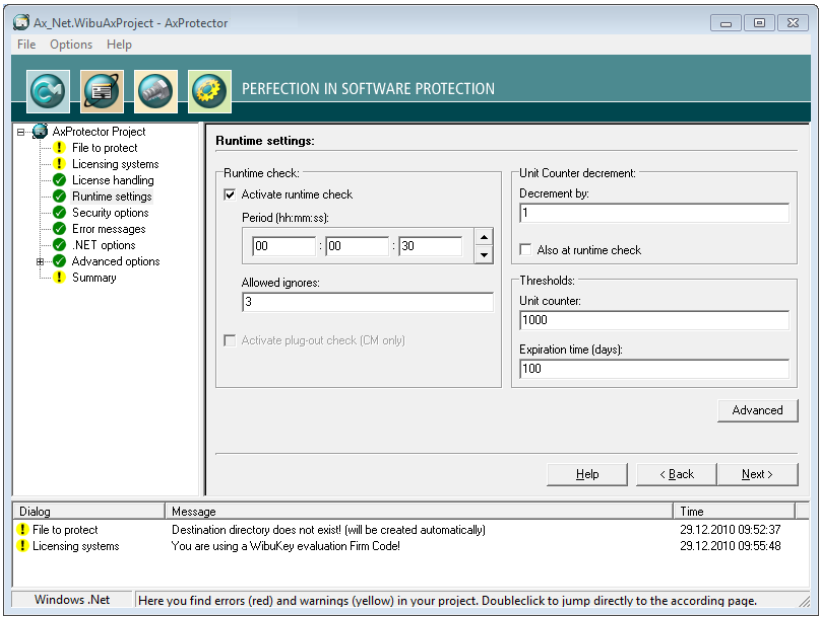


Figure 25: *AxProtector* - .Net "Runtime Settings"

Runtime Check

In this group you define whether and how often the protected application checks the license at runtime.

Activate Runtime Check — Activates or deactivates the check at runtime.

Period — Defines the period between two checks. You specify this time interval in the format: hours: minutes: seconds.

Max. Allowed Ignores — Defines how often the end-user is able to ignore a failed check.

If the connection to a *WibuBox* should fail or the license cannot be accessed, you can assign a reasonable number of “ignores” allowing the end-user to continue working without a license access.

Unit Counter Decrement

Decrementing a *UNIT COUNTER* can serve to establish the validity of licenses in a *WibuBox*. This group allows you to define this behavior.

Decrement by — Defines the value by which the *UNIT COUNTER* is decremented. This option causes a decrement of the counter when the protected application starts.

When the **"Also at Runtime Check"** option is activated and the specifications are set as shown in Figure 25 every 30 seconds (see the defined period) a set UNIT COUNTER is decremented by a value of 1.

Also at Runtime Check

Decrements the UNIT COUNTER also at runtime of the protected application.

This option works only when the **"Also at Runtime Check"** option in the **Runtime Check** group is activated.

Thresholds

In this group you define when a message is issued to give information on the validity of a license.

Unit Counter

Where the defined threshold falls short, a warning message is issued.

Expiration Time (days)

When the specified EXPIRATION TIME (in days) is achieved within the defined threshold, a warning message is issued.

3.5 Advanced Runtime Settings

This input window lets you define further settings at the runtime of an encrypted application.

With the exception of the UNIT COUNTER and EXPIRATION TIME check at runtime and the "Advanced option" area all other settings are valid only for the licensing systems *CodeMeter* or *CodeMeterAct*.

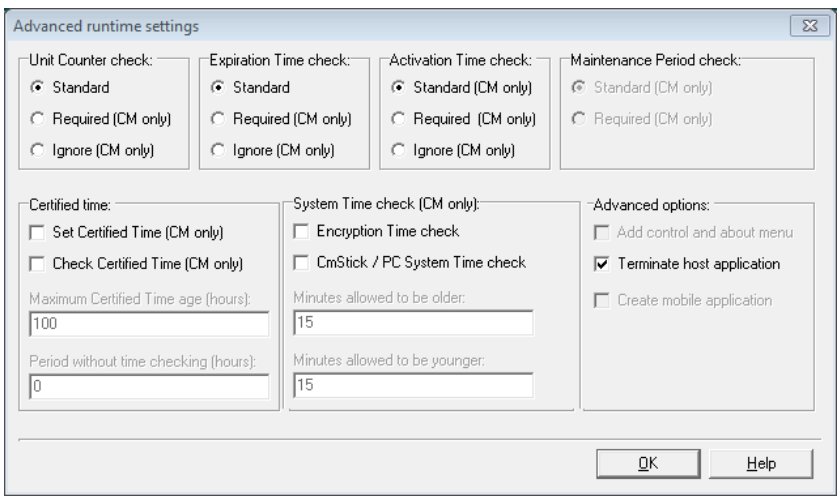


Figure 26: *AxProtector* - .Net "Advanced Runtime Settings"

Unit Counter Check

Defines the handling of a *UNIT COUNTER* when set in a license.

Standard

Decrements at runtime and/or start time an existing *UNIT COUNTER* entry in a license by the value defined on the previous page. When the *UNIT COUNTER* reaches 0 (null) the encrypted application does not start.

Expiration Time Check

Defines the handling of an *EXPIRATION TIME* set in a license.

Standard

Checks for an existing *EXPIRATION TIME* entry in a license. However, the application also starts when no *EXPIRATION TIME* entry exists, or the current date precedes the *EXPIRATION TIME*.

Advanced Options

Terminate Host Application

When no valid license is found, in the case of protected DLL application files the calling *.exe is terminated.

3.6 Security Options

This input window lets you select from different mechanisms and methods for protecting your application. You are able to search for debugger or lock a *WibuBox*.

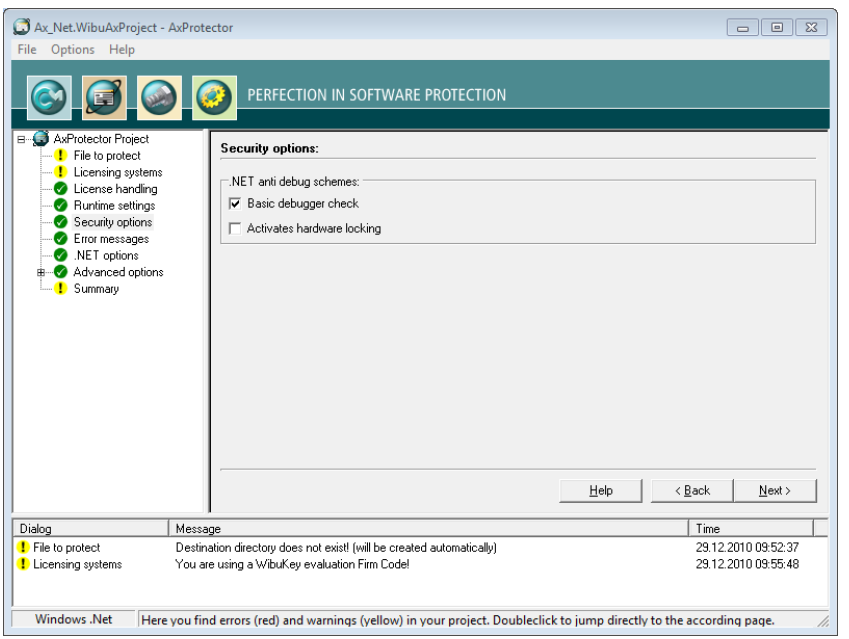


Figure 27: *AxProtector* - .Net "Security Options"

Debugger programs serve an honest role in searching for error and finding bugs. But they may also be used by hackers to analyze software. In this group you determine how to react to debugger programs.

Basic Debugger Check

The 'Basic Debugger Check', checks to see if a debugger is attached to your application. When a debugger is found, your application will not be started or exited.

Activates Hardware Locking

This option locks the used *WibuBox* as soon as a debugger program is detected. In this case the LIMIT COUNTER is set to a value of `0`.

Wibu-Systems recommends to deactivate this option when using the *WibuKey* feature LIMIT COUNTER. This because the *WibuBox* will be locked when the LIMIT COUNTER reaches a value of `0`.

The owner / end-user of the locked *WibuBoxes* must contact the software vendor for unlocking codes. How and how often unlocking is granted depends on the vendor's policy.

Subsequently, the developer is able to unlock the *WibuBox* via remote programming.

3.7 Error Messages

This input window lets you define the messages displayed if errors occur. You define whether a user message DLL with a separate error display is used, or whether you use default error message windows.

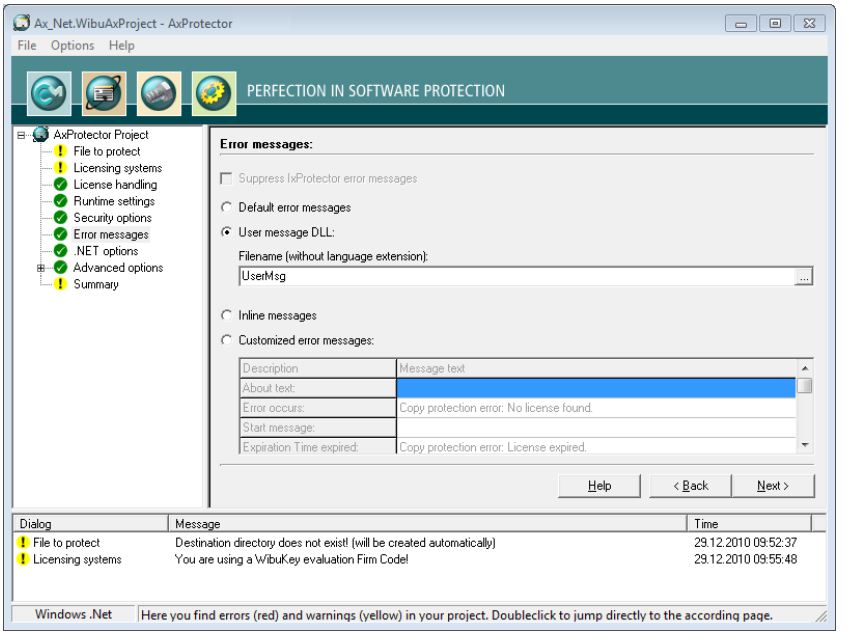


Figure 28: *AxProtector* - .Net "Error Messages"

Default Error Messages

All errors occurring at the runtime of a protected application display default error messages.

User Message DLL

The ability to use the `User Message DLL` is activated. Error messages can be localized to different languages using `*.ini` files. In addition, you have the option to integrate your own designs to this file, for example, by using separate logos or text.

The `*.ini` files with the respective country suffix and the `Dll` program library are automatically saved to the directory where the application locates the files protected by *AxProtector*.

Enter the file name without specifying path and language file extension.

The `UserMsgDll` is copied from the directory `%Program Files%\WIBU-SYSTEMS\AxProtector\DevKit\bin\UserMessage`. The corresponding `*.ini` files are also saved to this directory.

Inline Messages

Links for .NET projects, with an inline assembly, can also be configured by `*.ini` files.

When using Inline UserMessages the logging is saved to the directory "`%CommonApplicationData%`". When you want to

specify another path specify the parameter LogPath=<path> in the `*.ini` file.

Activate this option to enter customized error messages displayed in the message boxes below.

Customized Error Messages

3.8 .NET Options

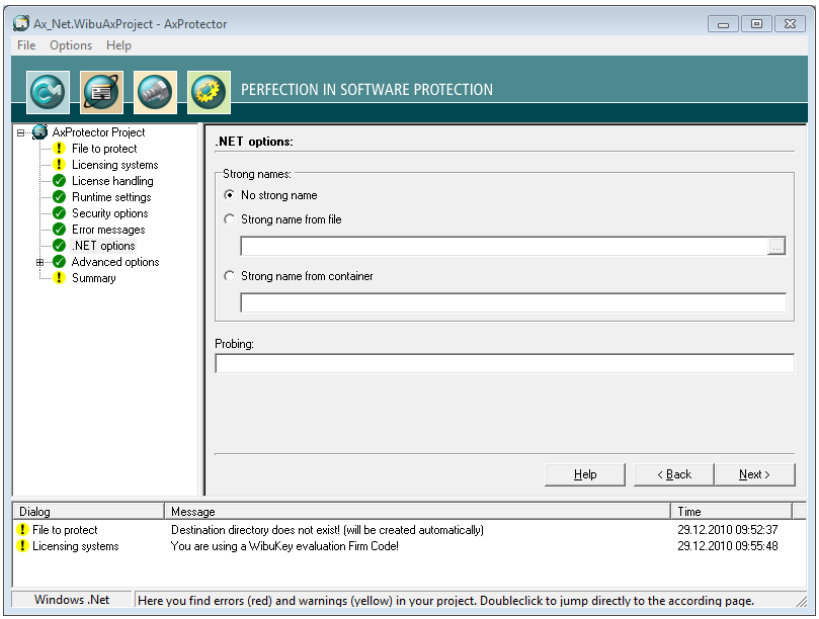


Figure 29: *AxProtector* - .Net ".NET Options"

Here you are able to specify whether your assembly is signed by *AxProtector*.

Activate this checkbox to not sign your assembly.

No Strong Name

Activate this checkbox to use a source file to sign the program class. Then specify a file holding the key pair to generate a strong name.

Strong Name from File

Activate this checkbox to use a container file to sign the program class.

Strong Name from Container

The **Probing** group allows you to specify the location of signed program classes in an `app.config` file.

Specify the path to which the access to the program class refers separated by ";".

Alternatively, specify the respective `app.config` file.

3.9 Advanced Options

This input window lets you set further options for the encryption.

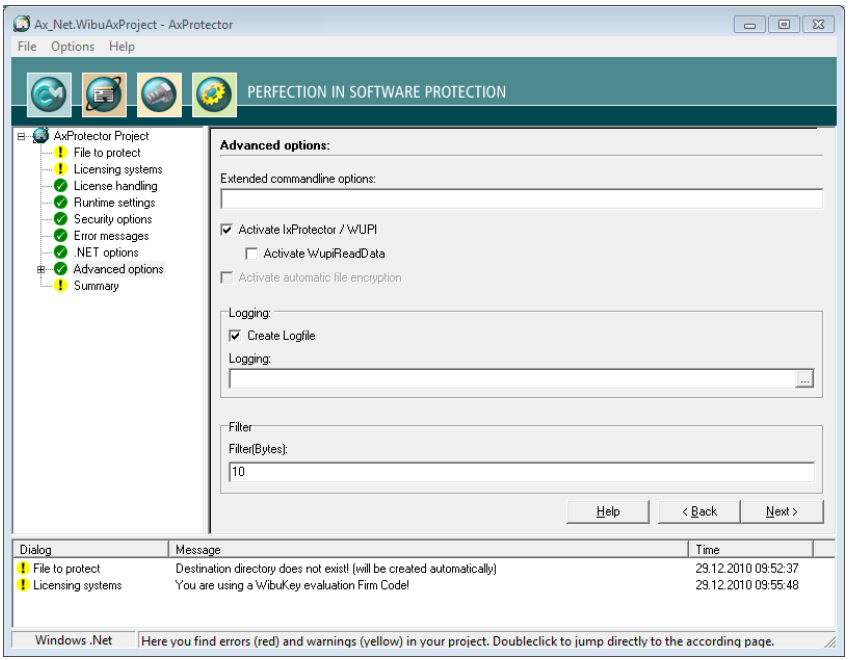


Figure 30: *AxProtector* - .Net "Advanced Options"

Here you are able to enter the xml file [`@name.xml`] holding the encryption information, i.e. NameSpace, Class, Method.

For more information, please contact support at Wibu-Systems.

Activate IxProtector

Activate this checkbox to allow for the later creation and editing of license lists and function lists. These you need to protect using *IxProtector* via the *Software Protection API (WUPI)* (see page 172).

Activate WupiReadData

When activated this checkbox results in reading data from the *WibuBox* which has been previously stored at a defined location (see page 195).

Filter (Bytes)

For an optimized performance specify here the minimum size for assemblies to be encrypted. The default setting is 10 bytes. This way you are able to exclude

methods from encryption which are smaller than the number of bytes you specify here. By setting a value of 0 this feature is deactivated.

Create Logfile

Activate this checkbox to create file logging for the activities of *AxProtector*.

Logging

Specify the path and file name of this log file.

If you specify the name of the file only, by default, this file is saved to the directory `%\Program Files\WIBU-SYSTEMS\AxProtector\DevKit\bin`.

3.9.1 License Lists

This menu item lets you manage license lists. Those you need to protect using *IxProtector* via the *Software Protection API (WUPI)* (see page 196) .

License lists consist of a unique identifier (**ID**), a **Description** and hold specifications on **Items** and **Item Details**.

This **ID** corresponds to the index number you require when addressing a license using most of the WUPI commands (see page 196).

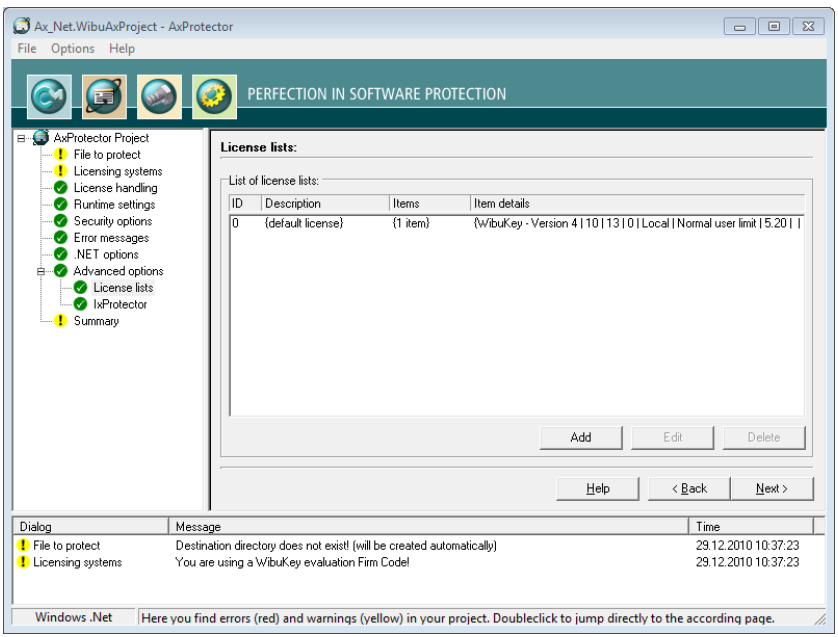


Figure 31: *AxProtector for .NET*- Windows License Lists

3.9.2 Add License Lists

This menu item lets you create license lists. Please proceed as follows:

❶ Click the **"Add"** button.

❷ Assign an **ID** in the *LICENSE LIST* group and complete the **Description** field.

Id

This ID uniquely identifies a license list and serves for referencing.

By default, an **ID** of 0 is initially set by the selection of the copy protection system. Following, you are able to add license list entries starting with an **ID** of 1.

Description

Here you will describe a license list with text.

❸ Define the license by completing the fields in the *LICENSE ITEM DETAILS* group.

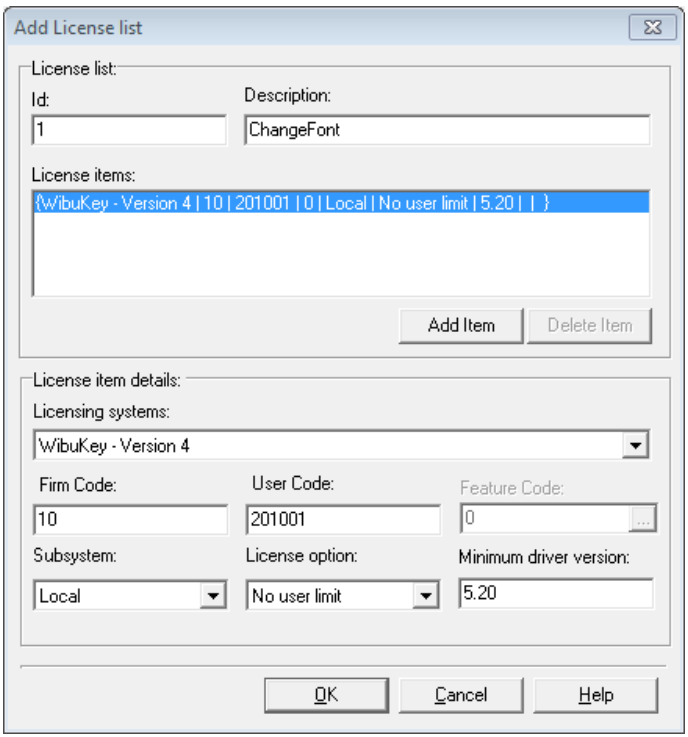


Figure 32: *AxProtector for .NET* Add License Lists

Select the copy protection system used for protection of the license (*CodeMeter, CodeMeterAct* or *WibuKey*) .

Copy Protection System

Enter the FIRM CODE used for the protection of the license.

Firm Code

Enter the PRODUCT CODE used for the protection of the license.

User Code

Select the subsystem in which the protected application is to search (local or network), and define the search order.

Subsystem

Select the options for license allocation:

License Option

- *NORMAL USER LIMIT*
- *STATION SHARE*
- *WK COMPATIBILITY MODE*
- *EXCLUSIVE MODE*
- *NO USER LIMIT*

Specify the required minimum driver version for the protected application.

Minimum Driver Version

❸ Click on the **"Add"** button in the *LICENSE LIST* group.
The summary of your specifications are displayed in the license item list.

❹ Click the **"OK"** button.
The new license data is added to the license list.

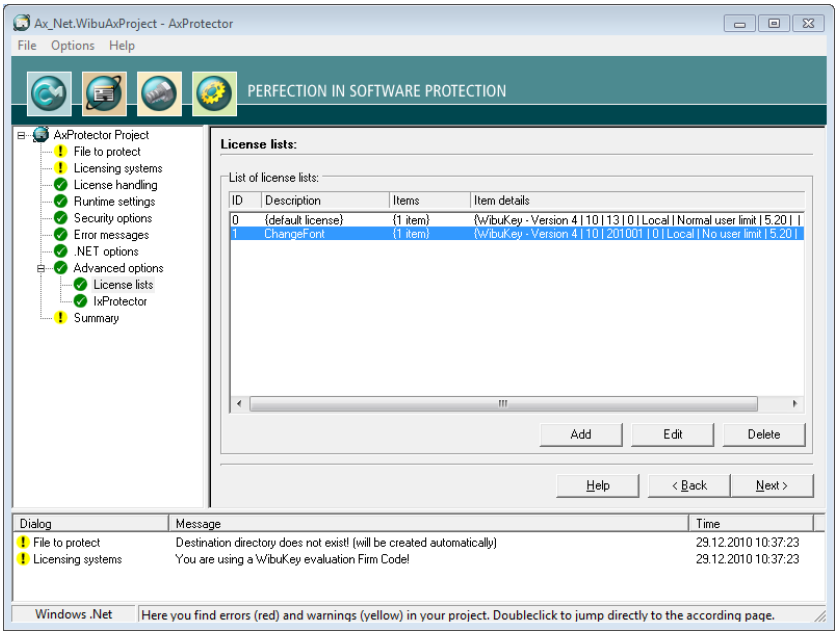
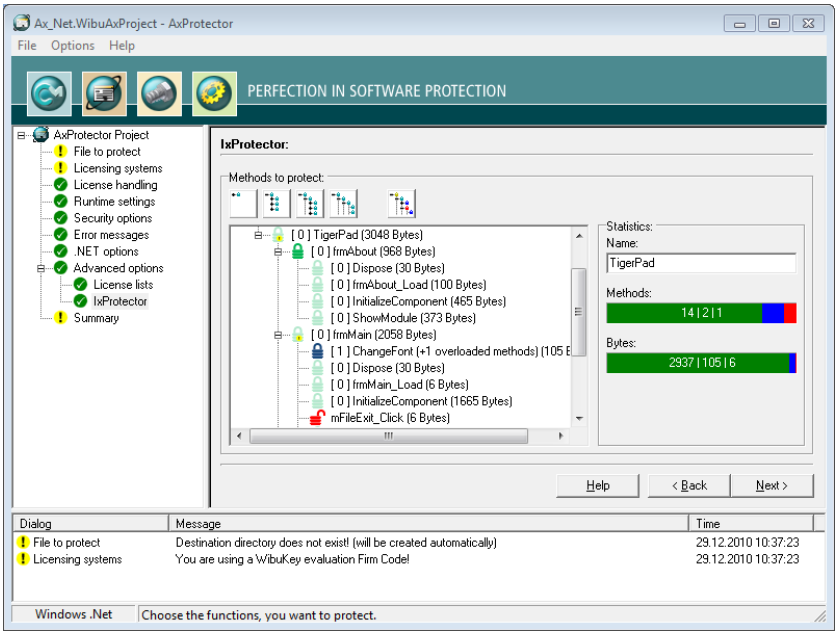


Figure 33: *AxProtector for .NET* – Completed License List

3.9.3 *IxProtector*

Using this menu item allows you to separately define single encryption types for single assembly elements (see also page 172) .

In the case you activated the checkbox "IxProtector" in the menu item "Advanced options" the source assembly is loaded and displayed in a tree view making available all namespaces, classes, and modules.



Click the different buttons in the upper *IxProtector* area to select from different assembly views.

Views

Button	Description
	Closes all assembly levels of the tree structure.
	Expands the namespace level of the assembly.
	Expands the class level of the assembly.
	Expands the method level of the assembly.
	Expands all parent levels of the assembly.  In this view see all levels where modifications have been made.

Statistics

The area Statistics on the right shows you more encryption details depending on the selection you have made for the tree view.

Name

This field refers to the name of the element you have marked in the tree view.

The bars 'Methods' and 'Bytes' provide you a combined numerical and visual overview on encryption details. This allows you to see the effects your settings will have when protecting your application.

Methods

Using different colors the bar 'Methods' shows you the protection technology used or not used when encrypting or not encrypting. At the same time, the displayed numbers inform you about the number of encrypted or non-encrypted methods for each protection technology.

Color	Description
Green	Shows that the method will be encrypted using *AxProtector* and that the *LICENSELIST ID* has a value of `0` (default license).
Blue	Shows that the method will be encrypted using *IxProtector* and that the *LICENSELIST ID* has a value unequal `0`.
Red	Shows that the method in not encrypted.

Bytes

Using different colors the bar 'Bytes' also shows you the protection technology used or not used when encrypting or not encrypting. At the same time, the displayed numbers inform you about the number of encrypted or non-encrypted bytes for each protection technology.

Color	Description
Green	Shows that the method will be encrypted using *AxProtector* and that the *LICENSELIST ID* has a value of `0` (default license).
Blue	Shows that the method will be encrypted using *IxProtector* and that the *LICENSELIST ID* has a value unequal `0`.
Red	Shows that the method in not encrypted.

Modifying Protection Technology

You also have the option to separately assign the protection technologies *AxProtector* and *IxProtector* to single assembly elements, or exclude single elements from encrypting. To assign a protection technology by using the secondary menu, please proceed as follows:

❶ In the left tree view, select the favored assembly element (namespace, class, or method).

❷ Click the right mouse button.
The secondary menu opens.

❸ Assign the favored encryption types by using symbols.

The *LICENSELIST ID*‘s you are prompted are automatically transferred from the entries you added to the license list.

Symbol	Description
	Excludes the selected element from encryption.
	Encrypts the selected element using *AxProtector* (*LICENSELIST ID* with a value of `0`, i.e. default license).
	Encrypts the selected element using *IxProtector* (*LICENSELIST ID* with a value unequal to `0`, i.e. according to existing license list entries).

The modifications you made instantly display in the left area.

3.10 Summary

This input window shows you a summary of all the settings you defined for the automatic protection of your application, and allows you to start the encryption process.

For subsequent use, the contents of this page can be copied to a `*.wbc` file (WIBU Configuration file). Copy the content into a text file, and change the file extension to `*.wbc`.

Alternatively, you may also use this file to protect your application using the *AxProtector* commandline tool. In the commandline type `AxProtector.exe @*.wbc` (see page 189).

Alternatively, using the **"File | export wbc-file"** menu item, you can also create the corresponding `*.wbc` file.

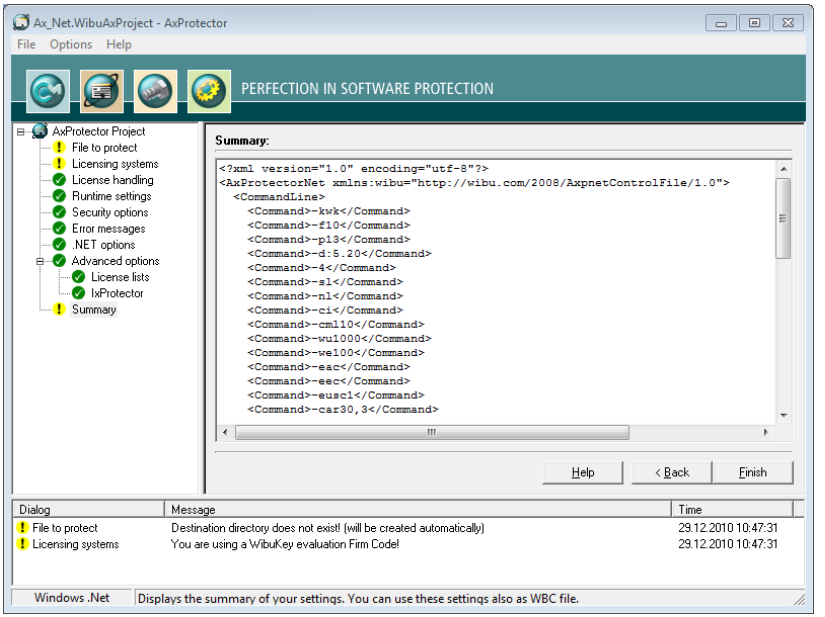


Figure 34: *AxProtector* - .Net "Summary"

Finish

Starts the encryption using *AxProtector* applying the settings you previously defined.

3.11 Protection Result

The result of the encryption with all relevant settings is displayed in a separate window.

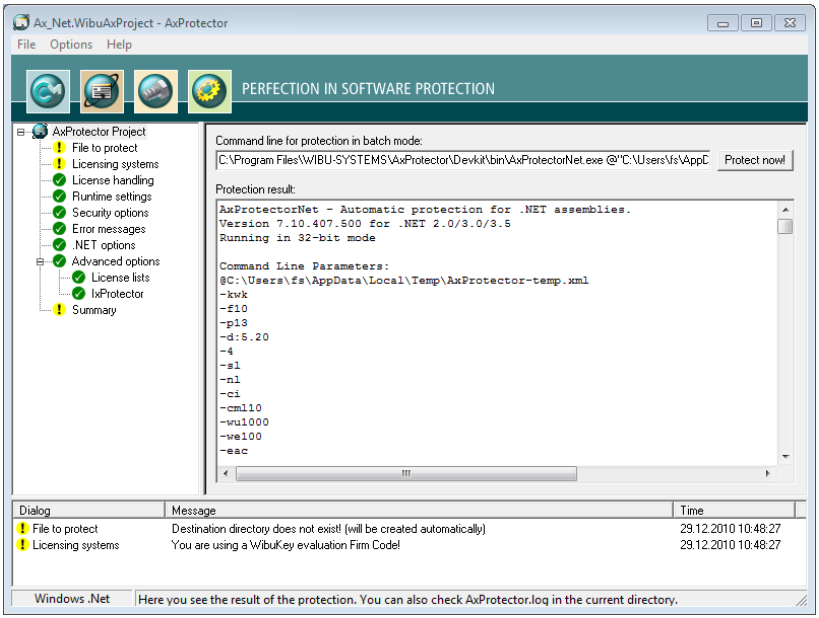


Figure 35: *AxProtector* - .Net "Protection Result"

When you need to repeat the encryption operation, click the **"Protect Now"** button. Then the *AxProtector* commandline is executed in batch mode.

You are also able to copy the *AxProtector* commandline for the batch mode to the clipboard and insert it in the commandline input. Subsequently, you can edit it and apply any desired changes.

4 Mac OS X Applications

For this project type applications to be encrypted comprise Mac OS X applications.

4.1 File to protect

To safely encrypt an executable file using *AxProtector*, first select the file you want to protect.

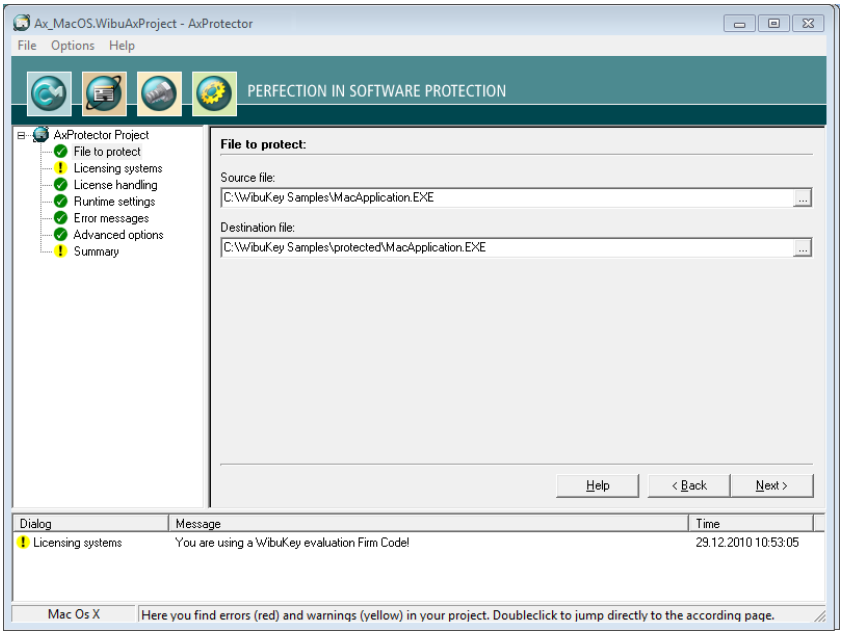


Figure 36: *AxProtector* - Mac OS X "File to protect"

Source File

Click on the **"..."** button and select the file to protect using the **"Open"** system dialog. Alternatively, manually specify the path and name of the file in this field.

As alternative to the **"..."** button, you may also directly drag & drop the source file from Windows Explorer into the source file field.

Destination File

After you selected the source file, *AxProtector* automatically creates a subfolder `[..\protected\..]`. You may change this default by manually specifying the path and name of the destination file. Then the destination file corresponds to your protected application.

4.2 Licensing Systems

After you select the file to be protected, the "Licensing systems" fields will appear in the input window. This is where you can select which protection schemes will be used. Depending on your requirements, you can select one or all of the check boxes.

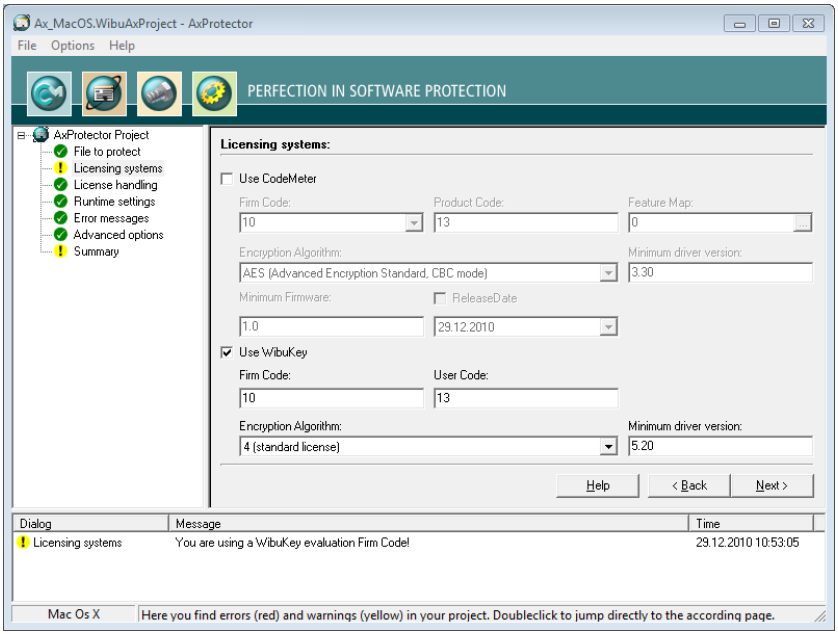


Figure 37: *AxProtector* - Mac OS X "Licensing systems"

If you are switching from *WibuKey* to *CodeMeter*, please activate both hardware platforms.
In this way, you are able to ship updates and upgrades to existing customers who already have a *WibuBox* without the need to replace the hardware. New end-users will be the ones to receive a *CmStick* together with the protected application.

You may also choose to protect using *CodeMeterAct*, the software-based copy protection system. For more information on *CodeMeterAct* visit the Wibu-Systems homepage.

For *WibuKey* the following settings are available:

Firm Code

Specify the FIRM CODE to be used for encrypting the software.

The FIRM CODE 10 used in the figure above is the Evaluation-FIRM CODE found in the *WibuKey* Software Development Kits (SDK).
In real life you would not use a FIRM CODE of 10, since this would be insecure. As a registered Licensor, you will be issued your own unique FIRM CODE.

User Code

Enter the USER CODE which defines the encryption of a specific product. You can freely choose this identifier, e.g. for a separate module of a software application, or for a single application.

Encryption Algorithm

Select the algorithm to encrypt your software. By default, *WibuKey* currently supports Algorithm 4. Algorithm 5 is for reduced licenses. Use Algorithms 1 to 3 for downward compatibility only.

Minimum Driver Version

Enter the minimum driver version required for the installed *CodeMeter* driver.

When setting the minimum driver version to 5.20 the session handling for terminal servers is automated. This means that *AxProtector* automatically handles sessions of the protected software, and each session is allocated one of the available licenses.

Setting the driver version is also required when, for example, you wish to use new features for the encryption of an application. Older driver versions will not support these new features, and will trigger error messages when starting the protected software.

For setting *CodeMeter and CodeMeterAct* options, see the separate *CodeMeter* Developer Guide.

4.3 License Handling

This input window lets you to define whether the protected application is to search for existing licenses locally in *WibuBoxes*, in the network, or both. Moreover, you can define the license allocation mode.

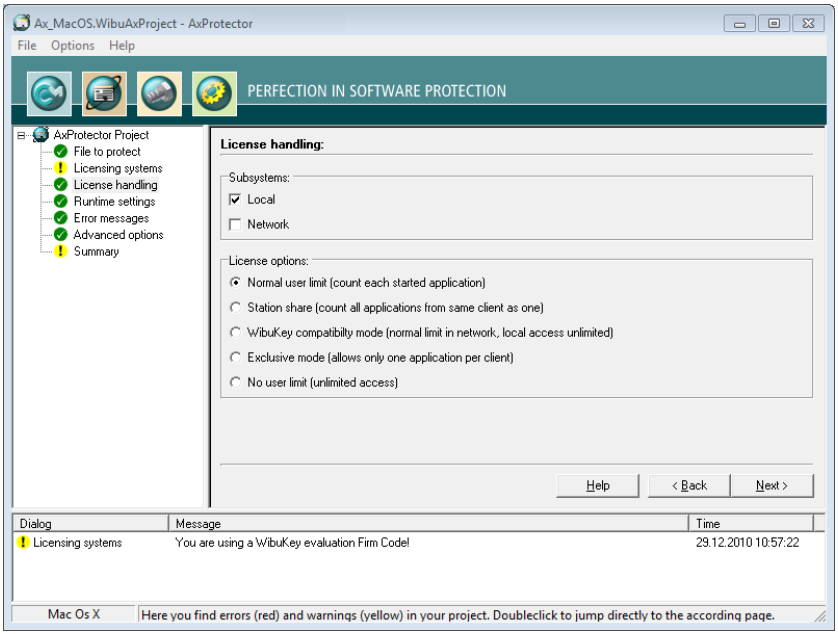


Figure 38: *AxProtector* - Mac OS X "License Handling"

Subsystems

Here you can define in which subsystem (local or network) the protected application is to search for matching license(s).

Local

This setting determines if the protected application searches exclusively for licenses located on the same PC, or allocated to the same VMware.

Network

This setting determines that the license of the protected applications is to be sought in the network, i.e. only PCs are accessed where *WibuKey Server* runs.

On selecting both subsystems at the same time, the license is first sought locally and then subsequently on the network.

License Options

In this group you define how started instances of the protected applications perform, together with the allocation of licenses.

Normal user limit

Here each started instance allocates a single license. It does not make a difference if the *WibuBox* was found locally, or on a network.

Station Share

Here multiple instances can be started on a single PC. But, allocate only a single license.

You use this setting, for example, when you want to provide the end-user with the option of starting the application several times. On a terminal server each session allocates a license. In virtual machines each machine allocates a license.

WibuKey Compatibility Mode

Here each started instance in the network allocates a license (normal user limit) but the local access is unlimited (no user limit).

This allocation option exists only because of compatibility issues with *WibuKey*. Wibu-Systems recommends the setting 'normal user limit' and 'station share'.

Exclusive Mode

Here a protected application can be started only once on a PC.

No user limit

Here any number of instances of the protected application can be started locally or in a network, and no additional licenses are allocated. Allocated licenses in this mode can be re-used.

4.4 Runtime Settings

This input window lets you define the application's runtime settings, e.g. license checks in *WibuBoxes*, issue warnings, etc.

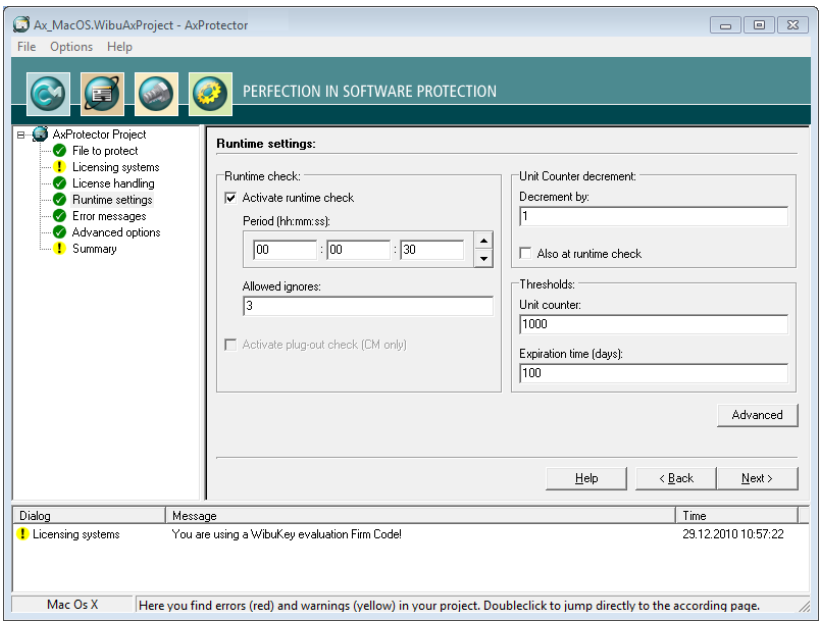


Figure 39: *AxProtector* - Mac OS X "Runtime Settings"

Runtime Check

In this group you define whether and how often the protected application checks the license at runtime.

Activate Runtime Check

Activates or deactivates the check at runtime.

Period

Defines the period between two checks. You specify this time interval in the format: hours: minutes: seconds.

Max. Allowed Ignores

Defines how often the end-user is able to ignore a failed check.

If the connection to a *WibuBox* should fail or the license cannot be accessed, you can assign a reasonable number of 'ignores' allowing the end-user to continue working without a license access.

Unit Counter Decrement

Decrementing a *UNIT COUNTER* can serve to establish the validity of licenses in a *WibuBox*. This group allows you to define this behavior.

Decrement by

Defines the value by which the *UNIT COUNTER* is decremented. This option causes a decrement of the counter when the protected application starts.

When the **"Also at Runtime Check"** option is activated and the specifications are set as shown in Figure 39 every 30 seconds (see the defined period) a set UNIT COUNTER is decremented by a value of 1.

Also at Runtime Check

Decrements the UNIT COUNTER also at runtime of the protected application.

This option works only when the **"Also at Runtime Check"** option in the **Runtime Check** group is activated.

Thresholds

In this group you define when a message is issued to give information on the validity of a license.

Unit Counter

Where the defined threshold falls short, a warning message is issued.

Expiration Time (days)

When the specified EXPIRATION TIME (in days) is achieved within the defined threshold, a warning message is issued.

4.5 Advanced Runtime Settings

This input window lets you define further settings at the runtime of an encrypted application.

With the exception of the UNIT COUNTER and EXPIRATION TIME check at runtime all other settings are valid only for the licensing systems *CodeMeter* or *CodeMeterAct*.

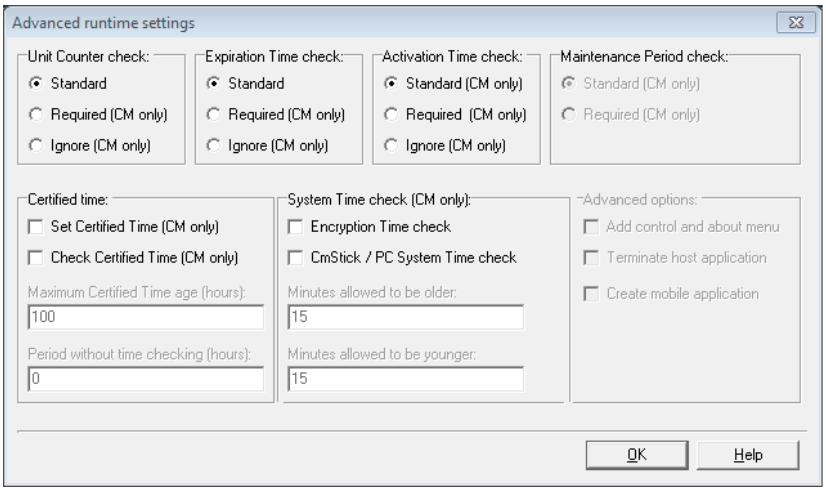


Figure 40: *AxProtector* - Windows "Advanced Runtime Settings"

Unit Counter Check

Defines the handling of a *UNIT COUNTER* when set in a license.

Standard

Decrements at runtime and/or start time an existing *UNIT COUNTER* entry in a license by the value defined on the previous page. When the *UNIT COUNTER* reaches 0 (null) the encrypted application does not start.

Expiration Time Check

Defines the handling of an *EXPIRATION TIME* set in a license.

Standard

Checks for an existing *EXPIRATION TIME* entry in a license. However, the application also starts when no *EXPIRATION TIME* entry exists, or the current date precedes the *EXPIRATION TIME*.

4.6 Error Messages

This input window lets you define the messages displayed if errors occur. You define whether a user message DLL with a separate error display is used, or whether you use default error message windows.

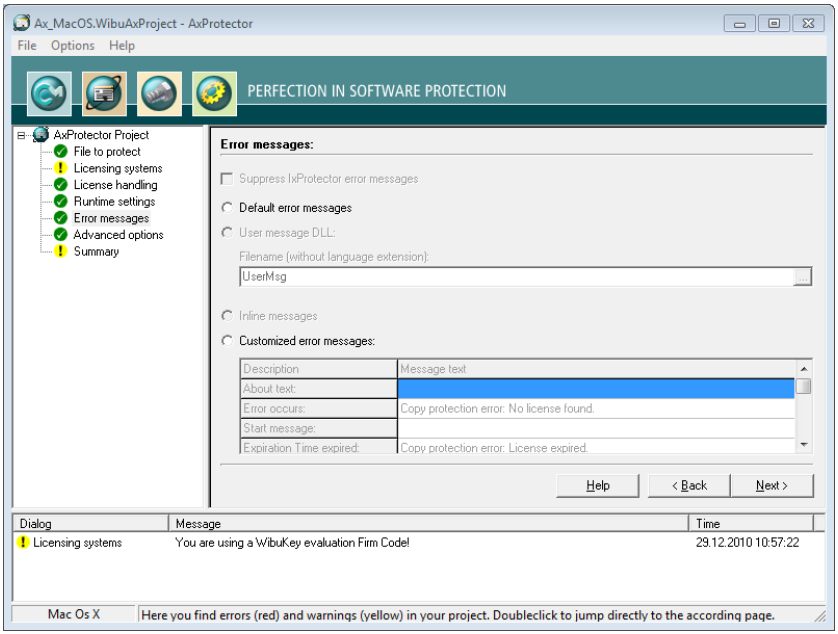


Figure 41: *AxProtector* - Mac OS X "Error Messages"

Default Error Messages

All errors occurring at the runtime of a protected application display default error messages.

Customized Error Messages

Activate this option to enter customized error messages displayed in the message boxes below.

4.7 Advanced Options

This input window lets you set further options for the encryption.

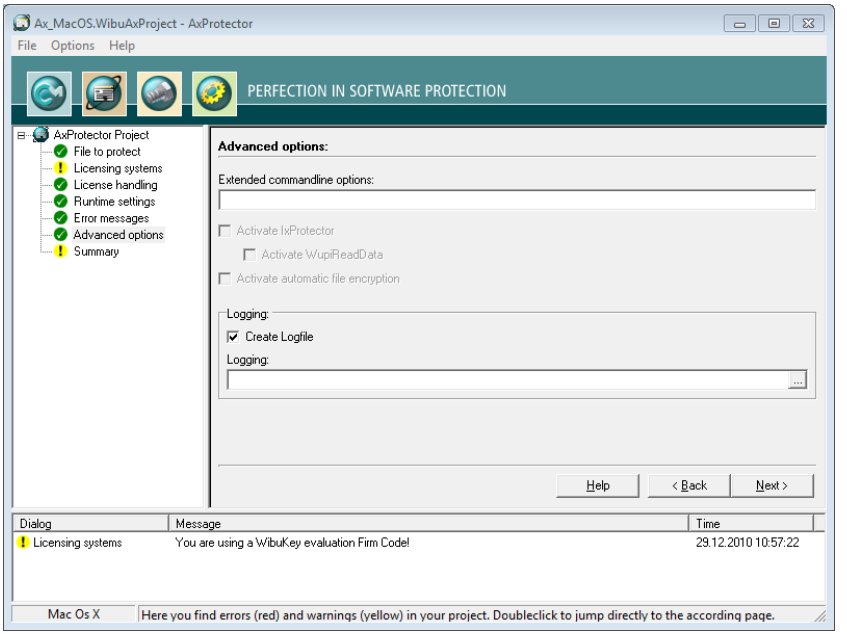


Figure 42: *AxProtector* - Mac OS X "Advanced Options"

Extended Commandline Options

Here you are able to directly enter extended options or new feature functions using *AxProtector* commandline parameters.

For more information, please contact support at Wibu-Systems.

Create Logfile

Activate this checkbox to create file logging for the activities of *AxProtector*.

Logging

Specify the path and file name of this log file.

If you specify the name of the file only, by default, this file is saved to the directory `%\Program Files\WIBU-SYSTEMS\AxProtector\DevKit\bin`.

4.8 Summary

This input window shows you a summary of all the settings you defined for the automatic protection of your application, and allows you to start the encryption process.

For subsequent use, the contents of this page can be copied to a `*.wbc` file (WIBU Configuration file). Copy the content into a text file, and change the file extension to `*.wbc`.

Alternatively, you may also use this file to protect your application using the *AxProtector* commandline tool. In the commandline type `AxProtector.exe @*.wbc` (see page 189).

Alternatively, using the **"File | export wbc-file"** menu item, you can also create the corresponding `*.wbc` file.

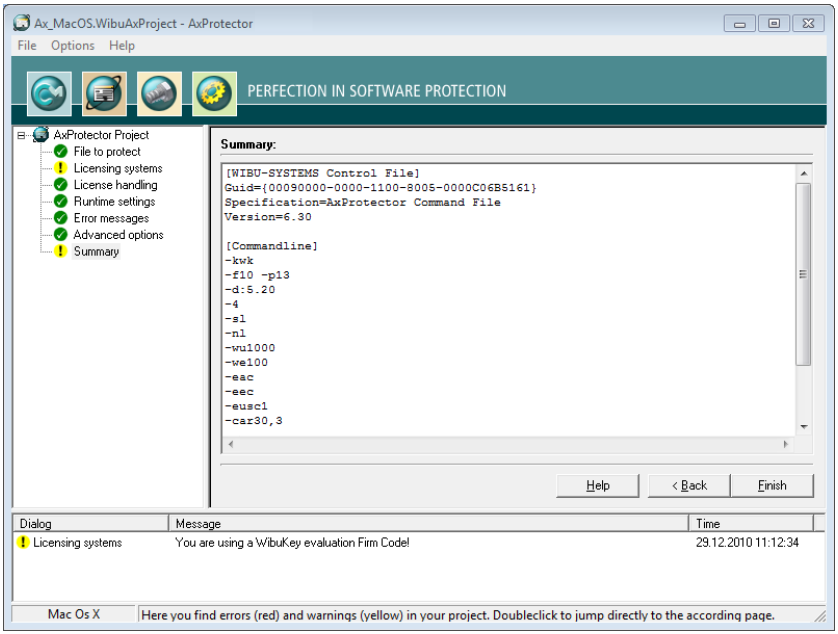


Figure 43: *AxProtector* - Mac OS X "Summary"

Finish

Starts the encryption using *AxProtector* applying the settings you previously defined.

4.9 Protection Result

The result of the encryption with all relevant settings is displayed in a separate window.

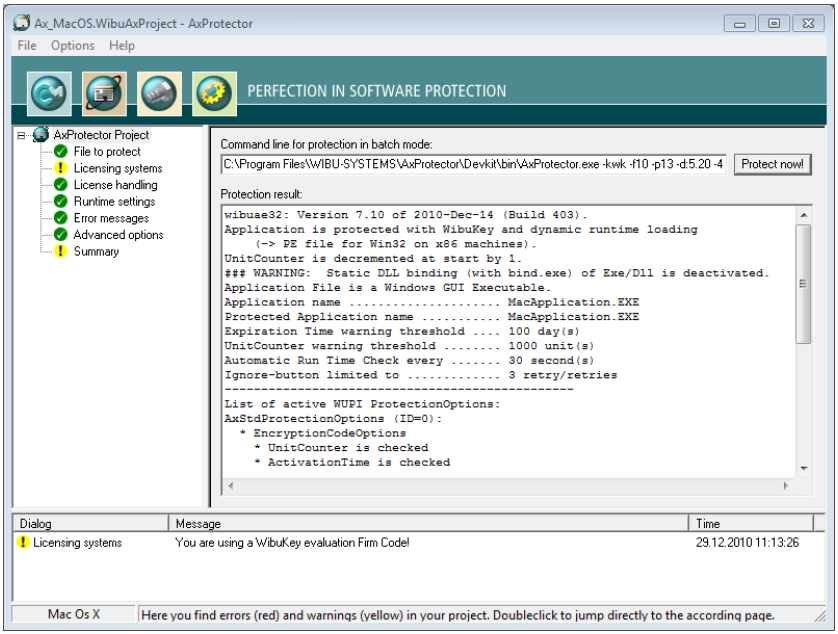


Figure 44: *AxProtector* - Mac OS X "Protection Result"

Protect Now

When you need to repeat the encryption operation, click the **"Protect Now"** button. Then the *AxProtector* commandline is executed in batch mode.

You are also able to copy the *AxProtector* commandline for the batch mode to the clipboard and insert it in the commandline input. Subsequently, you can edit it and apply any desired changes.

5 Linux Applications

Currently, no graphical user interface for *AxProtector* exists for the automatic protection of Linux applications.

However, you have the option to use *AxProtector* in its commandline variant for 32-bit and 64-bit systems: *AxProtectorLin* (see page 172ff).

Currently encrypting is limited to *executables* in the standard binary format for executable programs (ELF, Executable and Linking Format). The encryption of program libraries (*shared objects* - `*.so`) is currently not supported.

For an overview of encrypting options currently or not yet supported by *AxProtectorLin* see the following table.

Feature	Description
Copy protection system	*WibuKey*, *CodeMeter*, *CodeMeterAct*
Anti-Debugger detection	All `–cag` options except 8 are supported (see page 176).
Security options	The `–cag` options are currently not supported (see page 175).
Error messages	Support of standard Linux error message dialogs or definition using the option `–m` (see page 184f).
UserMessage	The use of `UserMessage`-functions is not supported.
Static linking	Support as non-changeable default setting (`-x` option) (see page 172).
IxProtector	Is currently not supported.
File encryption	Is currently not supported.

6 Java Applications

Compiled Java code, unlike .NET-code, can be re-translated into uncompiled source code: easily and without any special programming knowledge required. Thus, almost everything what happens in the application is principally publicly available, and competitors are able to easily analyze the software. Your intellectual property is virtually unprotected. In addition, even a built-in license management can be easily removed from the software. Thus, sooner or later for each Java developer the question arises: How to protect intellectual property and to prevent use violations?

The *AxProtector for Java* solves this challenge. Basically, a Java compilation is composed of a mere collection of compiled classes, of class files. Usually, these are bundled, saved, and delivered as `jar`-archives. The basic principle of the *AxProtector for Java* is to separately encrypt each single class. For this purpose, automatically the `jar`-archive is unpacked, each `class` file is encrypted according to the selected settings, and afterward re-packed in the archive together with some necessary class files of Wibu-Systems.

ClassLoader Modification

On start of the application, at first, the SystemClassLoader is loaded: by and by, on demand it reloads the classes required in the application. It is exactly here where *AxProtector* comes in. On encrypting the *AxProtector* also modifies the manifest file resulting in a modified application start. Instead loading the System `ClassLoader` *AxProtector* loads a `ClassLoader` provided by Wibu-Systems

using two helper classes (wrapper / starter). This `WibuClassLoader` manages the loading of the encrypted classes, and not encrypted classes are still loaded by the originally Java-included `ClassLoader`.

Decryption in native Code

Java itself provides a number of different options to interact into loading processes. Thus, decrypting within Java code is not reasonable, and easily nullified. In the *AxProtector for Java* the `WibuClassLoader` passes on the encrypted classes to a native library. In this `wibuXPM4J` library the class is now decrypted according to the selected copy protection system (*CodeMeter* / *CodeMeterAct* / *WibuKey*), and passed on to the native Java library. The decrypted class then is available in Java without any restrictions.

Additional Security Mechanisms

In addition to this loading principle, the *AxProtector for Java* extends the application by other security mechanisms. In order to ensure that the allocated license is still available for further use, and, for example, that the dongle was not disconnected, a periodical check at application runtime can be specified. Then the allocated license is re-checked by decryption operations in customizable intervals, and in the case that an error is returned, the application halts.

Signature Check of the Runtime Environment

Since Version 6, Java sources are open and available. In principle, now anybody is able to assemble a slightly modified version of Java, and able to inward transfer own code into the native Java library to record the loading of decrypted classes. Therefore, in Java 6 the option exists to check the authenticity of the Java version in use. For that purpose, signatures of the native Java libraries are added to the application and checked on start. In the case a newer version of the Java library is used, the *AxProtector* spots this, and offers to automatically download new signatures from the Wibu-Systems website. This way, the application is able to handle not yet released versions at the time of encryption.

Requirements

The *AxProtector for Java* exclusively works with the original Sun Java, i.e. the usually used variant of Java. Along with the files located in the `jar` archive, the user requires the native `wibuXPM4J` library mentioned above. It is included for Windows and Mac in the Runtime Kits of *CodeMeter* and *WibuKey*, for Linux there exist small separate installer.

When encrypting an additional option is provided to include (white list), or to exclude (black list) specific classes. This allows, for example, to exclude classes of other vendors from encryption. Moreover, a minimum version can be specified.

This description so far related to Java applications, i.e. separate programs located on the user's hard drive. However, application scenarios using Java have become varied, and for example, also the protection of server applications becomes an option. For example, how to integrate software protection into the application server Tomcat?

Customized Use

The *AxProtector for Java* also meets protection requirements of, for example, Java Servlets, Eclipse Rich Client applications, or Java Web Start applications. When using *AxProtector for Java* in such environments, you have to note some special requirements, and make customizations. Meanwhile, Wibu-Systems provides several `ClassLoader` especially designed to meet requirements in specific cases, for example, the `ServletClassLoader`, or the `EclipseClassLoader`. Contact Wibu-Systems Support and inquire for matching samples, or support on integration.

6.1 File to Protect

To safely encrypt an executable file using *AxProtector*, first select the file you want to protect.

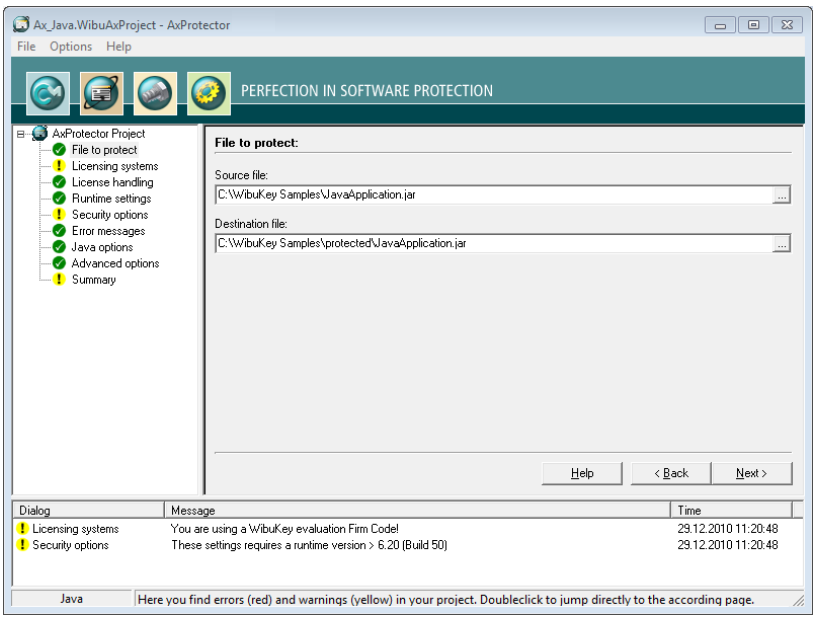


Figure 45: *AxProtector* - Java "File to Protect"

Source File

Click on the **"..."** button and select the file to protect using the **"Open"** system dialog. Alternatively, manually specify the path and name of the file in this field.

As alternative to the **"..."** button, you may also directly drag & drop the source file from Windows Explorer into the source file field.

Destination File

After you selected the source file, *AxProtector* automatically creates a subfolder `[..\protected\..]`. You may change this default by manually specifying the

path and name of the destination file. Then the destination file corresponds to your protected application.

6.2 Licensing Systems

After you select the file to be protected, the "Licensing systems" fields will appear in the input window. This is where you can select which protection schemes will be used. Depending on your requirements, you can select one or all of the check boxes.

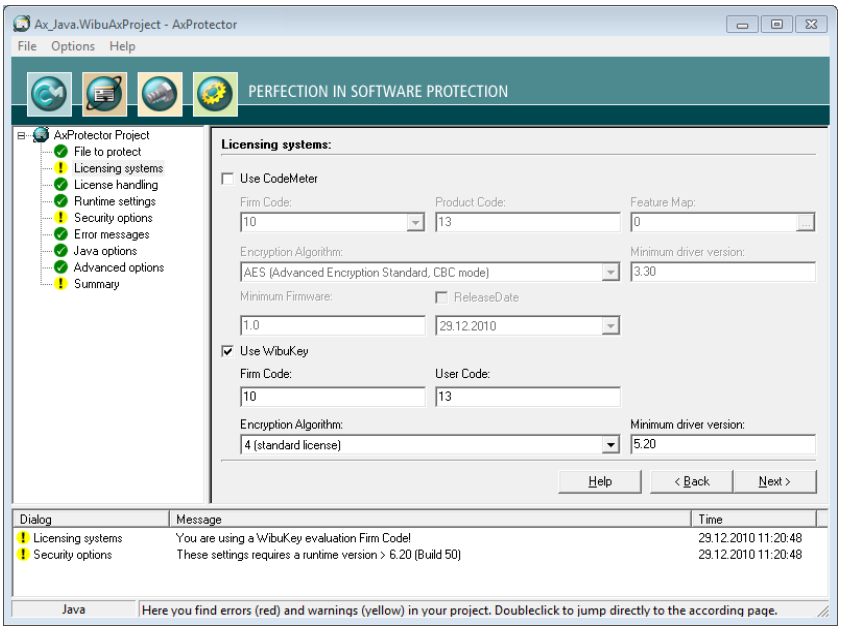


Figure 46: *AxProtector* - Java "Licensing systems"

If you are switching from *WibuKey* to *CodeMeter*, please activate both hardware platforms.
In this way, you are able to ship updates and upgrades to existing customers who already have a *WibuBox* without the need to replace the hardware. New end-users will be the ones to receive a *CmStick* together with the protected application.

You may also choose to protect using *CodeMeterAct*, the software-based copy protection system. For more information on *CodeMeterAct* visit the Wibu-Systems homepage.

For *WibuKey* the following settings are available:

Firm Code

Specify the FIRM CODE to be used for encrypting the software.

The FIRM CODE 10 used in the figure above is the Evaluation-FIRM CODE found in the *WibuKey* Software Development Kits (SDK). In real life you would not use a FIRM CODE of 10, since this would be insecure. As a registered Licensor, you will be issued your own unique FIRM CODE.

User Code

Enter the USER CODE which defines the encryption of a specific product. You can freely choose this identifier, e.g. for a separate module of a software application, or for a single application.

Encryption Algorithm

Select the algorithm to encrypt your software. By default, *WibuKey* currently supports Algorithm 4. Algorithm 5 is for reduced licenses. Use Algorithms 1 to 3 for downward compatibility only.

Minimum Driver Version

Enter the minimum driver version required for the installed *WibuKey* driver.

When setting the minimum driver version to 5.20 the session handling for terminal servers is automated. This means that *AxProtector* automatically handles sessions of the protected software, and each session is allocated one of the available licenses.

Setting the driver version is also required when, for example, you wish to use new features for the encryption of an application. Older driver versions will not support these new features, and will trigger error messages when starting the protected software.

For setting *CodeMeter* and *CodeMeterAct* options, see the separate *CodeMeter* Developer Guide.

6.3 License Handling

This input window lets you to define whether the protected application is to search for existing licenses locally in *WibuBoxes*, in the network, or both. Moreover, you can define the license allocation mode.

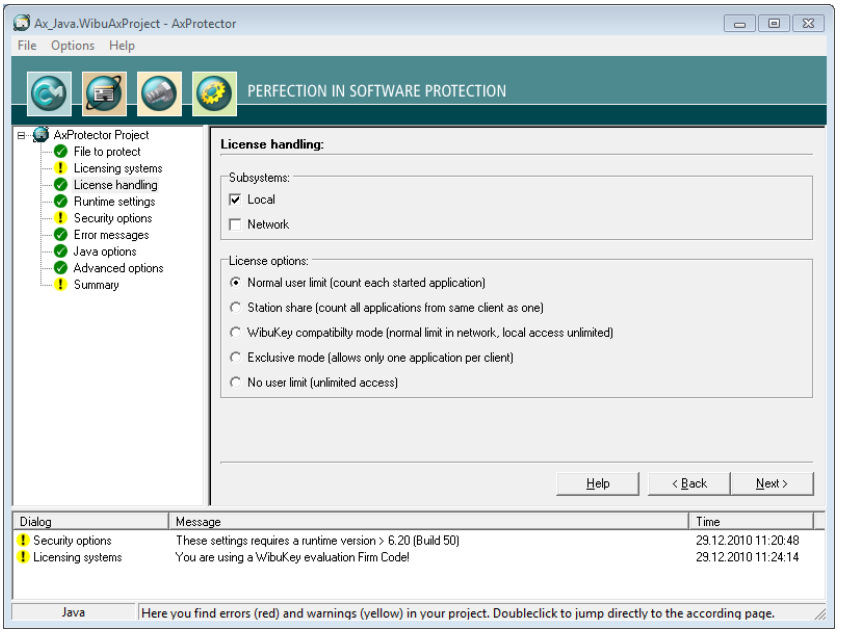


Figure 47: *AxProtector* - Java "License Handling"

Subsystems

Here you can define in which subsystem (local or network) the protected application is to search for matching license(s).

Local

This setting determines if the protected application searches exclusively for licenses located on the same PC, or allocated to the same VMware.

Network

This setting determines that the license of the protected applications is to be sought in the network, i.e. only PCs are accessed where *WibuKey Server* runs.

On selecting both subsystems at the same time, the license is first sought locally and then subsequently on the network.

License Options

In this group you define how started instances of the protected applications perform, together with the allocation of licenses.

Normal user limit

Here each started instance allocates a single license. It does not make a difference if the *WibuBox* was found locally, or on a network.

Station Share

Here multiple instances can be started on a single PC. But, allocate only a single license.

You use this setting, for example, when you want to provide the end-user with the option of starting the application several times. On a terminal server each session allocates a license. In virtual machines each machine allocates a license.

WibuKey Compatibility Mode

Here each started instance in the network allocates a license (normal user limit) but the local access is unlimited (no user limit).

This allocation option exists only because of compatibility issues with *WibuKey*. Wibu-Systems recommends the setting 'normal user limit' and 'station share'.

Exclusive Mode

Here a protected application can be started only once on a PC.
This is the most restrictive mode, i.e. on the local system and on the network client each program copy is interpreted as a license.

No user limit

Here any number of instances of the protected application can be started locally or in a network, and no additional licenses are allocated. Allocated licenses in this mode can be re-used.

6.4 Runtime Settings

This input window lets you define the application's runtime settings, e.g. license checks in *WibuBoxes*, issue warnings, etc.

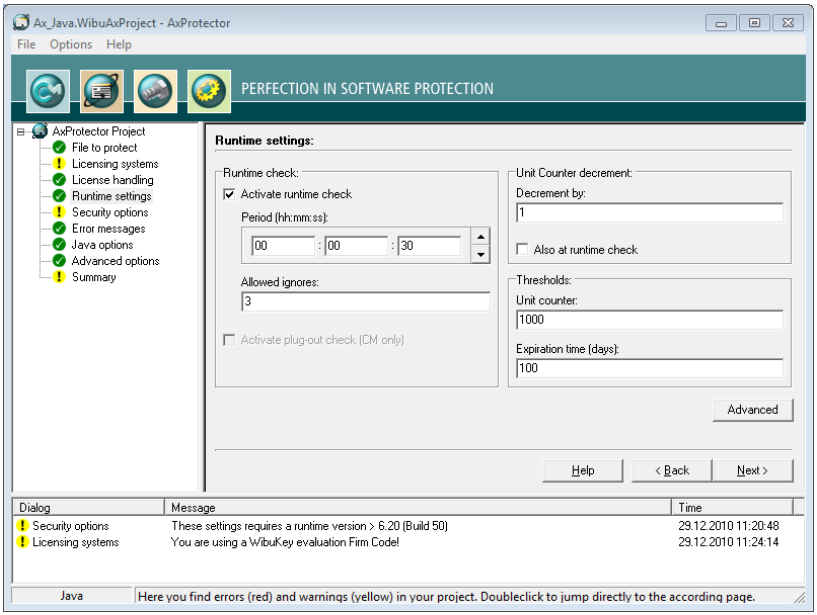


Figure 48: *AxProtector* - Java "Runtime Settings"

Runtime Check

In this group you define whether and how often the protected application checks the license at runtime.

Activate Runtime Check — Activates or deactivates the check at runtime.

Period — Defines the period between two checks. You specify this time interval in the format: hours: minutes: seconds.

Max. Allowed Ignores — Defines how often the end-user is able to ignore a failed check.

If the connection to a *WibuBox* should fail or the license cannot be accessed, you can you can assign a reasonable number of “ignores” allowing the end-user to continue working without a license access.

Unit Counter Decrement

Decrementing a *UNIT COUNTER* can serve to establish the validity of licenses in a *WibuBox*. This group allows you to define this behavior.

Decrement by — Defines the value by which the *UNIT COUNTER* is decremented. This option causes a decrement of the counter when the protected application starts.

When the **"Also at Runtime Check"** option is activated and the specifications are set as shown in Figure 48 every 30 seconds (see the defined period) a set UNIT COUNTER is decremented by a value of 1.

Also at Runtime Check

Decrements the UNIT COUNTER also at runtime of the protected application.

This option works only when the **"Also at Runtime Check"** option in the **Runtime Check** group is activated.

Thresholds

In this group you define when a message is issued to give information on the validity of a license.

Unit Counter

Where the defined threshold falls short, a warning message is issued.

Expiration Time (days)

When the specified EXPIRATION TIME (in days) is achieved within the defined threshold, a warning message is issued.

6.5 Advanced Runtime Settings

This input window lets you define further settings at the runtime of an encrypted application.

With the exception of the UNIT COUNTER and EXPIRATION TIME check at runtime all other settings are valid only for the licensing systems *CodeMeter* or *CodeMeterAct*.

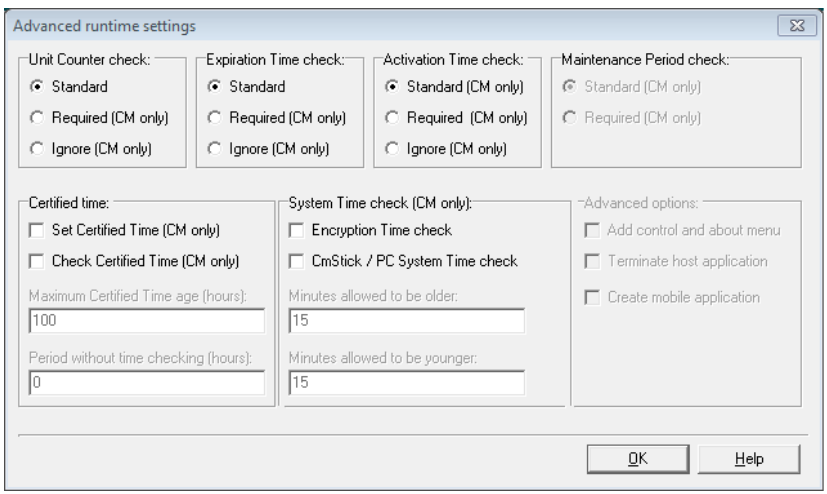


Figure 49: *AxProtector* - Windows "Advanced Runtime Settings"

Unit Counter Check

Defines the handling of a *UNIT COUNTER* when set in a license.

Standard

Decrements at runtime and/or start time an existing *UNIT COUNTER* entry in a license by the value defined on the previous page. When the *UNIT COUNTER* reaches 0 (null) the encrypted application does not start.

Expiration Time Check

Defines the handling of an *EXPIRATION TIME* set in a license.

Standard

Checks for an existing *EXPIRATION TIME* entry in a license. However, the application also starts when no *EXPIRATION TIME* entry exists, or the current date precedes the *EXPIRATION TIME*.

6.6 Security Options

This input window lets you select from different mechanisms and methods for protecting your application. You are able to scale the degree of security for yourself.

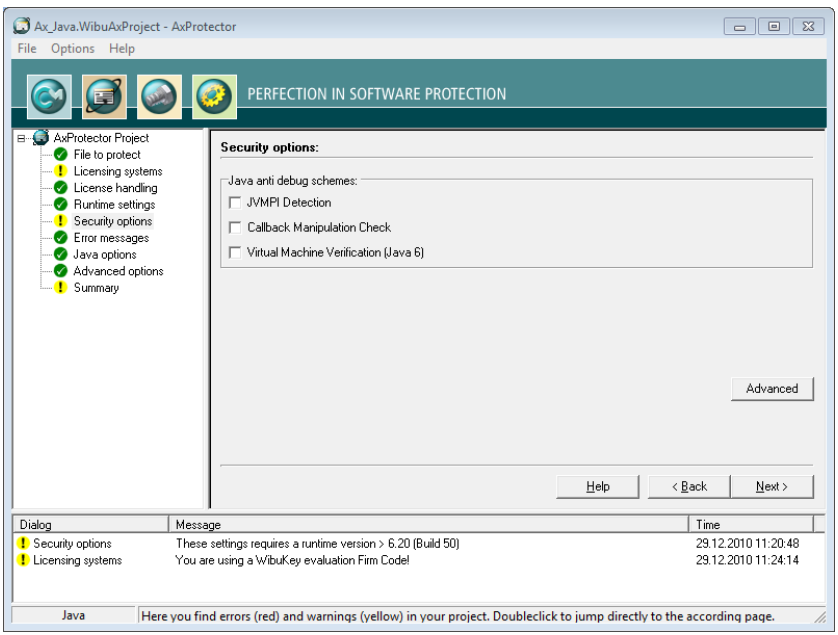


Figure 50: *AxProtector* - Java "Security Options"

JVMPI Detection

Activating this checkbox starts the detection of the Java Virtual Machine Profiler Interface (JVMPI). Using JVMPI the Java Virtual Machine is manipulable sending messages to the native code. In particular, the event `JVMPI_EVENT_CLASS_LOAD_HOOK` may be used to intercept the unaltered byte code of the class actually loaded. The activation of this option prevents this interception.

Callback Manipulation Check

Activating this checkbox protects against the manipulation of callback functions, i.e. functions which are transferred as parameters to other functions are checked.

VM Verification (Java 6)

Activating this checkbox checks for the correct Java Virtual Machine runtime environment for Java 6 (1.6).

6.7 Advanced Security Options

This input window lets you define further settings.

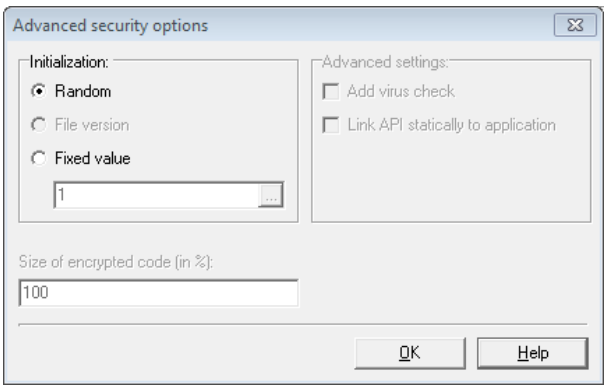


Figure 51: *AxProtector* - Java "Advanced Security Options"

Initialization

For the initialization of the encryption of a protected application several options exist.

Random

Uses random data to initialize the encryption.

File Version

Uses the file version as an integral part in initializing the encryption.

Fixed Value

Uses a fixed value to initialize the encryption.

Use a fixed value for initialization in order to get binary-compatible results each time you encrypt the same application.

For example, this makes sense when you want to regenerate an

encrypted application from an earlier project version. This guarantees an exact match to an earlier encrypted and delivered version.

6.8 Error Messages

This input window lets you define the messages displayed if errors occur. You define whether a user message class with a separate error display is used, or whether you use default error message windows.

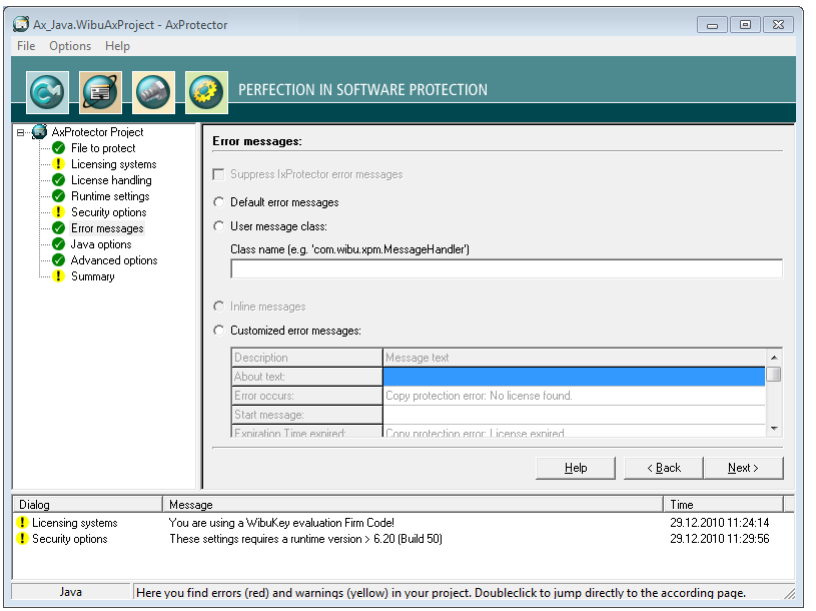


Figure 52: *AxProtector* - Java "Error Messages"

Default Error Messages

All errors occurring at the runtime of a protected application display default error messages.

User Message Class

Activate this option to use the `User Message Class`.

Class name

Specify here the file name without path information and extension.

Customized Error Messages

Activate this option to enter customized error messages displayed in the message boxes below.

6.9 Java Options

This input window lets you determine some parameters for the configuration of the Java runtime environment.

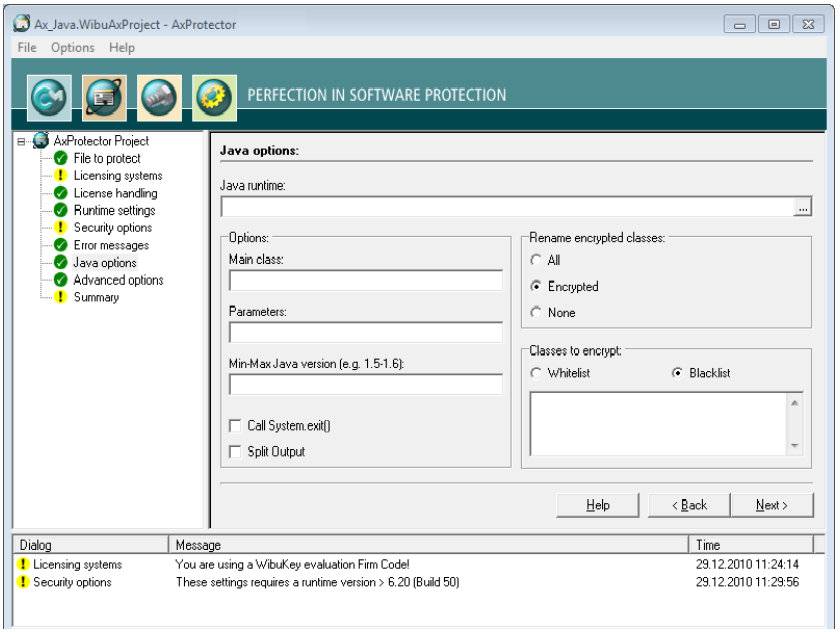


Figure 53: *AxProtector* - Java "Java Options" Java Runtime (java.exe)

Using the **"..."** button specify the `java.exe` file of the installed runtime environment.

Set further parameter in the **Options** group.

Main class

Enter here the name of the Java main class.

Parameter

Define here the parameters for calling the Java main class.

Minimum Java Version

Enter here the required minimum Java version.

When the check fails, a respective error message is issued.

This ensures already at start of the protected application that the functionality of your application requires is guaranteed.

Call System.exit()

Activate this option to exit the application by the call of `System.exit()` after return to the Java main class.

This ensures that in the case errors occur, the protected application correctly and completely shuts down. Even when the error occurred outside the Java main class.

Split Output Activate this option to save runtime classes to the separate `WibuXpm4Jruntime.jar` file.

Swapping the WIBU ClassLoader to a separate file increases performance of the protected application. Then even in the case of multiple encrypted classes, the WIBU ClassLoader will be only one-time loaded.

Using checkboxes in the **Rename encrypted classes** group allows you to determine the classes which classes will be renamed, and loaded into the WIBU ClassLoader.

For all class-related settings, the classes are renamed, and follow the pattern: `<MyClass>.class.wibu`.

All Activate this option to rename all existing classes.

Encrypted Activate this option to rename encrypted classes only.

None Activate this option to rename no classes.

When you rename encrypted classes only, only these classes are loaded by the WIBU ClassLoader. This improves the performance of the application. When you rename all classes, the security is increased at a small margin but eventually the performance of the protected application is negatively affected.

The group **Classes to encrypt** allows you to assign white or black list to classes.

Whitelist All classes referred to in the whitelist will be encrypted. This whitelist is saved to the jar-archive as an unencrypted text file `com/wibu/xpm/encrypted`.

Blacklist All classes referred to in the blacklist will not be encrypted.

AxProtector syntax: `-JL[W|B]:<whitelist|blacklist>`

Using these list give you direct bearing on the classes to be encrypted. For example, eventually it does not make sense to protect classes of third party providers, and stress the application performance.

For the output of error messages at the runtime of the encrypted Java application you may use the error class `com.wibu.xpm.MessageHandler`.

6.10 Advanced Options

This input window lets you set further options for the encryption.

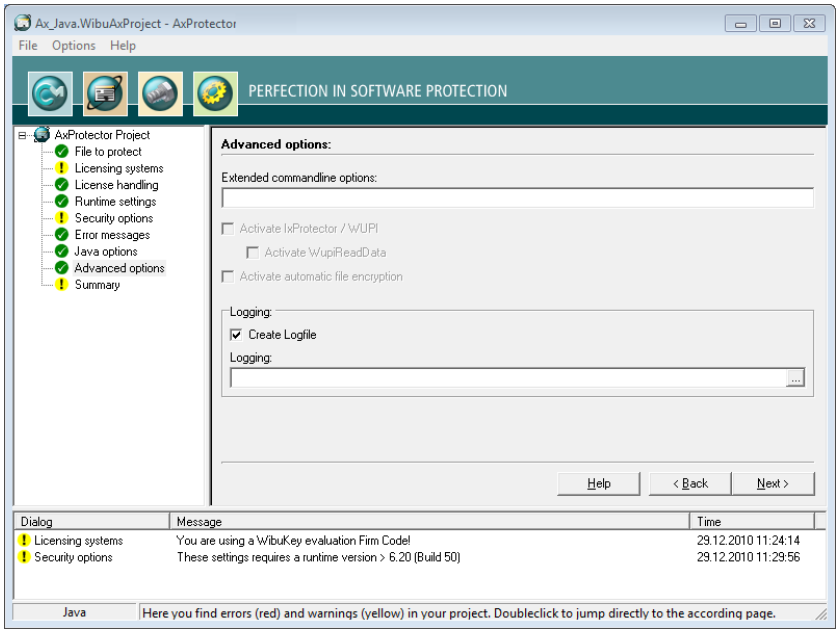


Figure 54: *AxProtector* - Java "Advanced Options"

Extended Commandline Options

Here you are able to directly enter extended options or new feature functions using *AxProtector* commandline parameters.

For more information, please contact support at Wibu-Systems.

Create Log File

Activate this checkbox to create file logging for the activities of *AxProtector*.

Logging

Specify the path and file name of this log file.

If you specify the name of the file only, by default, this file is saved to the directory `%\Program Files\WIBU-SYSTEMS\AxProtector\DevKit\bin`.

6.11 Summary

This input window shows you a summary of all the settings you defined for the automatic protection of your application, and allows you to start the encryption process.

For subsequent use, the contents of this page can be copied to a `*.wbc` file (WIBU Configuration file). Copy the content into a text

file, and change the file extension to `*.wbc`.

Alternatively, you may also use this file to protect your application using the *AxProtector* commandline tool. In the commandline type `AxProtector.exe @*.wbc` (see page 189).

Alternatively, using the **"File | export wbc-file"** menu item, you can also create the corresponding `*.wbc` file.

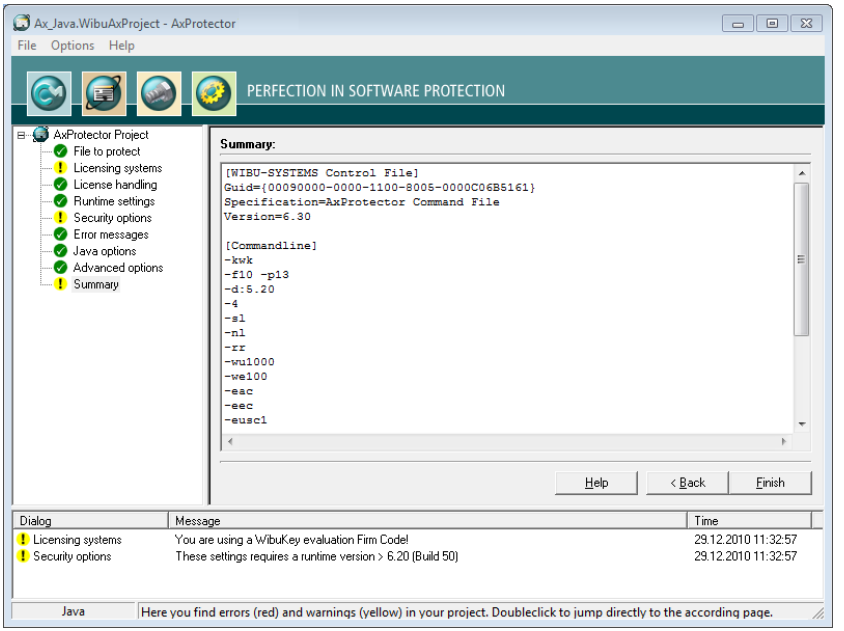


Figure 55: *AxProtector* - Java "Summary"

Finish Starts the encryption using *AxProtector* applying the settings you previously defined.

6.12 Protection Result

The result of the encryption with all relevant settings is displayed in a separate window.

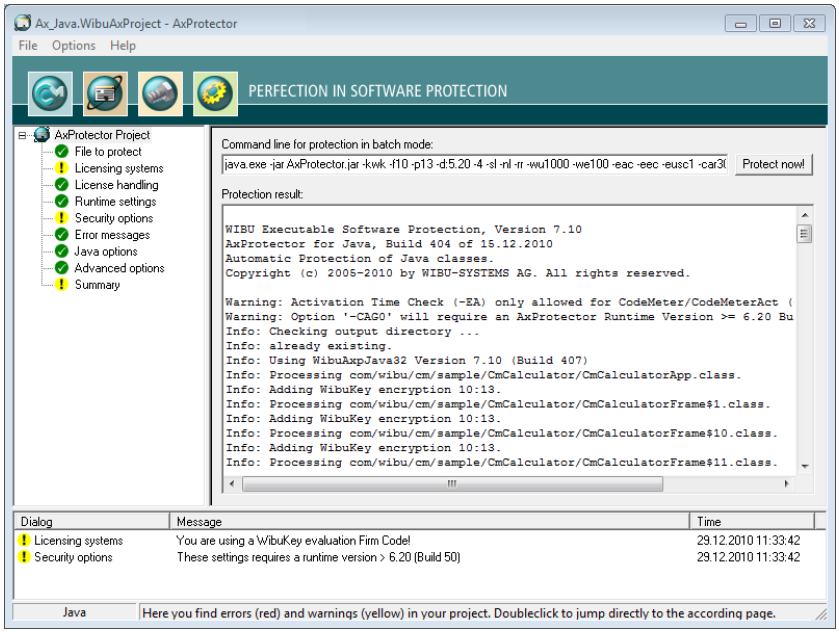


Figure 56: *AxProtector* - Java "Protection Result"

Protect Now

When you need to repeat the encryption operation, click the **"Protect Now"** button. Then the *AxProtector* commandline is executed in batch mode.

You are also able to copy the *AxProtector* commandline for the batch mode to the clipboard and insert it in the commandline input. Subsequently, you can edit it and apply any desired changes

7 File Encryption

AxProtector provides the automatic protection of files your protected application uses. This protection by encryption without altering the source code covers, for example:

- Flash applications consisting of a single `*.exe` und many `*.swf` files,
- database applications, e.g. Visual Fox Pro applications consisting of a single *.exe and a single or multiple database files,
- configuration data saved to separate files to be read by your software,
- scripts saved to separate files to be processed by your software,
- data, e.g. measuring data recorded or visualized in your application,
- documents the user generates using your protected application.

7.1 File to Protect

To safely encrypt an executable file using *AxProtector*, first select the file you want to protect.

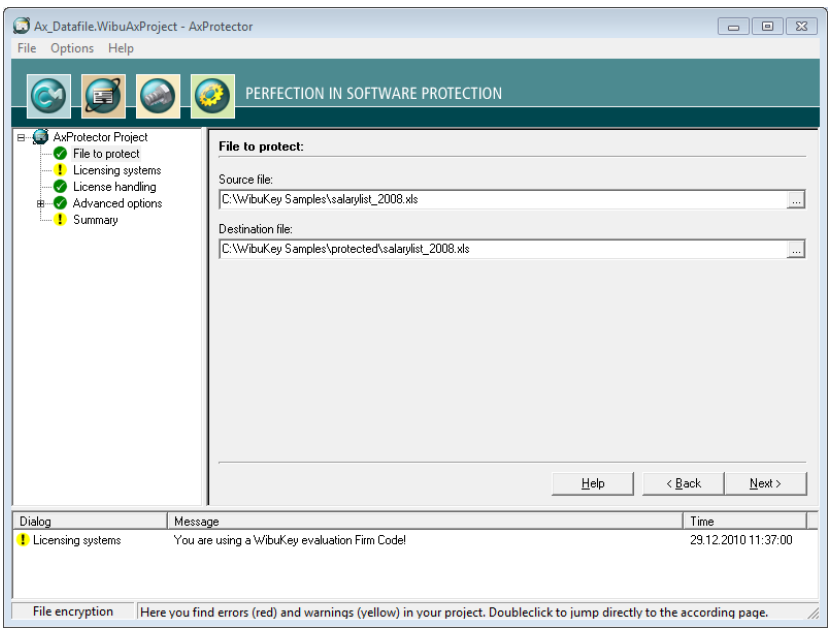


Figure 57: *AxProtector* - File Encryption "File to Protect"

Source File

Click on the **"..."** button and select the file to protect using the **"Open"** system dialog. Alternatively, manually specify the path and name of the file in this field.

As alternative to the **"..."** button, you may also directly drag & drop the source file from Windows Explorer into the source file field.

Destination File

After you selected the source file, *AxProtector* automatically creates a subfolder `[..\protected\..]`. You may change this default by manually specifying the path and name of the destination file. Then the destination file corresponds to your protected application.

7.2 Licensing Systems

After you select the file to be protected, the “Licensing systems” fields will appear in the input window. This is where you can select which protection schemes will be used. Depending on your requirements, you can select one or all of the check boxes.

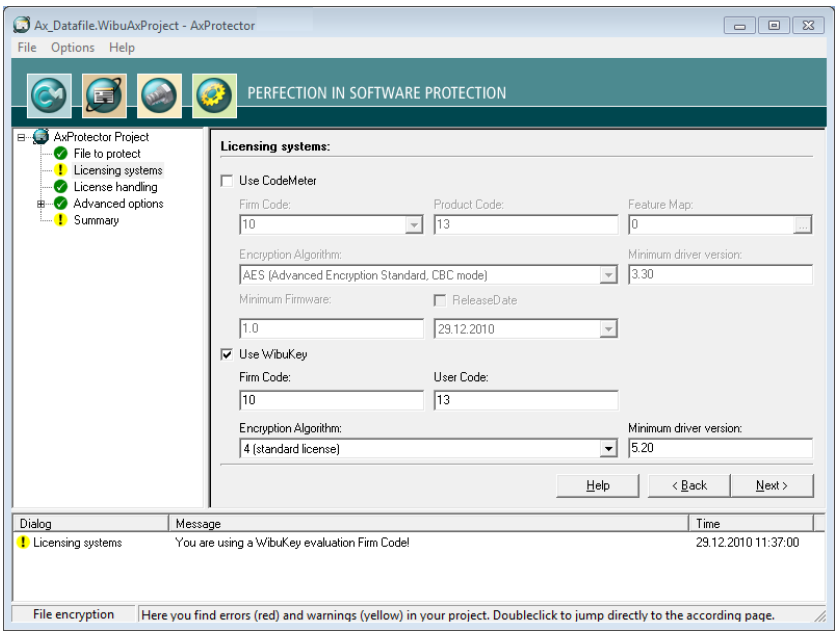


Figure 58: *AxProtector* – File Encryption "Licensing systems"

If you are switching from *WibuKey* to *CodeMeter*, please activate both hardware platforms.
In this way, you are able to ship updates and upgrades to existing customers who already have a *WibuBox* without the need to replace

the hardware. New end-users will be the ones to receive a *CmStick* together with the protected application.

You may also choose to protect using *CodeMeterAct*, the software-based copy protection system. For more information on *CodeMeterAct* visit the Wibu-Systems homepage.

For *WibuKey* the following settings are available:

Firm Code

Specify the FIRM CODE to be used for encrypting the software.

The FIRM CODE 10 used in the figure above is the Evaluation-FIRM CODE found in the *WibuKey* Software Development Kits (SDK). In real life you would not use a FIRM CODE of 10, since this would be insecure. As a registered Licensor, you will be issued your own unique FIRM CODE.

User Code

Enter the PRODUCT CODE which defines the encryption of a specific product. You can freely choose this identifier, e.g. for a separate module of a software application, or for a single application.

Encryption Algorithm

Select the algorithm to encrypt your software. By default, *WibuKey* currently supports Algorithm 4. Algorithm 5 is for reduced licenses. Use Algorithms 1 to 3 for downward compatibility only.

Minimum Driver Version

Enter the minimum driver version required for the installed *WibuKey* driver.

Setting the driver version is also required when, for example, you wish to use new features for the encryption of an application. Older driver versions will not support these new features, and will trigger error messages when starting the protected software.

For setting *CodeMeter* and *CodeMeterAct* options, see the separate *CodeMeter* Developer Guide.

7.3 License Handling

This input window lets you to define whether the protected application is to search for existing licenses locally in *WibuBoxes*, in the network, or both. Moreover, you can define the license allocation mode.

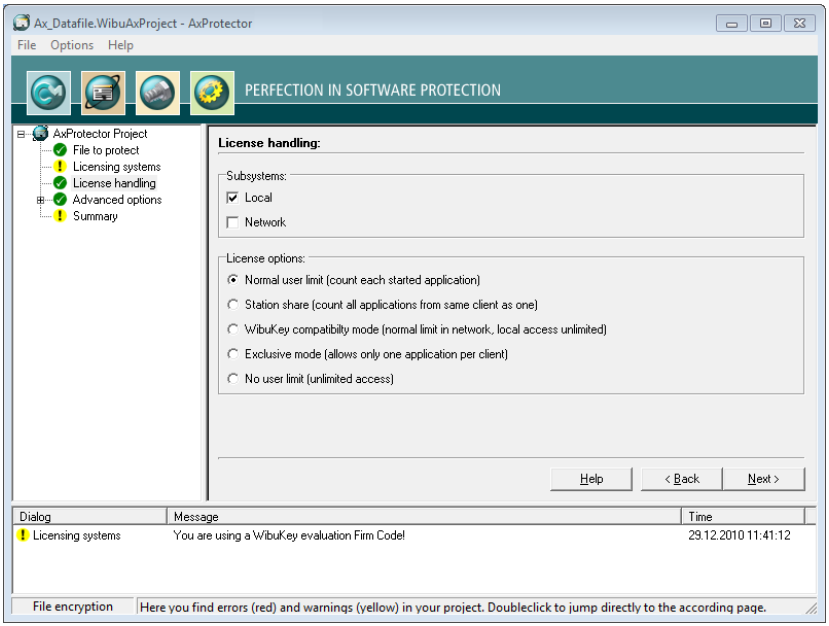


Figure 59: *AxProtector* - File Encryption "License Handling"

Subsystems

Here you can define in which subsystem (local or network) the protected application is to search for matching license(s).

Local

This setting determines if the protected application searches exclusively for licenses located on the same PC, or allocated to the same VMware.

Network

This setting determines that the license of the protected applications is to be sought in the network, i.e. only PCs are accessed where *WibuKey Server* runs.

On selecting both subsystems at the same time, the license is first sought locally and then subsequently on the network.

License Options

In this group you define how started instances of the protected applications perform, together with the allocation of licenses.

Normal user limit

Here each started instance allocates a single license. It does not make a difference if the *WibuBox* was found locally, or on a network .

Station Share

Here multiple instances can be started on a single PC. But, allocate only a single license.

You use this setting, for example, when you want to provide the end-user with the option of starting the application several times. On a terminal server each session allocates a license. In virtual machines each machine allocates a license.

WibuKey Compatibility Mode

Here each started instance in the network allocates a license (normal user limit) but the local access is unlimited (no user limit) .

This allocation option exists only because of compatibility issues with *WibuKey*. Wibu-Systems recommends the setting ‘normal user limit’ and ‘station share’.

Exclusive Mode

Here a protected application can be started only once on a PC.

No user limit

Here any number of instances of the protected application can be started locally or in a network, and no additional licenses are allocated. Allocated licenses in this mode can be re-used.

7.4 Advanced Options

This input window lets you set further options for the encryption.

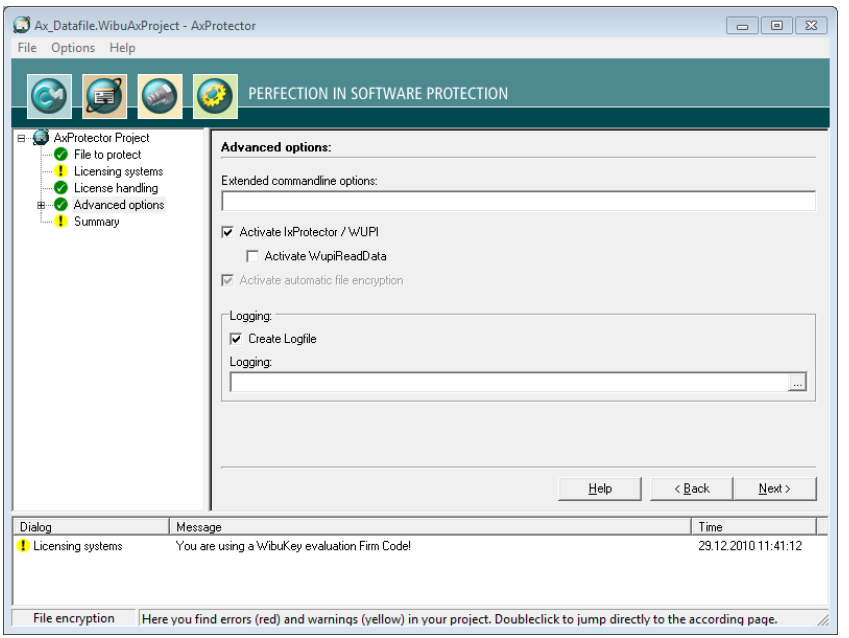


Figure 60: *AxProtector* - File Encryption "Advanced Options"

Extended commandline options

Here you are able to directly enter extended options or new feature functions using *AxProtector* commandline parameters.

For more information, please contact support at Wibu-Systems.

Activate IxProtector

Activate this checkbox to allow for the later creation and editing of license lists and function lists. These you need to protect using *IxProtector* via the *Software Protection API (WUPI)* (see page 172).

Activate WupiReadData

When activated this checkbox results in reading data from the *WibuBox* which has been previously stored at a defined location (see page 195).

Create Log File

Activate this checkbox to create file logging for the activities of *AxProtector*.

Logging

Specify the path and file name of this log file.

If you specify the name of the file only, by default, this file is saved to the directory `%\Program Files\WIBU-SYSTEMS\AxProtector\DevKit\bin`.

7.4.1 License Lists

This menu item lets you manage license lists. These you need to protect using *IxProtector* via the *Software Protection API (WUPI)* (see page 196) and for file encryption.

License lists consist of an unique identifier (**ID**), a **Description** and hold specifications on **Items** and **Item Details**.

This **ID** corresponds to the index number you require when addressing a license using most of the WUPI commands (see page 196).

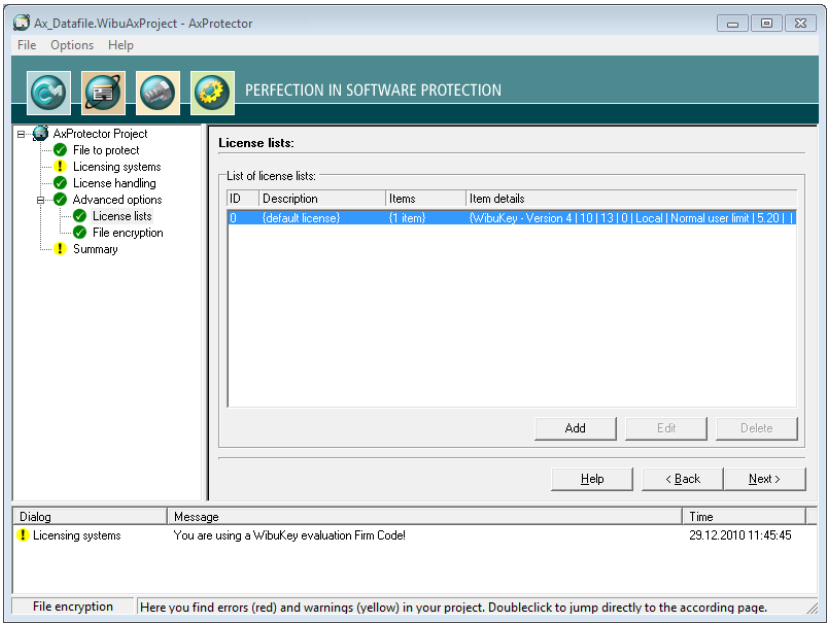


Figure 61: *AxProtector* - File Encryption "License Lists"

7.4.2 Add License Lists

This menu item lets you create license lists. Please proceed as follows:

❶ Click the button **"Add"**.

❷ Assign an **ID** in the *LICENSE LIST* group and complete the **Description** field.

ID

This ID uniquely identifies a license list and serves for referencing.

By default, an **ID** of **0** is initially set by the selection of the copy protection system. Following, you are able to add license list entries starting with an **ID** of **1**.

Description

Here you will describe a license list with text.

❶ Define the license by completing the fields in the *LICENSE ITEM DETAILS* group.

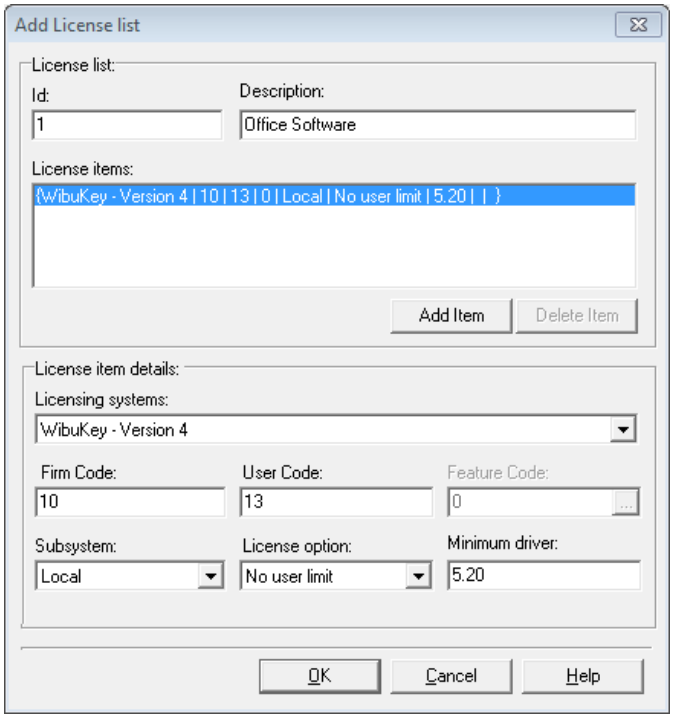


Figure 62: *AxProtector* - File Encryption "Add License Lists"

Licensing systems

Select the copy protection system used for protection of the license (*CodeMeter, CodeMeterAct* or *WibuKey*) .

Firm Code

Enter the FIRM CODE used for the protection of the license.

User Code

Enter the USER CODE used for the protection of the license.

Subsystem

Select the subsystem in which the protected application is to search (local or network), and define the search order.

License Options

Select the options for license allocation:

- *NORMAL USER LIMIT*
- *STATION SHARE*
- *WK COMPATIBILITY MODE*
- *EXCLUSIVE MODE*
- *NO USER LIMIT*

Minimum Driver Version

Specify the required minimum driver version for the protected application.

❷ Click on the **"Add"** button in the *LICENSE LIST* group.
The summary of your specifications are displayed in the license item list.

❸ Click the **"OK"** button.
The new license data is added to the license list.

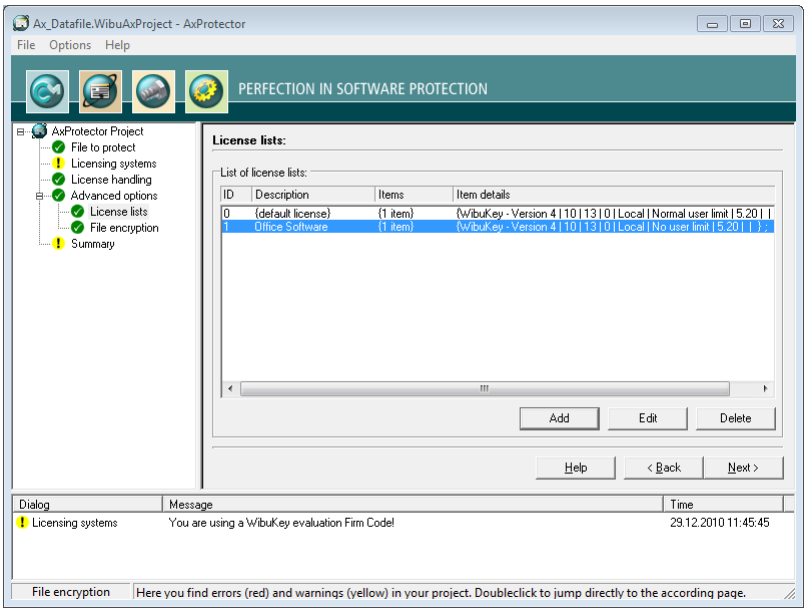


Figure 63: *AxProtector* - File Encryption "Specified License List"

7.4.3 File Encryption

This menu item lets you define the rules on how an application accesses the encrypted files- In addition, you have the option to define those rules in a list for different file types. You can add as many file types as possible. For a file only one file type is required.

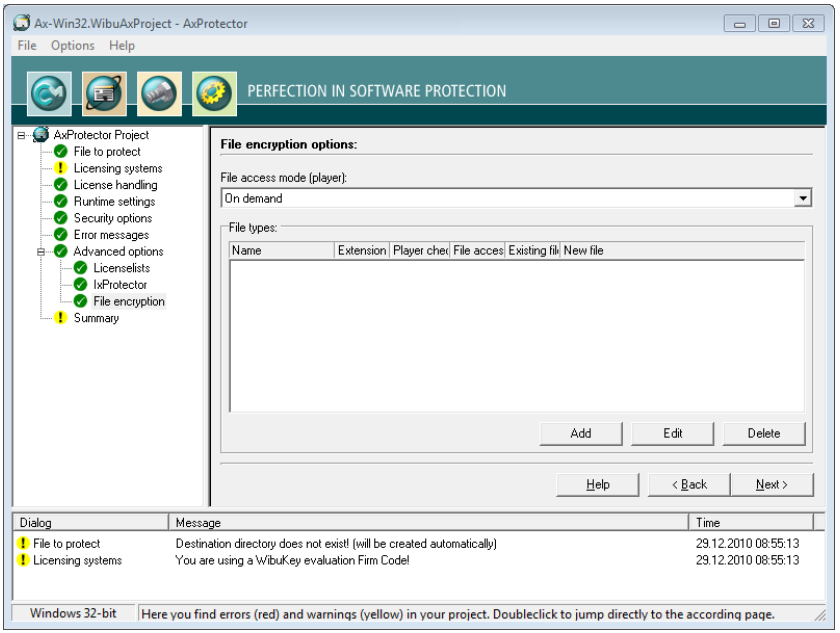


Figure 64: *AxProtector* - File Encryption "File Encryption"

Add File Types

Click on the **"Add"** button to add a new file type to the list.

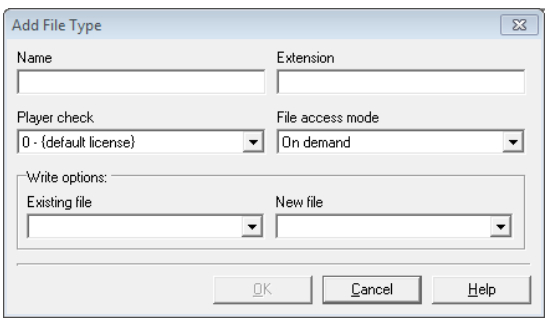


Figure 65: *AxProtector* - File Encryption "Add File Type"

Name

Enter the describing descriptive name for the file type. This name has no impact on the encryption.

Extension

Enter the file extension of the file type you create, e.g. `txt` for text files.

Player Check

In this group you define whether the license options of the accessing application (player) are checked when the encryption takes place.

Option	Description
`License list`	The player has to be encrypted using a license from this license list.  For example, this allows you to define that a specific file type is accessed exclusively by the application you encrypted.
`No player check`	No check of the accessing application is performed.

File Access Mode

The File Access Mode defines how the player is prepared for the access of protected files. This mode allows you to configure the memory required and the runtime behavior.

The selection of a suitable mode depends on the type of the player and the size of the file. For example, when working with video files you should select "Huge file mode (read only)". In the case of smaller files (configuration files) you may access several times, the mode "At once" is preferable.

Since the selection of different runtime settings for the player and the data are possible, at runtime the more restrictive settings apply

On demand

The player reserves RAM space for the complete file to be read; but reads only the required part – strictly speaking all 4 Kbyte blocks are holding this part – and decrypts these blocks. For further accesses to the protected file, more required blocks are loaded (on demand) and decrypted. When the required part is located in blocks already loaded, the decrypted image in the memory is used. In this way, step-by-step the player builds up a complete memory image of the required file.

This mode requires a lot of memory (the same size as the file to be loaded). However caching the decrypted data provides for good performance at runtime when accessing already decrypted blocks. This mode is available for read and write access.

At once

The player reserves RAM space for the complete file to be read; completely reads it, and completely decrypts it. Further accesses to the protected files, use the decrypted memory image.

This mode requires a lot of memory (the same size as the file to be loaded). However caching the decrypted data provides a good performance at runtime. Compared to the "On demand" mode, this mode requires more time for first access (the file is completely loaded and decrypted). The performance of each additional access is increased because the file resides completely in memory, in a decrypted form. This mode is available for read and write access.

Huge file mode

The player reads the currently required parts of the protected file and decrypts them. This data is not cached in the memory.

This mode requires no additional memory. Multiple accesses to the same data means that the data has to be read and decrypted each time. This mode is available for read access only.

Write Options

Existing Files

In this group you define the settings on how changes to an existing file are saved.

Option	Description
`Original`	Changes are allowed. Where the file was encrypted, it is re-encrypted. Unencrypted files are saved with no decryption.
`No writing`	Write actions are not allowed. Just read-only access is allowed.
`License list`	Changes are only encrypted using the license options defined in the selected license list.

New File

In this group you will define the settings on how new files are saved.

Option	Description
`Plain`	New files are only saved unencrypted.
`No writing`	New files cannot be saved.  A new file is saved, however no data is saved to this file.
`License list`	New files are only encrypted using the license options defined in the selected license list.

7.5 Summary

This input window shows you a summary of all the settings you defined for the automatic protection of your application, and allows you to start the encryption process.

For subsequent use, the contents of this page can be copied to a `wbc` file (WIBU Configuration file). Copy the content into a text file, and change the file extension to `*.wbc`.

Alternatively, you may also use this file to protect your application using the *AxProtector* commandline tool. In the commandline type `AxProtector.exe @*.wbc` (see page 189).

Alternatively, using the **"File | export wbc-file"** menu item, you can also create the corresponding `wbc` file.

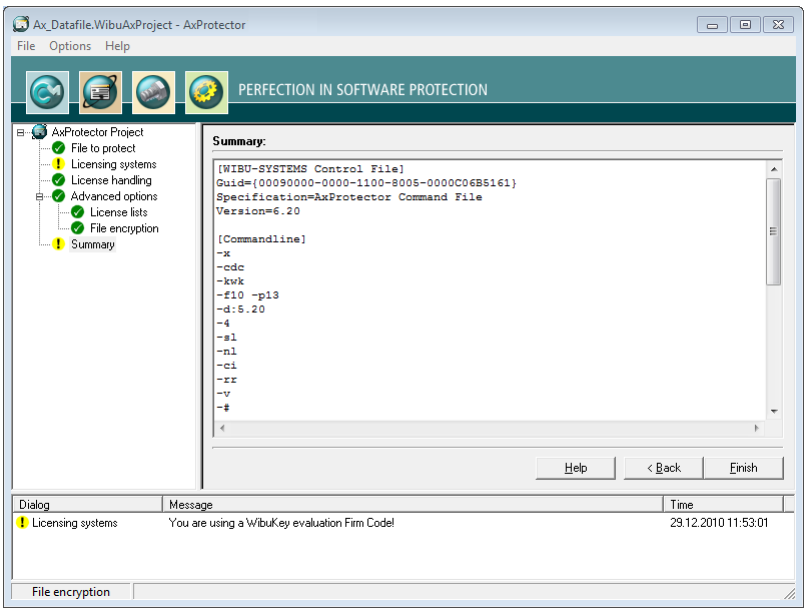


Figure 66: *AxProtector* - File Encryption "Summary"

Finish

Starts the encryption using *AxProtector* applying the settings you previously defined.

7.6 Protection Result

The result of the encryption with all relevant settings is displayed in a separate window.

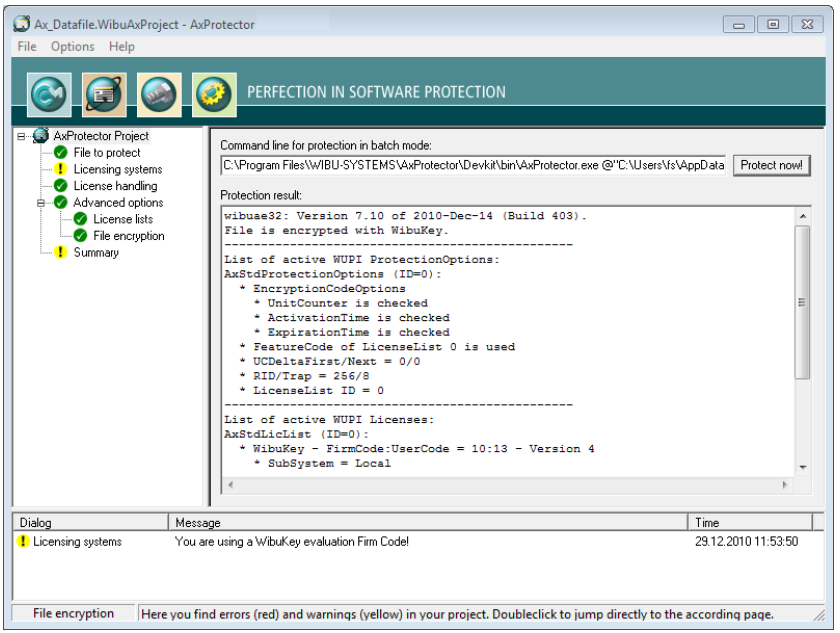


Figure 67: *AxProtector* - File Encryption "Protection Result"

Protect Now

When you need to repeat the encryption operation, click the **"Protect Now"** button. Then the *AxProtector* commandline is executed in batch mode.

You are also able to copy the *AxProtector* commandline for the batch mode to the clipboard and insert it in the commandline input. Subsequently, you can edit it and apply any desired changes.

8 IxProtector *only*

When you want to encrypt specified functions of an application using an index-based list, you select this project type. However, then the complete application is not additionally protected with *AxProtector.*

Wibu-Systems recommends to use *IxProtector* within *AxProtector* if no special requirements exist.

Then *IxProtector* finds the respective code areas and encrypts them. But even when you choose the project type "IxProtector only" increased security is fact, since *IxProtector* replaces the dummy DLL used by static code. Later on when executing the protected application, this DLL is not used any longer.

8.1 File to Protect

To safely encrypt affected code areas of an executable file using *AxProtector*, first select the file you want to protect.

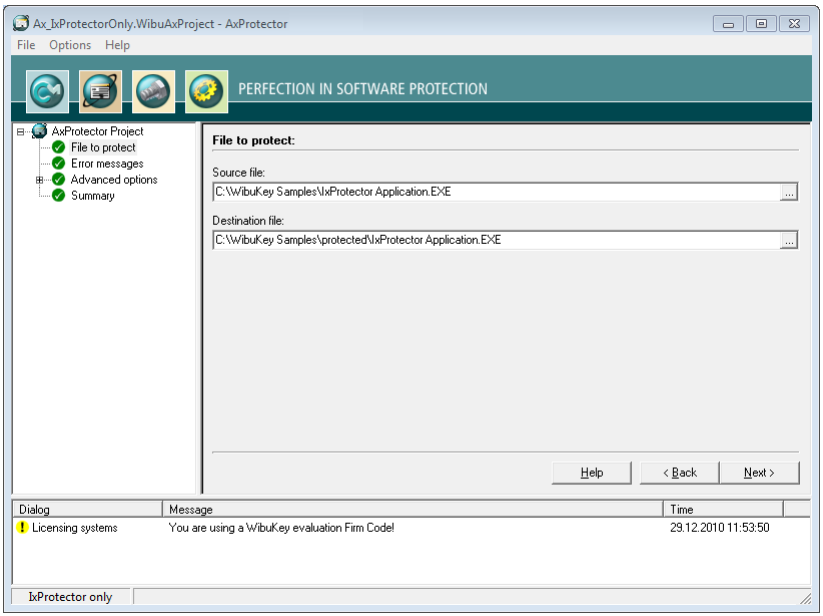


Figure 68: *AxProtector –IxProtector* only "File to Protect"

Source File

Click on the **"..."** button and select the file to protect using the **"Open"** system dialog. Alternatively, manually specify the path and name of the file in this field.

As alternative to the **"..."** button, you may also directly drag & drop the source file from Windows Explorer into the source file field.

Destination File

After you selected the source file, *AxProtector* automatically creates a subfolder `[..\protected\..]`. You may change this default by manually specifying the path and name of the destination file. Then the destination file corresponds to your protected application.

8.2 Error Messages

This input window lets you define the messages displayed if errors occur. You define whether a user message DLL with a separate error display is used, or whether you use default error message windows.

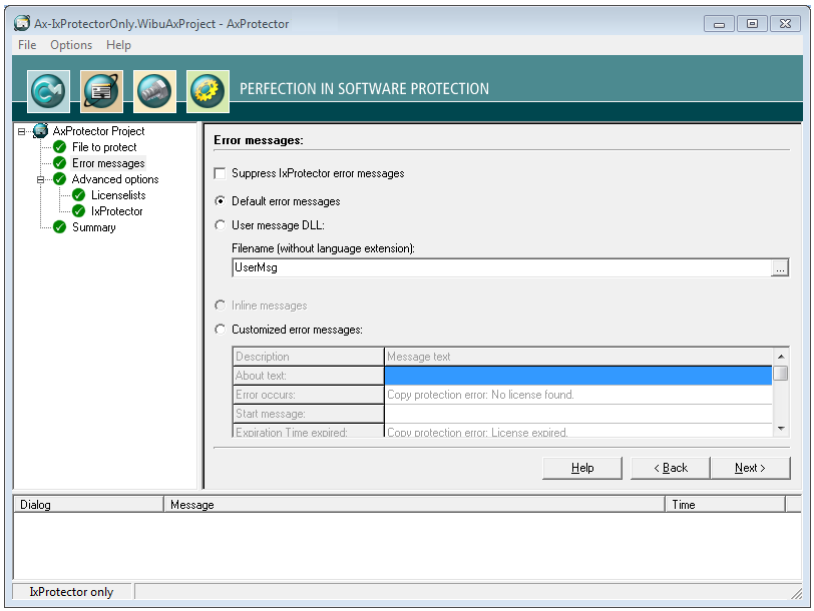


Figure 69: *AxProtector* - *IxProtector* Only "Error Messages"

Suppress IxProtector Error Messages

The output of *IxProtector* error messages is suppressed.

If you do not activate this option, when using *IxProtector* errors, additional message windows are displayed along with the messages you program in the project.

Default Error Messages

All errors occurring at the runtime of a protected application display default error messages.

User Message DLL

The ability to use the `User Message DLL` is activated. Error messages can be localized to different languages using *.`ini` files. In addition, you have the option to integrate your own designs to this file, for example, by using separate logos or text.

The *.`ini` files with the respective country suffix and the `Dll` program library are automatically saved to the directory where the application locates the files protected by *AxProtector*

File Name (without Language Extension)

Enter the file name without specifying path and language file extension.

The `UserMsgDll` is copied from the directory `%Program Files%\WIBU-SYSTEMS\AxProtector\DevKit\bin\UserMessage`. The corresponding *.`ini` files are also saved to this directory.

Customized Error Messages

Activate this option to enter customized error messages displayed in the message boxes below.

8.3 Advanced Options

This input window lets you set further options for the encryption.

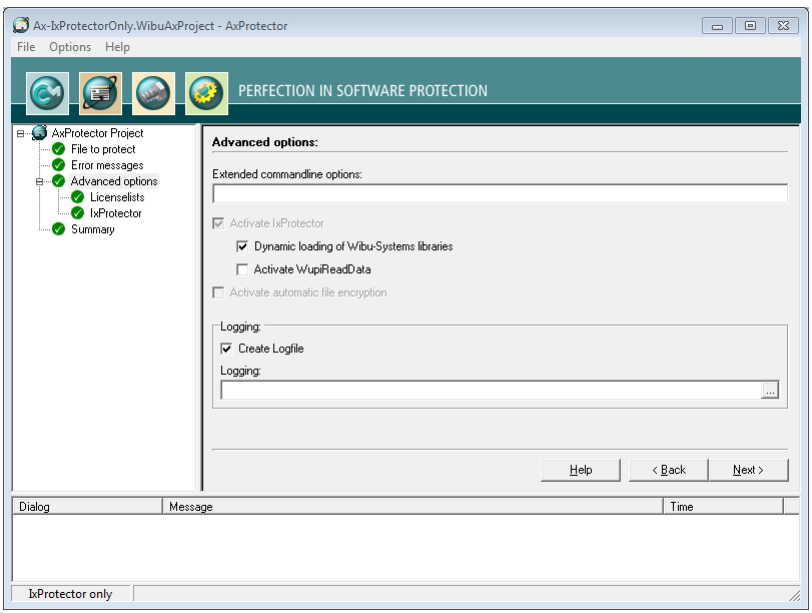


Figure 70: *AxProtector* - *IxProtector* Only "Advanced Options"

Extended Commandline Options

Here you are able to directly enter extended options or new feature functions using *AxProtector* commandline parameters.

For more information, please contact support at Wibu-Systems.

Dynamic loading of Wibu-Systems libraries

When activated this checkbox results in a special, more time-intensive process in the case of VB6 applications or on dynamic loading of Wibu-Systems libraries.

Activate WupiReadData

When activated this checkbox results in reading data from the *WibuBox* which has been previously stored at a defined location (see page 195).

Create Log File

Activate this checkbox to create file logging for the activities of *AxProtector*.

Logging

Specify the path and file name of this log file.

If you specify the name of the file only, by default, this file is saved to the directory `%\Program Files\WIBU-SYSTEMS\AxProtector\DevKit\bin`.

8.3.1 License Lists

This menu item lets you manage license lists. These you need to protect using *IxProtector* via the *Software Protection API (WUPI)* (see page 196).

License lists consist of a unique identifier (**ID**), a **Description** and hold specifications on **Items** and **Item Details**.

This **ID** corresponds to the index number you require when addressing a license using most of the WUPI commands (see page 196).

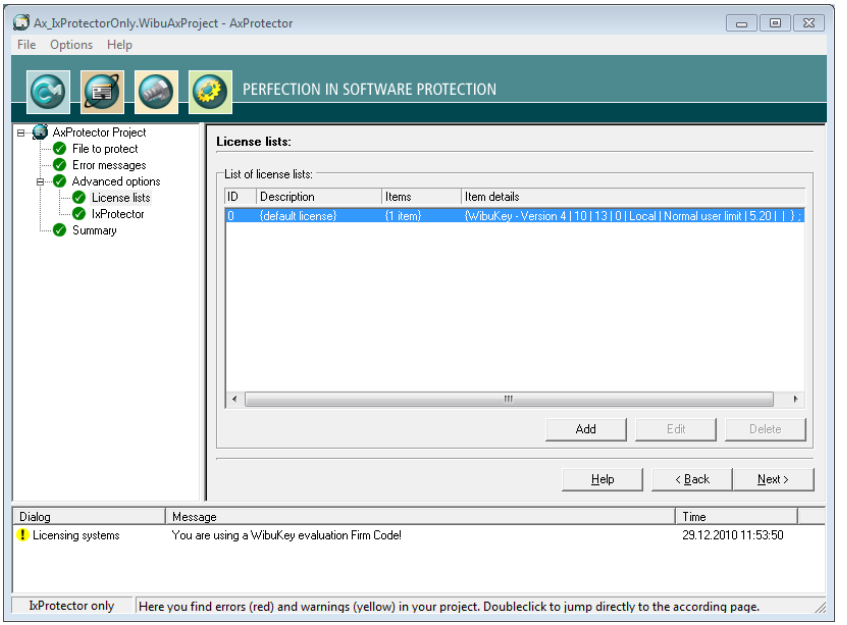


Figure 71: *AxProtector - IxProtector* Only "License Lists"

8.3.2 Add License Lists

This menu item lets you create license lists. Please proceed as follows:

❶ Click the button **"Add"**.

❷ Assign an **ID** in the *License List* group and complete the **Description** field.

ID

This ID uniquely identifies a license list and serves for referencing.

By default, an **ID** of **0** is initially set by the selection of the copy protection system. Following, you are able to add license list entries starting with an **ID** of **1**.

Description

Here you will describe a license list with text.

❶ Define the license by completing the fields in the *License Item Details* group.

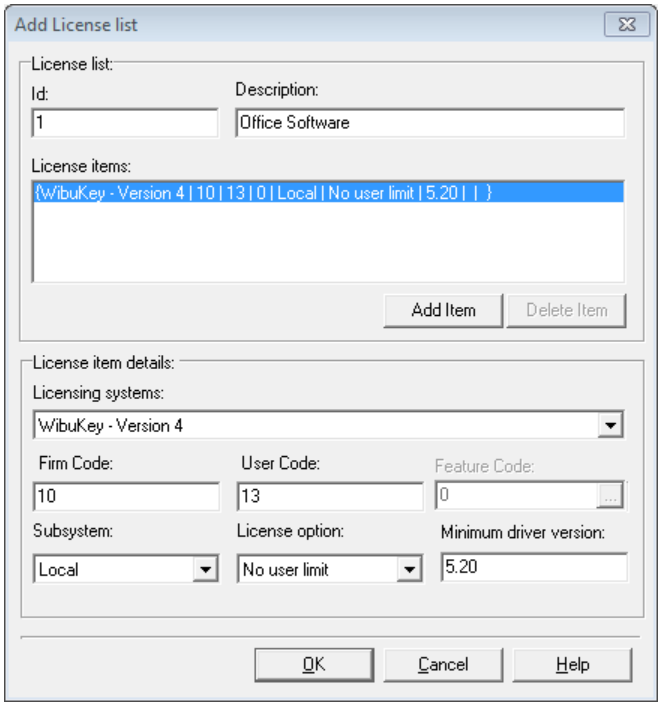


Figure 72: *AxProtector - IxProtector* Only "Add License Lists"

Licensing systems

Select the copy protection system used for protection of the license (*CodeMeter, CodeMeterAct* or *WibuKey*) .

Firm Code

Enter the FIRM CODE used for the protection of the license.

User Code

Enter the USER CODE used for the protection of the license.

Subsystem

Select the subsystem in which the protected application is to search (local or network), and define the search order.

License Option

Select the options for license allocation:

- *NORMAL USER LIMIT*
- *STATION SHARE*
- *WK COMPATIBILITY MODE*
- *EXCLUSIVE MODE*
- *NO USER LIMIT*

Minimum Driver Version

Specify the required minimum driver version for the protected application.

❶ Click on the **"Add"** button in the *LICENSE LIST* group.
The summary of your specifications are displayed in the license item list.

❷ Click the **"OK"** button.
The new license data is added to the license list.

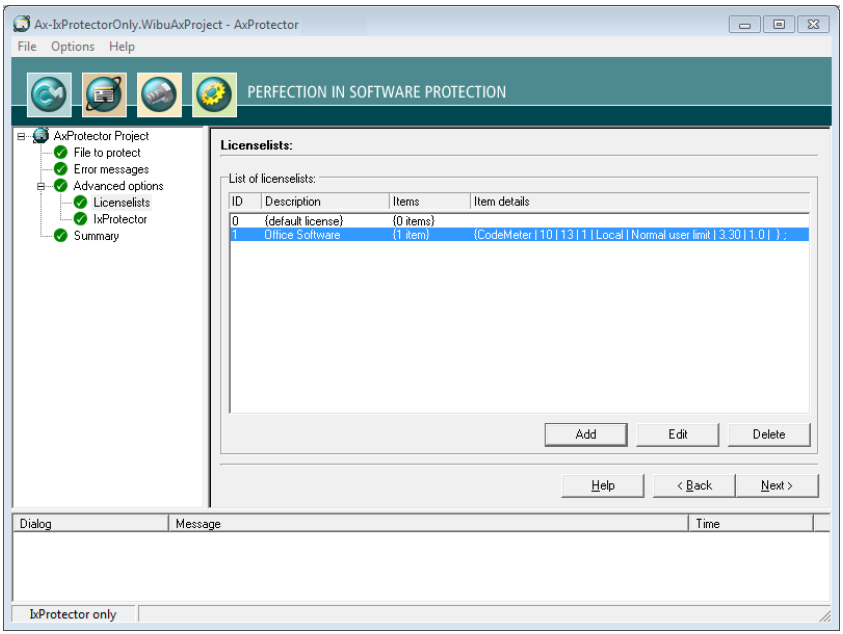


Figure 73: *AxProtector* - *IxProtector* Only "Specified License List"

8.3.3 *IxProtector*

This menu item lets you define single modules or program functions of the protected application (see page 172ff) .

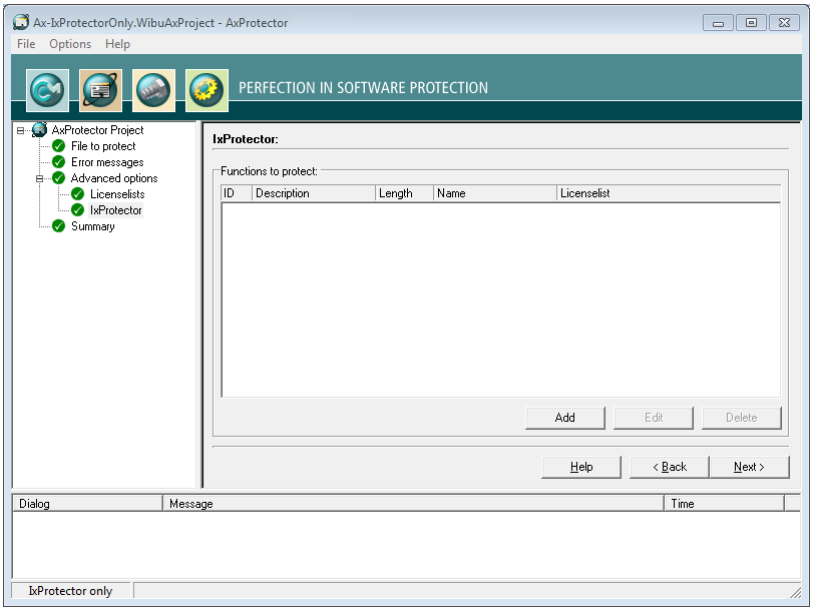


Figure 74: *AxProtector* - IxProtector Only "Function List"

Lists all specified function lists, including all properties.

Functions to protect

8.3.4 Add Function Lists

This menu item lets you create function lists. Please proceed as follows:

❶ Click the **"Add"** button in the *IxProtector Options* group.

❷ Define the function by completing the fields in the *Function* group.

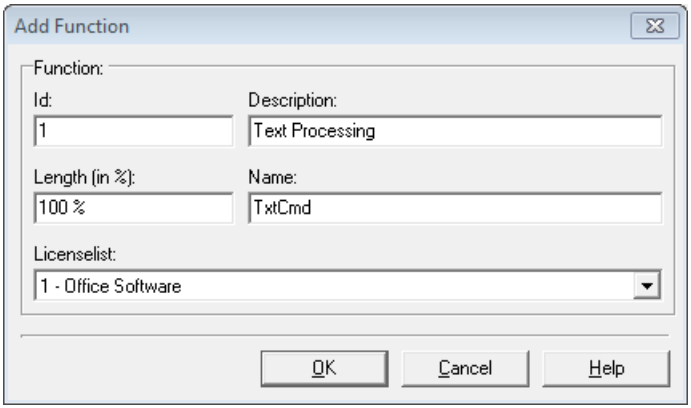


Figure 75: *AxProtector* - *IxProtector* Only – "Add Function"

ID

Uniquely identifies the function.

This **ID** corresponds to the identification you use when calling the WUPI commands **`WupiDecryptCode`** and **`WupiEncryptCode`** (see page 196).

Description

Enter a description of the function with text.

Length

The length of the array to be encrypted for the function is specified here. You enter the length, in percent, anywhere from 0 to 100%. If you want this number to represent percentage, you must enter the percent character (%). Alternatively, you are able to specify the length by number of bytes. Then *AxProtector* automatically calculates the length.

Name

Specify the name of the function to be encrypted.

The function name must match exactly the name used in the export list of the linked map file.
Please note the correct spelling (case sensitive, underline, etc.).

In order to obtain the exact function name from the executable file, you may, for example, use the application Microsoft Dependency Walker.

Microsoft Dependency Walker shows dependencies between 32-or 64-bit Windows PE files. A tree view shows all linked modules and imported and exported functions are displayed in tables. Dependency Walker is part of Windows XP SP2 Support Tools and also part of Microsoft Visual Studio including version 8.0 (Visual Studio 2008, i.e. version 9.0 no longer provides Dependency Walker).

License List

Selects an existing license to which the function is assigned. Then this license list is used for the encryption of the function.

❶ Click the **"OK"** button.
The new functions are added to the function list.

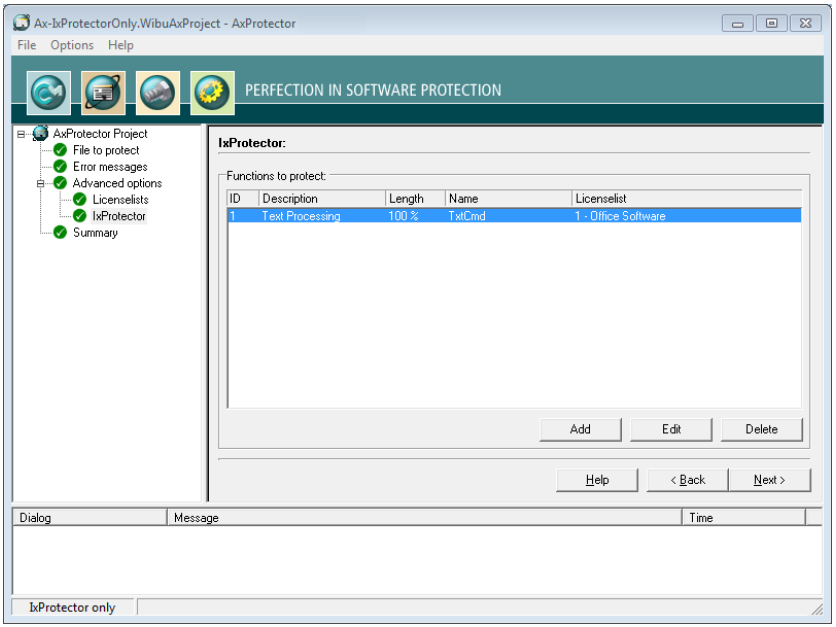


Figure 76: *AxProtector - IxProtector* Only "Specified Function List"

8.4 Summary

This input window shows you a summary of all the settings you defined for the automatic protection of your application, and allows you to start the encryption process.

For subsequent use, the contents of this page can be copied to a `wbc` file (WIBU Configuration file). Copy the content into a text file, and change the file extension to `*.wbc`.

Alternatively, you may also use this file to protect your application using the *AxProtector* commandline tool. In the commandline type `AxProtector.exe @*.wbc` (see page 189).

Alternatively, using the **"File | export wbc-file"** menu item, you can also create the corresponding `wbc` file.

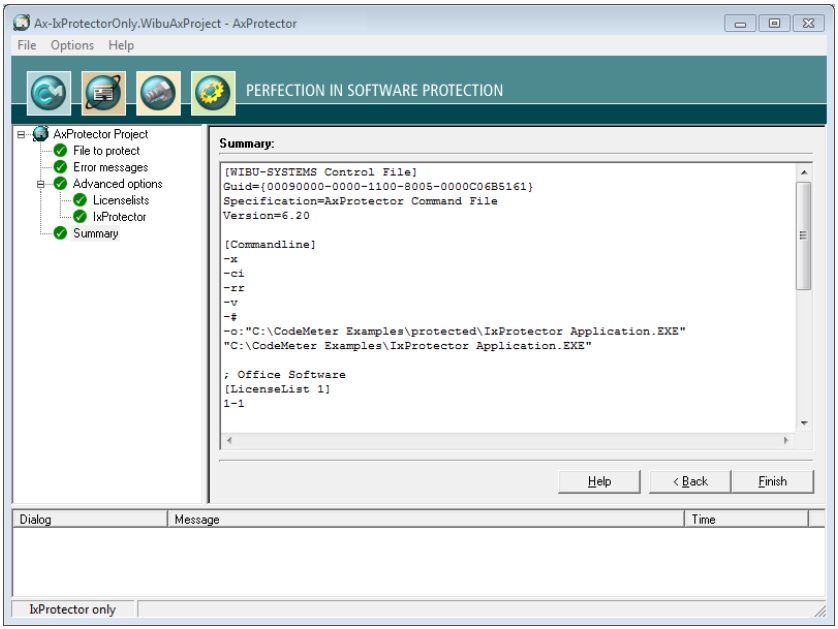


Figure 77: *AxProtector* - *IxProtector* Only "Summary"

Finish

Starts the encryption using *AxProtector* applying the settings you previously defined.

8.5 Protection Result

The result of the encryption with all relevant settings is displayed in a separate window.

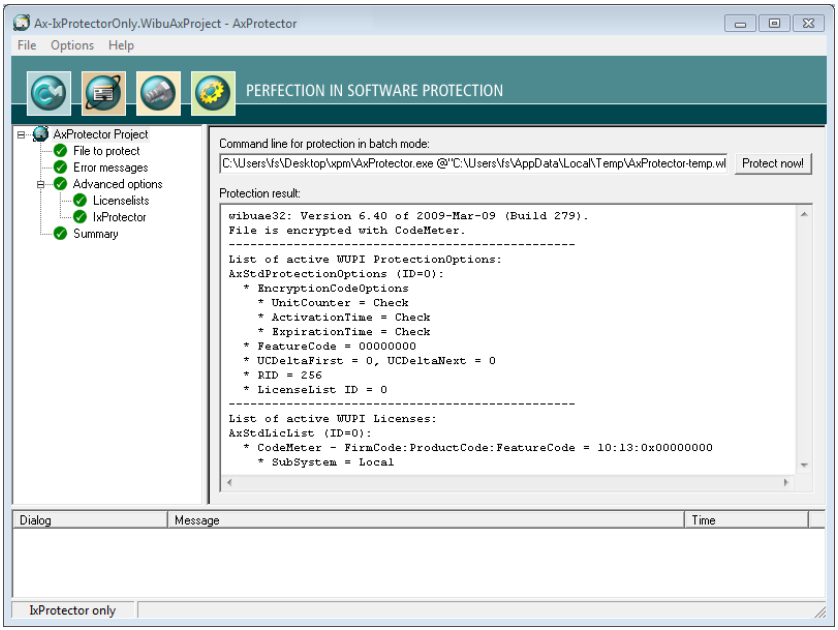


Figure 78: *AxProtector - IxProtector* Only "Protection Result"

Protect Now

When you need to repeat the encryption operation, click the **"Protect Now"** button. Then the *AxProtector* commandline is executed in batch mode.

You are also able to copy the *AxProtector* commandline for the batch mode to the clipboard and insert it in the commandline input. Subsequently, you can edit it and apply any desired changes.

9 Commandline options for *AxProtector*

Commandline versions

As an alternative to the graphical user interface of *AxProtector*, you can also set the options for automatic encryption using the *AxProtector* commandline. The commandline application comes in three versions you find in the directory "%\Program Files%\WIBU-SYSTEMS\AxProtector\DevKit\bin":

Version	Project types
AxProtector.exe	32 64 MacIxPro File
AxProtectorNet.exe AxProtectorNet4.exe	.NET
AxProtector.jar	Java
AxProtectorLin.exe	in a 32-bit and 64-bit version

Which options are valid for which *AxProtector* project type is indicated by the symbols in the margin.
Unless otherwise noted, an indication is valid until the next margin.

9.1 Basic Options

-R([F:s]|[R]|[V])	Initializes the random number generator.
F	Uses the specified value <s> for the initialization.
R	Uses a random value (default) for the initialization.
V	Uses information of the file version for the initialization (only 32 64 MacIxPro File).
-X	Links the static library of the copy protection system to the application to be protected.
	This option is set by default.
	Setting this option increases the security compared to linking the dynamic library.
-A	Searches a pre-defined space within the application to be protected to insert security code.
	This space is used only when defined big enough. The space must an initial *AxProtector* signature.
-A[AES]	Specifies the encryption algorithm [AES] (*CodeMeter* only).

By default, the AES encryption algorithm in CBC mode is used (Standard).

9.2 Options for the Copy Protection System

-K([CA]|[CM]|[WK]) Specifies the copy protection system.

CA uses *CodeMeterAct.*

CM uses *CodeMeter* (default).

WK[-x] uses *WibuKey*. x represents:

1 Encryption algorithm 1 (*WibuKey* only).

2 Encryption algorithm 2 (*WibuKey* only).

3 Encryption algorithm 3 (*WibuKey* only).

4 Encryption algorithm 4 (*WibuKey* only / Default).

5 Encryption algorithm 5 (*WibuKey* only).

-Fx Specifies the FIRM CODE (x) to be used.

Input of an unsigned integer value <n>.

-Px Specifies the PRODUCT CODE (x) to be used.

In the case of *WibuKey* specify the USER CODE.

Input of an unsigned integer value <n>.

-CFx Specifies the FEATURE CODE (x) to be used.

Only valid when using *CodeMeter.*

Input of an unsigned integer value <n>.

-RD([YYYYMonDD[,HH:MM:SS[:<Timezone>]]]|[:now])or according to Iso-8601 using T(ime) and Z(one) parameter :

-RD([<yyyy>-<mm>-<dd>[T<hh>:<mm>:<ss>[Z][±hh:mm oder ±hhmm] [±hh]]]|[:now])

defines the desired RELEASE DATE for encrypting and decrypting.
(*CodeMeter* and *CodeMeterAct* only)

Specification is in format year, month, and optional hours, minutes, seconds, and the timezone.

	The input of **[:now]** applies the current date.
-D:v	Specifies the minimum driver version (v).
	Input of v using (x.y).
	Default setting: *CodeMeter* 4.20.
	Default setting: *WibuKey* 5.20.
-FW:v	Defines the minimum firmware version (v).
	Input of v using (x.y).
	Default setting: *CodeMeter* 1.0
	is not used for *WibuKey*
-S([L]|[N]|C)	
	Specifies the search order of the subsystem when searching for licenses.
L	Uses the local subsystem (local).
N	User the network subsystem (network).
C	First searches the local and then the network subsystem, and if a network was found the network drive.
-N[C[A]|L[A]|N|X|X[A]]	
	Specifies the network access.
C[A]	*CONVENIENT MODE* (Compatibility Mode): here each started instance on the network allocates a (*NORMAL USER LIMIT*), and the local access is unlimited(*NO USER LIMIT*).  Since this is the default license allocation, this option ensures compatibility when both hardware platforms are used at the same time. (a: uses AUTO CANCEL (*WibuKey* only).
L[A]	*NORMAL USER LIMIT*: here each started instance allocates a license regardless whether the *CmStick* is found locally or on the network. (A: uses AUTO CANCEL (*WibuKey* only).
N	*NO USER LIMIT*: here any number of instances can be started locally or on the network. No licenses are allocated.

S — *STATION SHARE*: here several started applications on a client allocate only a single license.

> You use this option when allowing the end-user to start the protected application several times. On Terminal Server each session allocates a single license. In virtual machines each machine allocates a single license.

X]A] — *EXCLUSIVE MODE:* here only a single started instance per PC is allowed. Each access allocates a single license.

(A: uses AUTO CANCEL (*WibuKey* only).

> The options listed above should directly locate after the **–K** option since they refer to the actual defined copy protection system.
>
> When you use both copy protection systems for an executable file, the options set are valid for the defined copy protection system.

9.3 Options for Encryption and Decryption

-CA[[A[l]],[Ct[,u]],[D[m]],[E],[G[l[,1]]],[L],[M],[R[t][,m]],[S[p]],[T[t][,u]],[V], [Z]]

Encrypts the executable file using automatic encryption.

A `<l>` — Activates the security options (ADVANCED PROTECTION SCHEMES, APS).

`<l>` covers the options `[0, 15]`

> When applying more than one security option (APS), you can combine the options 1, 2, 4 and 8 with "or".
>
> APS2 and APS8 are mutually exclusive. In the case both options are set, automatically `-caa8` is applied.

Option	Description
1	Resource encryption applies (APS 1).
2	Static modification applies (APS 2).
4	Dynamic modification applies (APS 3).
8	Extended static modification applies (APS 4).

> `caa6` applies APS 2 and 3

	caa13 applies APS 1, 4 and 8
Ct<, u>	Checks the *CmStick* system time related to the PC time. A protected application runs only when the PC time in a time window is `t` minutes younger and, optionally, `u` minutes older than the *CmStick* system time. *CodeMeter* and *CodeMeterAct* only.
D `<m>`	Specifies the file access mode for the automatic encryption of files which have been encrypted using the option `-CD`.

`0`	Decrypts the file's content block by block (4Kb) on demand (on demand).
`1`	Decrypts the file's complete content at once on first access (at once). Depending on the file size an access may result in a delay.
`2`	Prevents that on file encryption the protected application is generally able to write data to the hard drive (vie only). Here all writing to a file is prevented.
`4`	Decrypts the content of very huge files (e.g. large MPG3 files with a size of 500 MB) used with file encryption (read and decrypt on demand). In this mode, by default, writing back is deactivated.

cae	Activates instantly the detection that a *CmStick* has been removed from the PC ('plug-out') (*CodeMeter* only). If the connection to a *CmStick* should fail or the license cannot be accessed, you can assign a reasonable number of "ignores" allowing the end-user to continue working without a license access.
G `<l>`	Activates Anti-Debugging mechanisms (*ANTI DEBUGGING CHECKS, ADC*). `<l>` covers the options `[0, 127]` When applying more than one Anti-Debugging mechanism (ADC), you can combine options with "or".

Option	Description

1	Checks whether a debugger is attached to your application. In the case a debugger is detected, the application does not start.
2	Checks additionally for Kernel debugger programs, e.g. "SoftICE". In the case a debugger is detected, the application does not start (ADC2).
4	Checks in an extended search for debugger programs running parallel to your applications. Cracker tools, such as *ImpREC* are detected. In the case a debugger is detected, the application does not start (ADC3).
8	Checks for all debugger programs. Then no debugger programs are allowed, i.e. also within developer environments (IDE), e.g. *Visual Studio*, *Delphi*. In the case a debugger is detected, the application does not start (ADC4).
16	Locking of the license entry and thus of the hardware when a debugger program has been detected (ADC5).  Using this option requires that the developer has programmed the *CmStick* with a Firm Access Counter. The Firm Access Counter is located at the Firm Item level of a *CmStick*. This counter allows you to control whether a Firm Item can be used for encryption and decryption operations. By default, the FAC is deactivated and has a value of `65535 (0xFFFF)`. It can be programmed to any other value.  The *CmStick* is locked when the FAC has a value of `0`. The owner / end-user of the locked *CmSticks* must contact the software vendor for unlocking codes. How and how often unlocking is granted depends on the vendor's policy.
32	Adds a mechanism to the application preventing the attachment of a debugger program to the application at runtime (generic debugger detection)

	(ADC6).
64	Detects whether the application is to be started in a virtual machine and prevents this (ADC7).

G<l> Activates Anti-Debugging mechanisms (*ANTI DEBUGGING CHECKS, ADC*).

<l> covers the options [0, 17]

When applying more than one Anti-Debugging mechanism (ADC), you can combine options by "or".
The default setting for <l> is 17.

Option	Description
1	Checks with a simple Debugger check (ADC1).
16	Locking of the license entry and thus of the hardware when a debugger program has been detected (ADC5).

Using this option requires that the developer has programmed the *CmStick* with a FIRM ACCESS COUNTER. The FIRM ACCESS COUNTER is located at the FIRM ITEM level of a *CmStick*. This counter allows you to control whether a FIRM ITEM can be used for encryption and decryption operations.

By default, the FAC is deactivated and has a value of 65535 (0xFFFF). It can be programmed to any other value.

The *CmStick* is locked when the FAC has a value of 0.

The owner / end-user of the locked *CmSticks* must contact the software vendor for unlocking codes. How and how often unlocking is granted depends on the vendor's policy.

G<l> Activates Anti-Debugging mechanisms (*ANTI DEBUGGING CHECKS, ADC*).

<l> covers the options [0, 7]

When applying more than one Anti-Debugging mechanism (ADC), you can combine options by "or".
The default setting for <l> is 7.

Option	Description
0	Applies no Anti-Debugging mechanism. Default setting when **-cag** is not specified.
1	Checks for the detection of the JVMPI (Java Virtual Machine Profiler Interface). JVMPI can be used to manipulate the Java virtual machine sending messages to the native code. In particular, the event JVMPI_EVENT_CLASS_LOAD_HOOK may be used to intercept the original byte code of the actual class. Activating this option prevents this interception.
2	Checks for manipulation of callback functions, i.e. the access to objects of other classes is checked.
4	Checks the Java Virtual Machine for Java version 6 (1.6).

L Limits the automatic encryption to specified areas.

M Adds the menu items **'Control'** und **'About'** to the application's system menu.

R <t>,<m>
Adds a runtime check to the automatic encryption.
The check occurs every <t> seconds. The default setting is 300 seconds (5 minutes).
<m> specifies how often the end-user is able to ignore a failed check (threshold).

S<p>
Specifies the size of the protected application to be encrypted. You enter the length, in percent, anywhere from 0 to 100%.
The default setting is 75 percent.

T (t)(,u)
On each start of the application, a CERTIFIED TIME update is triggered. Then the application starts regardless of whether the update was successful or not, and writes it into the

CmStick.

Then the application starts when the time difference between the CERTIFIED TIME and the system's PC-Time is not greater than <t> specified in hours.

<u> specifies the valid time span in hours within which the difference between the CERTIFIED TIME and the system time is allowed to range without a new CERTIFIED TIME update (*CodeMeter* only).

<t> has to be equal or greater than <u>.

V Adds a virus check to the automatically encrypted application using a checksum.

Z Saves the time when the encryption was performed within the protected application. Then the application runs only when the PC time is older than this encryption time.. Requires at least *CodeMeter* 4.10.

-CC[[H],[I],[M],[Q],[R],[S],[X]]	Sets compatibility parameters.

[H] Prevents all global hooking in a protected application.

[I] Allows to use the protected application in a way that the added protection does not change the eventual existing loading sequence of DLL files. This replaces the option "-ccm" no longer required.

[M] States that the protected application is to load the library `wibucrt32/64.dll` to avoid problems in the loading sequence by the library `msvcr*.dll`.

Q Clears license use for the protected applications not when WM_QUIT is called but with the call of ExitProcess().

[R] Deactivates the renaming of sections.

[S] Specifies that all licenses must locate in the same *CmStick* connected to the same PC as it was the case with the first license found. .

[X] States that also so-called mixed-mode assemblies are protected. This allows to encrypt .NET assemblies which can not be encrypted using *AxProtector for .NET*. Next to Win32 also Win64/x64 mixed-mode assemblies can be protected using the native *AxProtector*.

The library `wbcor32/64.dll` is required to allow running the protected assembly.

-CD[C][H](K([CA]|[CM]|[WK] Fx[Py])	
Encrypts a file 1:1 file and adds a header holding encryption information. These files can automatically decrypted by an automatically encrypted application.	
C Applies the filename extension from the `*.wbc` file.	
H Specifies that the license access is kept open when the player closes the file, and a handle is kept open. This option is valid for single, separate files.	
K Specifies the used copy protection system: `ca` used *CodeMeterAct* `cm` uses *CodeMeter* `wk` uses *WibuKey*.	
Fx Specifies the FIRM CODE (x) required for the encrypted application to be able to open the encrypted file.	
Py Specifies the PRODUCT CODE (y) required to open the protected application. The FIRM CODE must be previously set. More than one FIRM CODE - PRODUCT CODE pair can be set.	
-CI[H][N][D] Encrypts explicitly defined source code fragments within the executable file to be used with *IxProtector*.	
-CIH Defines that WupiXXX functions are not dynamically interfere in the *IxProtector* process (hooking).	
-CIN Defines that no error messages are displayed when an error occurs.	
-CID Encrypts within executable files explicitly defined source code areas in order to use them with *IxProtector*.WUPI now is also supported on Mac and Linux (Option –cid). Currently, a dynamically loaded variant is applied. 	
-CML<n> Excludes methods from encrypting which are shorter than <n> bytes. The default value is `10`. On specifying a value of `0` the feature is deactivated.	

-CP<l>	Installs a cleaning mechanisms deleting all created files and registry entries on exit of an application in the case the applications has been started from a *CmStick*.
	<l> defines that all deleted entries are written to a log file.

-CK<n>	Buffers the RID key of the application for <n> seconds into the cache memory.
	<n> may have values between 0 and 255.

-E[A(C|I|R)],[E(C|I|R)],[F],[M],[T], [U(S(C|R)[n]|R(C|R)[n]|I)])	
	Defines additional checks while encryption and decryption operations take place.
A	Activates an ACTIVATION TIME check (*CodeMeter* only).
C	Checks when the Product Item Option ACTIVATION TIME exists.
I	Ignores the Product Item Option ACTIVATION TIME (*CodeMeter* only).
R	Requires the Product Item Option ACTIVATION TIME.
E	Activates an EXPIRATION TIME check.
C	Checks when the Product Item Option EXPIRATION TIME exists.
I	Ignores the Product Item Option EXPIRATION TIME (*CodeMeter* only).
R	Requires the Product Item Option EXPIRATION TIME.
F	Activates the decrement of the FIRM ACCESS COUNTER (*CodeMeter* only).
M	Requires a set MAINTENANCE PERIOD
T	Enforces an CERTIFIED TIME update after turning on.
	This option requires an activated EXPIRATION TIME.
U	Activates an UNIT COUNTER check and the counter decrementing by the specified value <n>.

S	Checks and decrements at the start of the protected application only.
SC	Checks whether the Product Item Option UNIT COUNTER exists.
R(S)	Checks and decrements on each runtime check. The option **r** includes the option **s**.
I	Ignores the Product Item Option UNIT COUNTER (*CodeMeter* only).
R(R)	Requires the Product Item Option UNIT COUNTER.

<n> specifies the decrement. The default setting is 1.

`eurr2` activates a required UNIT COUNTER on each runtime check and decrements it each time by the value of 2.

-RIDx[,y]	Specifies the number of RID variants (x) and traps (y). If RID=0 is set automatically then the default va of 256 is used.
-RIDIXx[,y]	Specifies the number of RID variants (x) and trap variants (y) when using *IxProtector* (WUPI). If RID is set to a value of 0, automatically the default value (64/8) is applied.
-G[o,l]] [:"Marker",]]	Excludes the specified range from the encryption.
<o>	Defines the exclusion at the beginning of the range,
<l>	Defines the length of the range to be excluded (only 32 64). **"Marker"** identifies the text marker within the source code identifying the beginning of the range to be excluded from the encryption.
-FW	Sets the minimum Firmware version in the encryption.

-W[C[t]]E[t]][P][U[c]]	Specifies the threshold of issued warnings.

C[t] Specifies the threshold <t> in hours for the CERTIFIED TIME.

E[t] Specifies the threshold <t> for the EXPIRATION TIME.

P[t] Activates a warning when the USAGE PERIOD has not yet been activated.

U[c] Specifies the threshold <c> for the UNIT COUNTER.

9.4 Runtime Options

-I	Specifies that the exception handling for plugin DLL files is used.
	This option exclusively works with DLL files. When the plugin is loaded and no copy protection system linked, the plugin does not closes the complete application, and does not issue error messages.

-L:xx	Specifies the language of the user-defined message texts.

- cn: sets Chinese
- de: sets German
- fr: sets French
- jp: sets Japanese
- us: sets English (default)

-M[A][C][E] [I][S][T][U][W[C|P|T|U]]:<msg>	
	Specifies the output text of user messages of the protected application.

<msg> holds the string for the desired event.

Line breaks, inverted comma, tabs, etc. can be specified in the output text. Type "**\n**", "**\r**", or "**\t**", "\" in the string at the desired position.

A Specifies the application name which is transferred to the server and displayed in *CodeMeter WebAdmin.* No standard name is set. If this option is not set, the internal name of the executable file is used.

C Holds the text displayed in the system menu item 'About'.

I Holds the text displayed when the required driver of the copy protection systems is not installed.

E Holds the text when an error has occurred.

S Holds the text displayed while the protected application is started.

T Holds the text displayed when the date of the EXPIRATION TIME has been reached.

U Holds the text displayed when the UNIT COUNTER has reached a value of `0`.

WC Holds the text displayed when the time difference between CERTIFIED TIME and SYSTEM TIME is too big.

> This option requires the activated option **–WC<t>**.

WP Holds the text displayed on application start when an existing USAGE PERIOD has not yet been activated.

WT Holds the text displayed when the end of the EXPIRATION TIME or USAGE PERIOD is pending.

WU Holds the text displayed when the UNIT COUNTER soon is reaching the value of `0`.

-U[:file name]	Calls the user-defined Message DLL.	

Calls the file `UserMsgXX.dll` where XX stands for an optional country placeholder [Us, Sa, Cn, Dk, Nl, Fr, De, Gr, It, Hu, Jp, Ko, Br, Es, Se, Tw].

`[:file name]`When specifying the file name the user-defined Message DLL holds the name `FileNameXX.dll` where XX stands for an optional country placeholder [Us, Sa, Cn, Dk, Nl, Fr, De, Gr, It, Hu, Jp, Ko, Br, Es, Se, Tw }

(Project types  only).

-U[:file name]	Calls the user-defined message assembly `UserMsg` when this assembly exists.	

When `[:file name]` is specified the implemented message assembly hold the name `[:file name].dll`

> Specify the name without a file extension!

-U[:file name]	Calls the specified class for the message	

	handling.
	 The class must be a secondary class to the `com.wibu.xpm.MessageHandler`. For example, `com.wibu.xpmSwingMessageHandler` as default standard message handler of the runtime package.
-**UI**	Implements the message assembly inline configured by an `*.ini` file.
ANF	Specifies the text displayed when an assembly is not found.
	 Default setting: `The assembly "#requiredassembly#" could not be found.`
-**PROBING:**<Name>	Specifies the path information where assemblies can be found.
	 The input format is either separated by '**;**', or specification of the name of an `app.config` file.
-**SNK**[]:<Name>	Specifies the Strong Name key for the assembly, and use it for signing the assembly.
f	Signs the assembly with the key pair defined in the file <Name>.
n	Signs the assembly with the key pair defined in the key container <Name>.
-**TRAP[1[:n]]**	Inserts hacker traps into the protected Assembly.
-TRAP1[:n]	Adds approx. n% methods to the encrypted Assembly which will lock the *CmStick* on the next decryption process.
	 The default setting for n has a value of 10.
-o[:file name]	Specifies the path and the name of the encrypted destination file `[:file name]`.

9.5 Java-specific Options

-ja:"params"

Specifies arguments transferred to the Main Class at runtime.

-jb:<number>

Activates or de-activates a special error exception handling.

Contact Wibu-Systems support for more details

-jcl:<ClassLoader>

Specifies an alternative WIBU ClassLoader.

Currently, the following ClassLoader are available:

`ClassLoader`	ClassLoader derived from `java.net.URLClassLoader`
`DelegateClassLoader`	ClassLoader derived from `java.lang.ClassLoader`

-jd:vmin[-vmax]

Specifies the used minimum (and maximum) Java version.

The version must match the format as specified in the system property '*JAVA.VERSION*'. The final number can be left out.

-jd:1.4-1.5.0_04 allows the runtime versions from 1.4 to Java 5 Update 0 Maintenance 4.

-jh: `[a|e|n]`

Hides or renames encrypted classes.

a	Renames all classes according to the pattern '`<MyClass>.class.wibu`'.
e	Renames only encrypted class names according to the pattern '`<MyClass>.class.wibu`'.
	This corresponds to the default setting.
n	Renames no encrypted classes.

-jm:<Main-class>	Specifies the starting Main Class.
-jn:[p|s|t]	Activates the native class load process.
	 This process requires an intervention into the source code of the application.
	p Uses JVMPI (Java Virtual Machine Profiler Interface).
	s Uses the Java 6 module (not yet supported).
	t Uses JVMTI (Java Virtual Machine Tool Interface).
-jl:[w|b|t]:<list>	Specifies which classes are encrypted.
	w Whitelist: all classes matching this list are encrypted.
	b Blacklist: all classes matching this list are not encrypted.
	<list> Holds the complete class or package name, parts of the name (fragments). The list items are separated by '**:**'.
	 '-jlw:com.wibu.:de.wibu.MainClass'
-jo[[a:<jars>],[lsf],[e:[e]]:<list>	
	Specifies output options.
a:	**<jars>** Adds the specified jar file to the output.
	 `-joa:CodeMeter.jar,WibuKey.jar` adds the contents of those jar files to the output specified by **-o**.
l:	Lists the license information of an encrypted jar file.
S:	Separates the output in two different jar files. The WIBU runtime classes and the source jar files are not merged.
	 This option is recommended for servlets, or when you encrypt several jar files in a

	project to save space. The created `WibuXpm4JRuntime.jar` file must be manually added to the class path.
`f:`	Separates the output in three different jar files. This is an extension of the option **`-jos`** and creates a file with the name `WibuXpm4JO<outputfile>.jar` holding a few options only.
`e:` `[e]`	Specifies which files are excluded from the output. `[e]`specifies the file to be excluded, e.g. `com/wibu/xpm/encrypted.`
`-jvm:`	Considers the selected option for the virtual machine. Some Java runtime settings require a separate internal handling for the project type '*AxProtector* for Java'. The following option is available: `<server>`: The Java virtual machine is started with the server.
`-jx`	Exits the application using the `System.exit()` call after the 'Main-Class' main method has returned a value.

9.6 Operational Options

`-!`	Creates a command file(`*.wbc`).
`-v`	Activates the verbose mode. In the case of .NET use **`-vn`.**
`-#[File]`	Prints the logging to the specified `File`. This option exists next to automatic output to the `AxProtector.log`.[`//...\user`].
`-?` or **`-h`**	Shows the options in commandline mode.
`@cmds.wbc`	Specifies a `*.wbc` file holding parameters for the automatic encryption of an executable file.

10 *IxProtector* and the *Software Protection API* – WUPI

The *IxProtector* protection technology allows you to define 'real' single segments (modules, functions) in the source code when developing an application, encrypt them, and then link them to license entries at runtime by using index-based placeholders. The *Software Protection API* WUPI (*WIBU Universal Protection Interface)* assists you in this process.

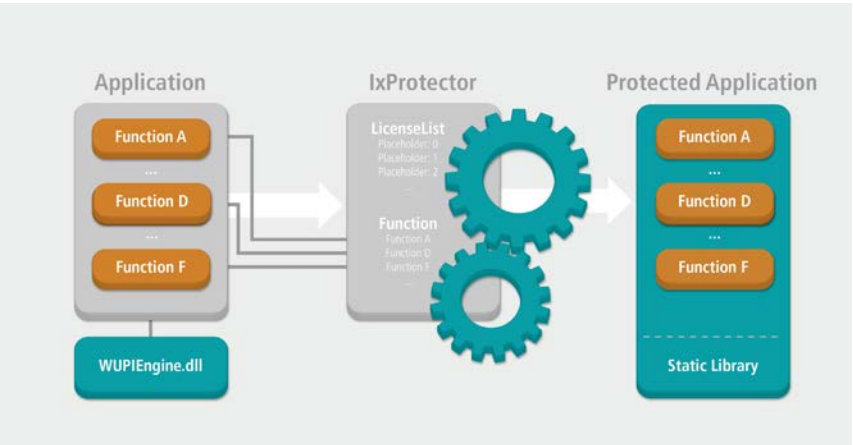


Figure 79: *WUPI*: Individual software protection

The interaction of *IxProtector* and WUPI is suited for the following application areas:

- Protecting and activating single modules of an executable file, i.e. modular software protection , using specified function and license lists.
- Integrating individual license queries. You freely define where and when.
- Encrypting 'real' code fragments to increase the level of security.
- Implementing pay-per-use functionalities, i.e. decrementing counters for specified software actions.
- Specifying which kind of anti-debugging measures *AxProtector* applies at which point in time.
- Simultaneously implementing for all copy protection systems (*WibuKey*, *CodeMeter, CodeMeterAct*) while still making future changes possible.
- Accessing a license allocated by *AxProtector* for further use in the *WibuKey Core* API.

- Reading of password-protected data previously stored in a HIDDEN DATA FIELD at runtime of the protected application.

A writing functionality is currently not implemented but pending for the near future.

Using WUPI you can realize:

- Easy-to-accomplish protection: available for many programming languages with a one-time implementation in the same executable file without recompiling your source code.
- Constantly updated protection: continuing the security-related revisions and improvements of AxProtector functionalities without requiring changes to your source code.

11 WUPI Functions

Overall, the lean and effective *Software Protection API* WUPI provides the following functions.

Access API: Allocating and Releasing Licenses

WUPIALLOCATELICENSE()	This function allocates a license **(LicenseList)** for the selected copy protection system.
	Return value
	`TRUE (1)` when the function was successfully executed, otherwise FALSE `(0)` when an error has occurred.
WUPIFREELICENSE()	This function releases a license **(LicenseList)** for the selected copy protection system.
	Return value
	`TRUE (1)` when the function was successfully executed, otherwise FALSE `(0)` when an error has occurred.
WUPIGETHANDLE()	This function returns the actual native handle of the license **(LicenseList)**.
	Return value
	The actual native handle of the license is returned. Otherwise `(0)` is returned when an error has occurred.

WUPIENCRYPTCODE()	This function encrypts a function **(Function).**	**Encryption and Decryption API**
	Return value	
	`TRUE (1)` when the function was successfully executed, otherwise `FALSE (0)` when an error has occurred.	
WUPIDECRYPTCODE()	This function decrypts a function **(Function).**	
	Return value	
	`TRUE (1)` when the function was successfully executed, otherwise `FALSE (0)` when an error has occurred.	
WUPICHECKDEBUGGER()	This function **(LicenseList, Level)** checks whether the protected application runs within a debugger, a debugger runs on the system, or a Kernel debugger is installed (see page 176).	**Security API**
	Return value	
	`TRUE (1)` when the function has detected a debugger attack, otherwise `FALSE (0)`.	
WUPICHECKLICENSE()	This function checks a license **(LicenseList)** for the selected copy protection system.	
	Return value	
	`TRUE (1)` when the function was successfully executed, otherwise `FALSE (0)` when an error has occurred.	
WUPIDECREASEUNITCOUNTER()	This function **(LicenseList, Units)** decrements a *UNIT COUNTER* in the specified license by the defined number of units.	
	Return value	
	`TRUE (1)` when the *UNIT COUNTER* was successfully decremented, otherwise	

	WUPIDECREASEUNITCOUNTER()	This function **(LicenseList, Units)** decrements a *UNIT COUNTER* in the specified license by the defined number of units.
		`FALSE (0)`.
Information Query	**WUPIQUERYINFO**()	This function returns information on an entry **(LicenseList)** or on a *WibuBox*.
		Return value
		When the queried value exists it is returned. In the case an error occurred or the queried information does not exist, `-1` is returned including a related error code.

Except for the functions **WUPIENCRYPTCODE()** and **WUPIDECRYPTCODE()** referring to function lists, all other functions relate to license lists.

Reading of data

When using the *Software Protection API* WUPI at runtime of the protected application you have the option to read raw data you previously saved to the *WibuKey*, for example, to use the saved data for the program functionality. Reading previously saved data is provided by the WUPI functions WUPIREADDATA or WUPIREADDATAINTEGER.

For *WibuKey* the raw data is stored in the PROTECTED EXTENDED MEMORY which is available in all WibuBoxes with a "+" at the end, e.g. *WibuBox/U+* or *WibuBox/RU+* (for more information see page 17). Each page of the EXTENDED MEMORY has 32 bytes. There are 256 pages in the PROTECTED EXTENDED MEMORY, so you can store up to 8192 bytes.

Accessing the PROTECTED EXTENDED MEMORY requires a signature access. The derivation of the signature is calculated under the assumption that:

- the first ENTRY of the *WibuBox* (where the PROTECTED EXTENDED MEMORY is bound to) has no added data (ADDED DATA, LIMIT COUNTER or EXPIRATION DATE) or
- existing added data has no dependencies (`serial, programming counter, data`).

Currently, the settings for reading raw data need to be specified in the `*.wbc` command file in the section of the license definition. The license definition area of the `*.wbc` files then looks as follows:

```
[License WK1]
Type=WibuKey
...
UserData=read ; necessary, activates the mode for reading data
```

WupiReadData `(int iLicenseList, int iOffset, void* pvData, unsigned int cbData);`

This function reads raw data which has been previously stored at a specified location from the *WibuBox*.

This function can be used for all programming languages working with pointers, i.e. special variable holding memory addressed.

For the other programming languages the function WupiReadDataInteger is provided (see page 195).

Parameter	Description
`iLicenseList`	specifies the number of the license list index.
`iOffset`	holds in number of bytes the offset from the start of the data block.
`pvData`	holds the data array to be filled.
`cbData`	holds the number of bytes for `cbData`.

Return value

Number of bytes stored in `pvData`.

In the case the return value of `0` call WupiGetLastError to obtain more detailed information(see page 196).

WupiReadDataInteger`(int iLicenseList, int iOffset);`

This function reads raw data which has been previously stored at a specified location from the *WibuBox*.

The data is read 2 bytes at a time.

This function can be used for all programming languages.

For programming languages working with pointers, i.e. special variable holding memory addressed, Wibu-Systems recommends the function WupiReadData (see page 195).

Parameter	Description
`iLicenseList`	specifies the number of the license list index.
`iOffset`	holds in number of bytes the offset from the start of the data block.

Return value

The return value has a size of 4 bytes. It is separated in 2

WUPIREADDATAINTEGER(int iLicenseList, int iOffset);	

	upper bytes holding status flags for error and message handling and 2 lower bytes holding the data.
	The upper 2 bytes may, for example, hold the following values:
	`#define WupiRDError (0x80000000)`
	`#define WupiRDMoreDataAvail (0x40000000)`

Error API

WUPIGETLASTERROR()	This function returns the actual defined error code of the actual defined license type **(LicenseList)**.
	Return value
	`wibu::UpiErrorNoError (0)`→ no error occurred.
	`wibu::UpiErrorNoDefaultLicense (-1)` → no default license is set, i.e. the application is not additionally automatically encrypted.
	`wibu::UpiErrorLicenseNotFound (-2)` → the specified index for a license could not be found.
	`wibu::UpiErrorFunctionNotFound (-3)` → the specified index for function could not be found.
	`wibu::UpiErrorRuntimeTooOld (-4)` → the drivers of the copy protection system in use are outdated.
	`wibu::UpiErrorDebuggerDetected (-5)` → a debugger attack had been detected.

Table 2: WUPI Functions – Overview

12 Individual Software Protection using WUPI: an Example

Index-based Placeholders

In the programming sequences of your application, the *Software Protection API* WUPI links software protection mechanisms and license queries with parts of the source code using index-based placeholders. In the following, excerpts from the sample application "WupiCalculator" show you how to create modular software protection in combination with a "pay-per-use" license model via WUPI.

You will find the full example after installing the *WibuKey* SDK for respective programming languages in the directory "`%Program Files%\WIBU SYSTEMS\AxProtector\Samples\IxProtector\…\WupiCalculator Index`".

The basic task in this example is to create, for end-users, a copy-protected application. The use of the application requires matching entries in the *WibuBox*. In order to use the different modules, the end-user will need additional matching entries in his/her *WibuKey*.

The creation comprises five single steps: (1) defining modules (2) creating index-based license and function lists, (3) programming license entries, (4) integration into the source code, (5) encryption of the application.

12.1 Definition of Modules

The licensing model of the "WupiCalculator" example is modular by design. When the customer buys a "basic license" calculator, s/he is able to use the basic arithmetic functions only. The memory function is not billed separately and each use of a higher mathematical function (angle, power, factor, binomial) reduces a counter by a defined value.

Table 3 below summarizes the modules and the pay-per-use options, and assigns a unique ID for each module.

Defining which modules are active for which customers can be accomplished by the production and sales departments. This provides for the clear separation of the development of license strategies and models on the one hand, and the actual implementation in the software development department on the other hand.

Modules	Per use	ID
Basic license	reduce by 1	1
Memory function	0 (no decrement)	2
Angle function	reduce by 5	3
Power function	reduce by 2	4
Factor function	reduce by 10	5
Binomial function	reduce by 10	6

Table 3: WUPI Calculator – Definition of Modules

12.2 Placeholders in *IxProtector* License and Function Lists

This information, on the designed license model, is sufficient for the subsequent connection between *IxProtector* and the WUPI function calls, made from within the source code for creating index-based placeholders.

Table 4 summarizes the required information you need for the later completion of the license and function lists.

Module	LIMIT COUNTER per use	Exaxt function name
Basic license	reduce by 1	`CalcSimpleOperation`
Memory function	0 (no decrement)	`OnButtonCalcMemClear`
Angle function	reduce by 5	`CalcAngle`
Power function	reduce by 2	`CalcPower`
Factor function	reduce by 10	`CalcFactorial`
Binominal function	reduce by 10	`OnButtonCalcBinomial`

Table 4: WupiCalculator – Overview

To create the placeholders, proceed as follows:

❶ Activate *IxProtector* in *AxProtector* in the "Advanced Options" input window.

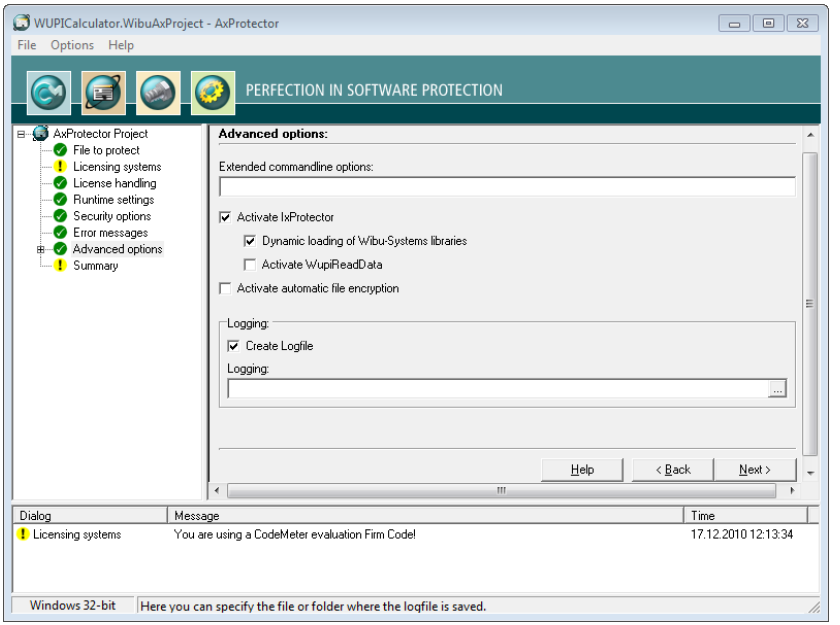


Figure 80: Activate *IxProtector* within *AxProtector*

With this option *IxProtector* finds the related source code segments, and encrypts them before *AxProtector* wraps a protection envelope around the compiled application.
If you want to use *IxProtector* without *AxProtector*, select the project type "*IxProtector* only" (see page 160).
Unless you have a special reason, Wibu-Systems recommends using *IxProtector* within *AxProtector*.

❷ Navigate to the "License Lists" input window.

License List

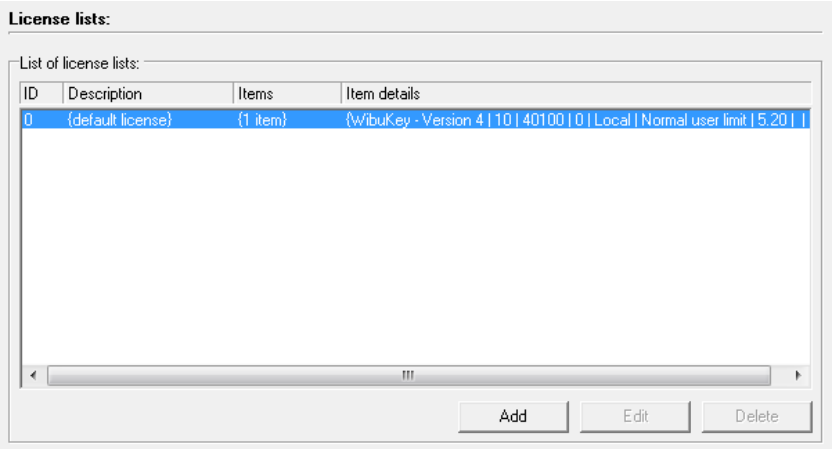


Figure 81: *IxProtector* – License Lists

License lists allow to summarize licenses with different license elements (copy protection system, FIRM CODE, USER CODE etc.) into single entries. A single entry may hold several license elements.

Entries in license list can hold all Wibu-Systems copy protection systems (*WibuKey*, *CodeMeter* and *CodeMeterAct*). You can later change an assignment to single copy protection systems without altering the source code. However, then the changed license information has to become part of the encryption.

The license list entry of 0 describes the license to which *AxProtector* refers.

---

❸ Click on the **"Add"** button.

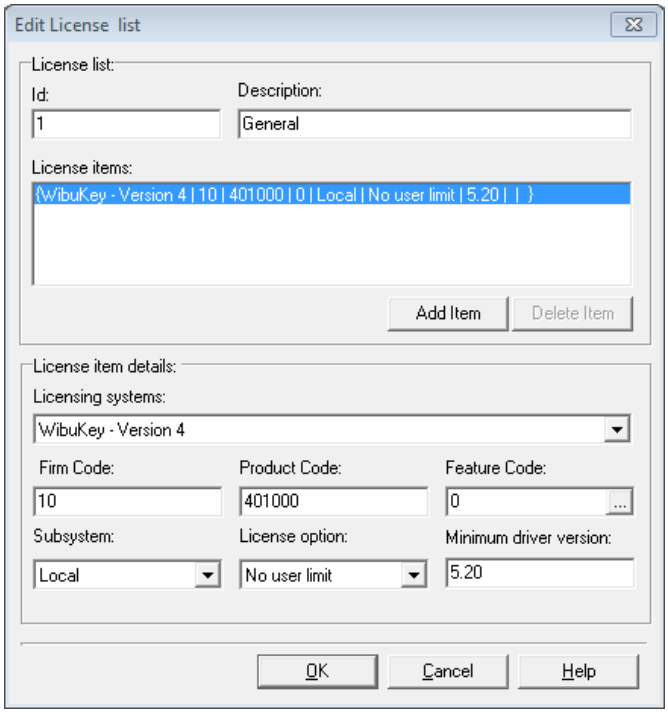


Figure 82: *IxProtector* – License Lists - Item

In the "WupiCalculator" example, *IxProtector* prompts you with an index-based placeholder of ID=1 for the Basic License. Transfer the necessary data of Table 4, page 198 for the Basic License, i.e. FIRM ITEM 10 and PRODUCT ITEM 1608 with a *FEATURE MAP* bit value of `0` (FEATURE CODE).

For the allocation of the Basic License by *AxProtector* specify the license option *NO USER LIMIT*. With it the subsequent use of the single modules will not allocate an additional license.

For the optional modules to be activated for users of "WupiCalculator", transfer the further information of Table 4, page 198. After that, Figure 83 should display the following license list.

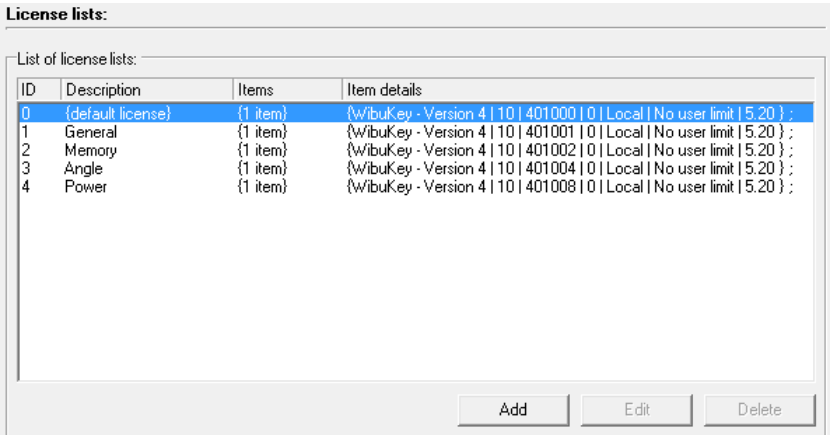


Figure 83: IxProtector – Specified License List

The "ID" column now holds the index-based placeholders which will be addressed by the WUPI license calls (see page 194).

Function List

❹ Navigate to the "IxProtector" input window.

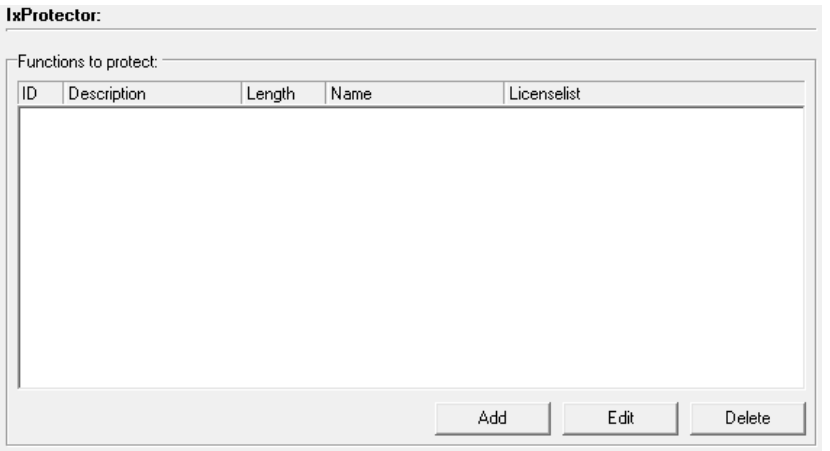


Figure 84: *IxProtector* – Function List

The *IxProtector* options define the functions to be protected, and allow for the assignment of functions to the license list entries you defined above.

❺ Click on the **"Add"** button.

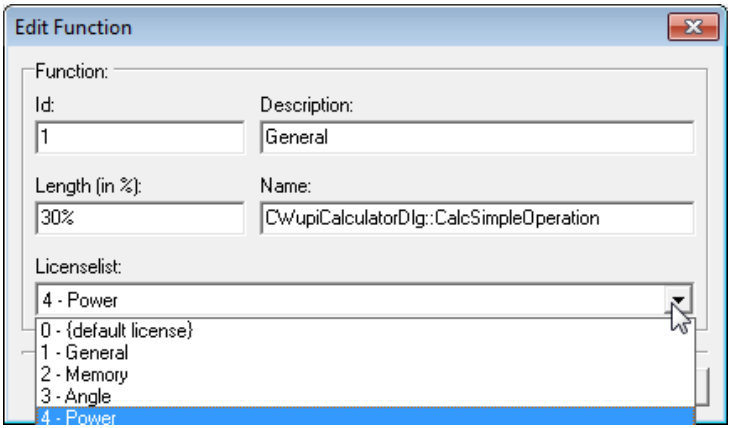


Figure 85: *IxProtector* – Function List - Item

In the "WupiCalculator" example, *IxProtector* prompts you with an index-based placeholder of ID=1 for the Basic Function. Transfer the necessary data from Table 4, page 198 for the Basic Function.

The description of the function in field **"Name"** must exactly match the name which is later addressed in the source code by the index-based placeholder. Overloaded functions are not supported.

Specify the length of the array to be encrypted for the function.
You enter the length, in percent, anywhere from 0 to 100%. If you want this number to represent percentage, you must enter the percent character (%). Alternatively, you are able to specify the length by number of bytes. Then *AxProtector* automatically calculates the length.

Then select the license list to which the function is to be assigned.

For the other functions to be protected within "WupiCalculator", transfer the additional information from Table 4, page 198. After that, Figure 86 should display the following function list.

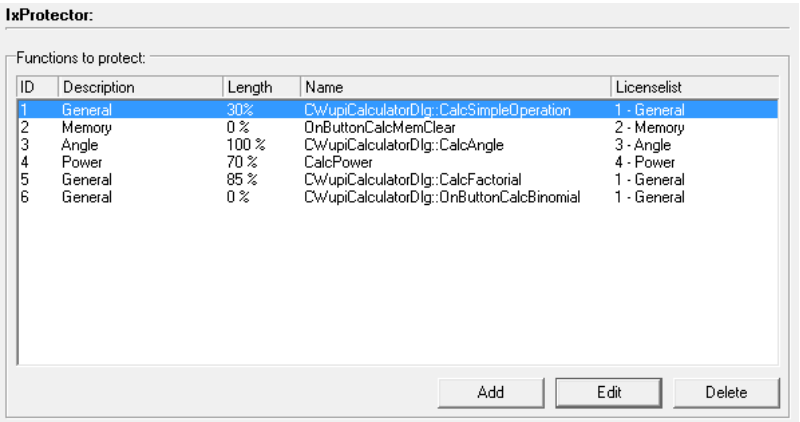


Figure 86: *IxProtector* – Specified Function List

Now all required data has been completed in *IxProtector*, and all index-based placeholders are defined.

12.3 Programming of the *WibuBox*

After defining "WupiCalculator" protection options using *IxProtector*, you now have to transfer the license entries into the *WibuBox*. For this use **WkList32** (see page 209).

The programming covers:

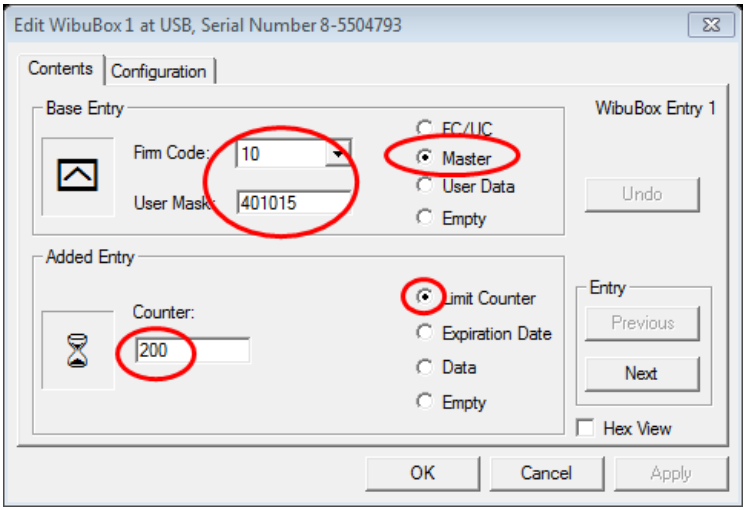
- a FIRM CODE of `10` and a USER MASK of `401015` for the USER CODES `401000` through `401008`,
- a UNIT COUNTER reading of `200` units.

WkList32

In **WkList32** proceed with the following steps:

❶ Click in the programming interface.

❷ Complete the FIRM CODE, MASTER, and USER MASK and program a **LIMIT COUNTER** of 200.



❸ Click the **"Apply"** button to program this license entry into the connected *WibuBox*.

12.4 Integration into the Source Code

Subsequently, you insert the WUPI functions into the source code where the software protection mechanism or license queries have to be applied.

The WUPI functions now will refer to the index-based placeholders you created before in *IxProtector* by completing the license and function lists.

When developing, you first have to integrate a Dummy-DLL which holds the WUPI function calls.
Depending on the operating system (Windows 32- or 64-bit), use the `WupiEngine32.dll` or `WupiEngine64.dll` functions. When protecting .Net applications, use the `WupiEngineNet.dll`.

These files are located in the directory "`%Program Files%\WIBU-SYSTEMS\AxProtector\DevKit\lib`".

In the following, some source code samples show how some of the WUPI functions have been realized for the "WupiCalculator" example.

The source code file for "WupiCalculator", in the programming language C++ (and also for the other languages) you can find as `WupiCalculatorDlg.cpp` in the directory "`%Program Files%\WIBU`

SYSTEMS\AxProtector\Samples\IxProtector\C++\WupiCalculatorIndex". All following samples are taken from this implementation file.

WUPICHECKLICENSE

For the "Memory" function in the "WupiCalculator" example no encryption is designed. Only a query is applied, i.e. a license check. The source code below shows the use of this WUPI function for the deletion of the memory.

```
void CWupiCalculatorDlg::OnButtonCalcMemClear()
{
        if ((eError != meLastClick) && WupiCheckLicense(2)) {
                mdftMemoryValue = 0;
                m_memoryValue = mdftMemoryValue;
                UpdateData(FALSE);
        }
}}
```

Figure 87: *WUPI* – License Check

WUPIDECRYPT and WUPIENCRYPT

For the functions "Angle", "Power" and "Factor" in the "WupiCalculator" example an encryption is designed The source code below shows the use of this WUPI function for the function "Angle".

Note that after decryption using **WUPIDECRYPT()** you re-encrypt the respective part of the source code using **WUPIENCRYPT()**!

WUPIDECREASEUNIT COUNTER

Moreover, using this function means decrementing the UNIT COUNTER by 10.

```
void CWupiCalculatorDlg::OnButtonCalcAngleSine()
{
        BeginWaitCursor();
        if (eError != meLastClick) {
                //decrease counter by 5
                if (!WupiDecreaseUnitCounter(1, 5)) {
                        WupiErrorOccured(WUPIERROR_COUNTER,
"UnitCounter decrease on sine operation failed!");
                        EndWaitCursor();
                        return;
                } else {
                        SetTicks();
                        // Wupi: Decrypt the CalcAngle Function
                        if (!WupiDecryptCode(3)) {
                                WupiErrorOccured(WUPIERROR_DECRYPT,
"Decryption of CalcAngle");
                                EndWaitCursor();
                                return;
                        } // if

                        UpdateData(TRUE);
                        m_calcOut = CalcAngle(ANGLE_SIN, m_calcOut,
(meAngleBasis == eRad ? true : false));
                        meLastClick = eSimpleOperator;
                        UpdateData(FALSE);

                        // Wupi: Encrypt the CalcAngle Function again
                        if (!WupiEncryptCode(3))
```

```
                                WupiErrorOccured(WUPIERROR_ENCRYPT,
"Encryption of CalcAngle");
                }
        }
        EndWaitCursor();
}
```

Figure 88: *WUPI* Encryption and UNIT COUNTER

When programming the *WibuBoxes*, the UNIT COUNTER was initially set to `200` units. The source code below shows how to implement this WUPI function for querying license information when turning on the calculator using **WUPIQUERYINFO()**.

WUPIQUERYINFO

```
void CWupiCalculatorDlg::OnButtonOn()
{
        meLastClick = eNum;
        BeginWaitCursor();
        if (WupiCheckLicense(1)) {
                unsigned long iTicks = WupiQueryInfo(1,
WupiQIUnitCounter); // read the UnitCounter
                if (-1 == iTicks) {
                        // error on WupiQueryInfoId
                        WupiErrorOccured(0, "No start possible without
counter! Aborting.");
                        OnButtonOff();
                        return;
                }
```

Figure 89: Query *WUPI* information

After inserting the WUPI functions you compile your application.

12.5 Encryption using *AxProtector*

After compiling "WupiCalculator" you encrypt using *AxProtector* with *IxProtector* simultaneously activated. *IxProtector* now replaces the placeholders by entries of the license or function list.

IxProtector is integral part of *AxProtector*. You can alternatively use it alone by choosing the project type "IxProtector only", or additionally in combination with *AxProtector*. When integrating *IxProtector* in *AxProtector*, *IxProtector* searches the respective source code parts and encrypts them before *AxProtector* encrypts the completed application. But remember when using the project type "IxProtector only", a higher security level is provided, since the Dummy-DLL is replaced by static code. This DLL is not used later when the application is executed.

Part V Programming *WibuBox* Hardware

WibuBox is programmed by connecting it to one of the applicable interfaces (table below). Normally, no additional hardware is necessary to program a *WibuBox*. The one exception is for external programming of the *WibuBox/CI*. In this case the programming adapter WIBU-PROG and a power pack are required. They supply the higher voltage which is necessary for the programming of the *WibuBox/CI*. The programming adapter WIBU-PROG is connected directly to the printer interface, and the *WibuBox/CI* is then connected to the adapter.

When programming *WibuBox* hardware it is possible to

- add,
- change or
- delete entries.

The table shows which *WibuBox* hardware exists for different interfaces and computers:

Computer	Interface	*WibuBox*
IBM PC compatible systems	LPT	*WibuBox/P* / *WibuBox/RP*
IBM PC compatible systems, UNIX systems and latest Apple Power PC	USB	*WibuBox/U* / *WibuBox/RU*
UNIX Workstations any computer with RS232	COM	*WibuBox/ST*
Notebooks	PCMCIA	*WibuBox/M*

To program a *WibuBox* the software developer needs:

- His/her own FIRM CODE for which only s/he possesses the usage rights. This means that a sequence in the `WKFIRM.WBC` file for one or more specific FIRM CODES must exist.
- Furthermore the corresponding FIRM SECURITY BOX (FSB). This needs to be connected to one of the computer's LPT to LPT3 interface, USB-interface or ADB interface, or to a **`WkLAN`** server. The FIRM SECURITY BOX allows the use of `WKFIRM.WBC` and the stored FIRM CODE. A `WKFIRM.WBC` without FSB is useless.

It is very important to store the `WKFIRM.WBC` and the Firm Security Box (FSB) in a manner not allowing unauthorized access.

The `WKFIRM.WBC` file for a specific FIRM CODE is shipped in conjunction with a license contract per floppy disk. Subsequent re-orders, e.g. in the case of non-readable floppies, are **not** possible. It is recommended to make security backups of the `WKFIRM.WBC` file.

The FIRM SECURITY BOX and the `WKFIRM.WBC` are available with the *WibuKey* Firm License.

The programming of the *WibuBox* hardware can either occur with the interactive **`WKLIST`** application or from the commandline variant of **`WKCRYPT`**.

`WKLIST32` is available

- as 32 bit version **`WKLIST32.EXE`** for Windows 95/98/Me/NT/2000.
- as Macintosh version **`WkMacList`**

Because all of these programs have identical functionality the following text will use the term **`WKLIST32`**.

In order to facilitate the handling of multiple *WibuBox* hardware, we recommend inserting a normal LPT or COM extension cable between the PC interface and the *WibuBox* to be programmed on the developer's machine.

1 WkList - Interactive programming

The GUI program **`WKLIST`** is the most simple and fastest tool to change the contents of a *WibuBox*. After starting **`WKLIST`** a user interface appears in the following manner:

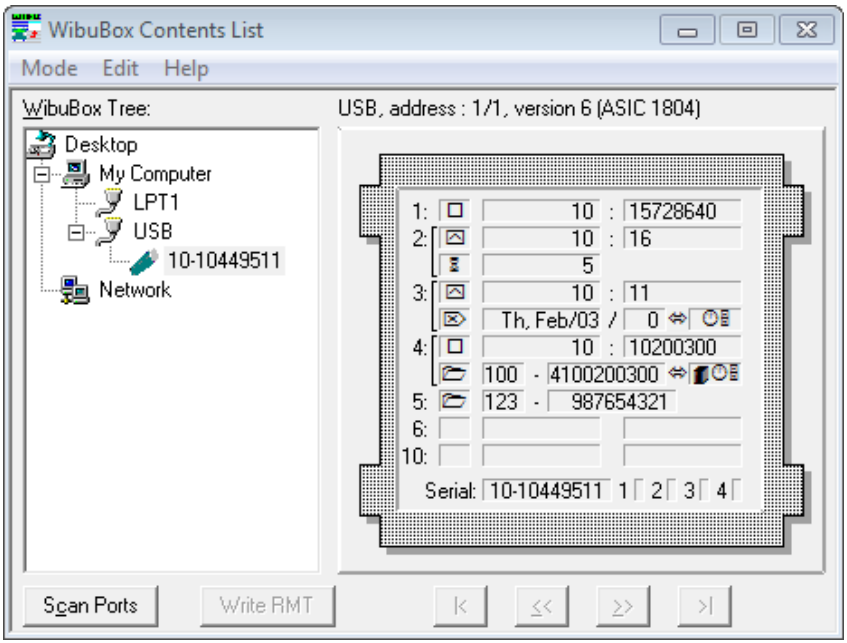


Figure 16: Contents of a WibuBox displayed with WKLIST.EXE

In the line below the menu above the *WibuBox* symbol the port address and number of the *WibuBox* selected from the *WibuBox* Tree is displayed.

On the left side the local station and the available ports are displayed in the *WibuBox* Tree. If a **WkLAN** server process is running on the network, a double click on the network symbol shows the available **WkLAN** server and *WibuBoxes* in the network. By default the first *WibuBox* at the first port is selected. If a port symbol marked by a plus sign is clicked, the *WibuBoxes* attached to this port will be listed.

On the right side all of the contents of the selected *WibuBox* are displayed in graphical form. One specific *WibuBox* entry can be selected with the keyboard or the mouse. The number of the selected entry will be marked by a dotted frame.

WibuBox symbols:

BASE ENTRIES (see page 18)	
☐	specifies a BASE ENTRY (FIRM CODE/USER CODE) (entry 1, 2, 4, 6, 10).
🖂	specifies a MASTER CODE entry (entry 3).

BASE ENTRIES (see page 18)	
	without a bracket specifies a USER DATA Entry (entry 5).
ADDED ENTRIES are bundled by a bracket to a BASE ENTRY (see page 18)	
	specifies a LIMIT COUNTER Entry (bundled with entry 2).
	specifies an EXPIRATION DATE (bundled with entry 3).
	specifies an ADDED DATA Entry (bundled with entry 4).
	The serial number: The programming sequence depends on the serial number of the *WibuBox* and differs from *WibuBox* to *WibuBox* even for the same entry.
	The internal programming counter. The programming sequence depends on the internal programming counter of the *WibuBox* and changes with every programming operation.
	The data contents: The programming sequence depends on the contents of the data entry or the EXPIRATION DATE.

A BASE ENTRY can be stored in any entry of the range from 1 to 10. ADDED ENTRIES can be bundled only with a BASE ENTRY in the range from 1 to 5 and are stored in the entry with index of 5 more than the corresponding BASE ENTRY. The encryption sequence of an EXPIRATION DATE or an ADDED DATA Entry can depend on the *WibuBox* serial number, the internal programming counter and data contents.

1.1 Storing and modifying BASE ENTRIES

To change an entry in the *WibuBox*, use the 'Edit Box' command in the 'Edit' menu. The command is only enabled if a valid *WibuBox* is displayed. As an alternative, the desired entry can be double-clicked with the mouse. After one of these actions, a dialog box similar to the picture beside appears.

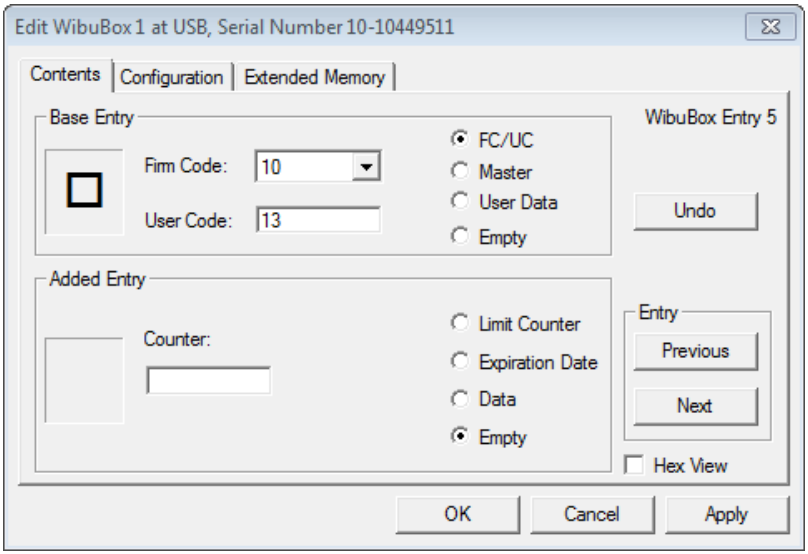


Figure 17: Edit dialog of `WKLIST.EXE`

The Contents page contains the symbols of the entry to change. If the entry is empty there are no symbols. The FIRM CODE pulldown list contains all FIRM CODES for which a software developer has the user rights. USER CODE can be any value between `0` and `16,777,215`.

To change a Base Entry from FIRM CODE/USER CODE type to a USER DATA or Master Entry, use the corresponding radio button in the right part of the dialog box. To erase an entry choose the 'Empty' button.

Multiple *WibuBox* entries can be changed with one call of the Edit dialog box. For navigation between the entries there are the 'Previous' and 'Next' buttons available. Each preliminary done modification can be removed again by the 'Undo' button. No physical programming actions occur before the 'OK' button is pressed. Pressing the 'Cancel' button instead ignores all entered actions. The *WibuBox* contents are not changed.

Values may be entered in decimal or hexadecimal notation. To specify a hexadecimal value, use the C notation with the leading "0x" symbol. Normally all valid entered values are displayed in decimal. All values can alternatively be displayed in hex by clicking the 'Hex View' check box in the right bottom corner of the dialog box.

1.2 Storing and modifying Added Entries

To modify an Added Entry or to add such an entry to a Base Entry, select a valid Base Entry or create a new Base Entry. Select this Base Entry by opening the Edit dialog box or by navigating with the 'Previous' and 'Next' buttons.

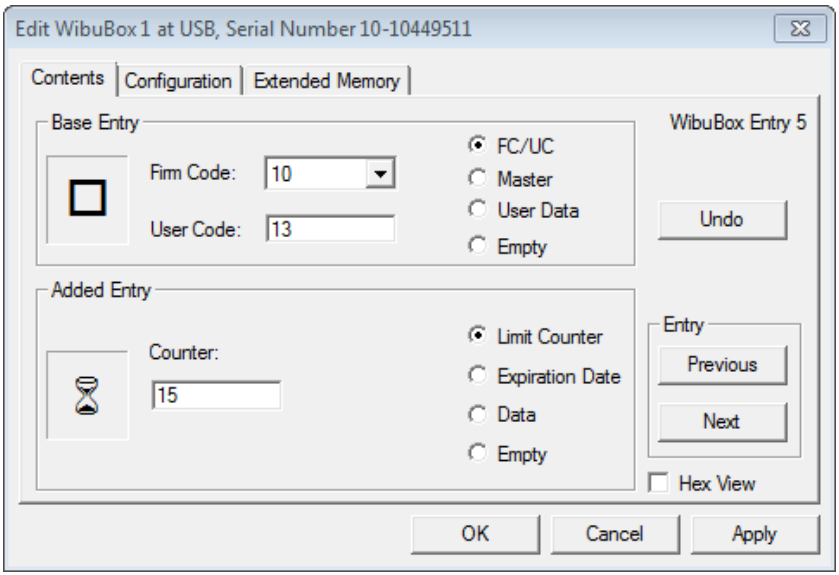


Figure 18:Edit dialog for an Added Data Entry of `WKLIST.EXE`

The bottom half of the 'Edit' dialog box displays the current contents of the Added Data Entry. The dialog controls change depending on the type of entry. To change an Added Entry use the radio buttons on the right side. They include an 'Empty' button to remove existing Added Entries. Added Entries are changed by setting the corresponding parameters and by defining a modify dependence for the encryption sequence.

1.3 Storing and modifying User Data and Protected Data

To read or write USER DATA or PROTECTED DATA the **WKLIST** application can be used. The handling of the two 8 kByte blocks of data is done by binary files, which can be created or modified using a editor like **notepad.exe**.

```
30313233 34353637 38394142 43444546 0123456789ABCDEF
4768696A 6B6C6D6E 6F707172 73747576 Ghijklmnopqrstuv
0A0D0A0D 0A0D0A00 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
```

Fig.: Hex dump view

WKLIST therefore provides a page to display and to read and write that kind of binary content file.

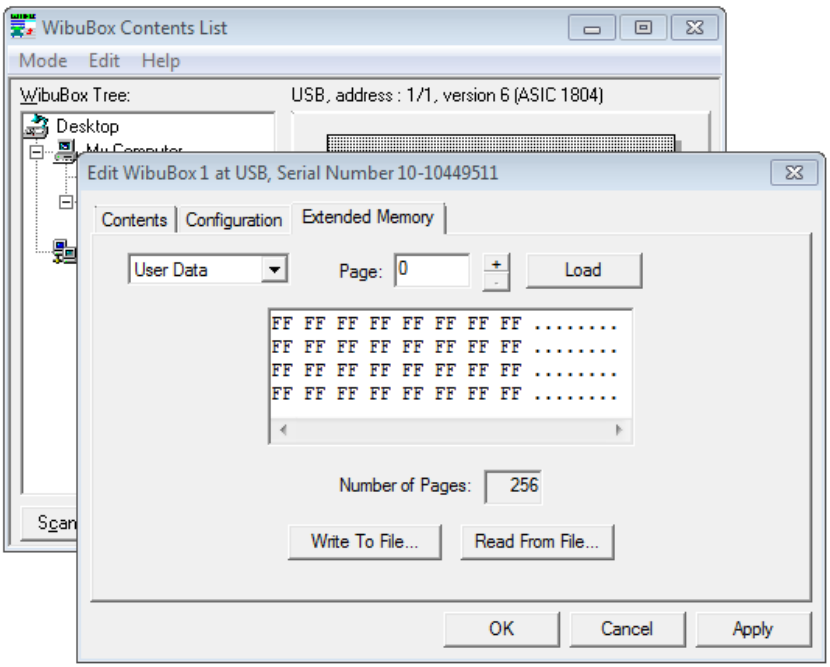


Figure 19: Page 0 of the USER DATA

All 32 bytes of a memory page are displayed in a hexadecimal view. Using the Page option all available memory pages can be displayed page by page. The area selection specifies which memory area is accessed for file operations.

To modify parts of USER DATA first the content must be written to a file. Then the binary file can be edited. To update the memory content the modified binary file must by read by **`WKLIST`**. This method can be used for both kinds of memory areas. To modify PROTECTED DATA also a FIRM SECURITY BOX is needed.

The Additional Memory only is available at *WibuBoxes/P* and */U* of version 6 and at *WibuBoxes/RP* and */RU* of version 7.

1.4 Modifying *WibuBox* configuration attributes

The **WKLIST** application permits the interactive changing of configuration attributes for *WibuBoxes*. Such configuration attributes depend on the interface and on the type of *WibuBox*.

To modify the configuration attributes of a specific *WibuBox*, use the Configure Box... command in the Edit menu of **WKLIST** or choose the Configuration page in the application window. The following dialog box appears in which the configuration attributes can be edited:

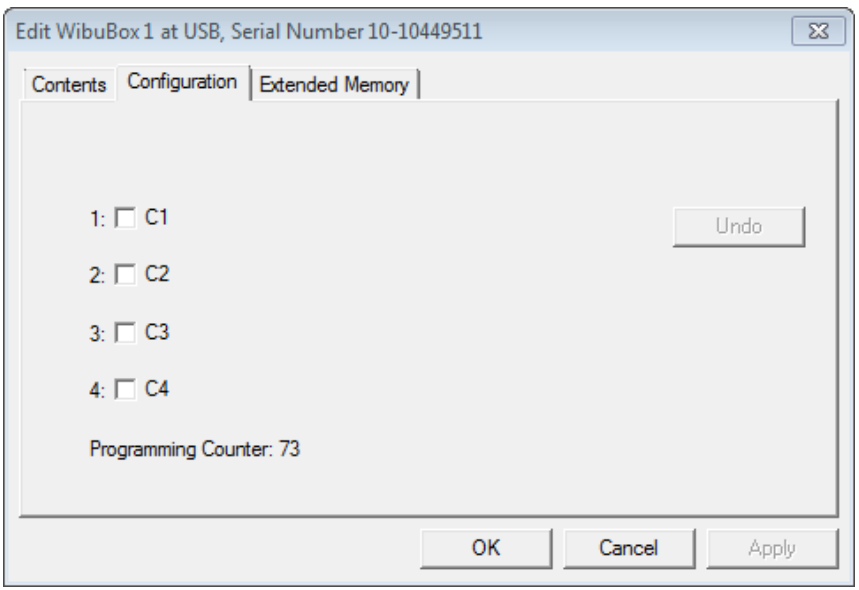


Figure 20: Edit configuration dialog of `WKLIST.EXE`

Some *WibuBox* types, for example the *WibuBox/M*, have protected configuration attributes which generally cannot be changed. Such attribute values are displayed by the **WKLIST** program in an inactive state and cannot be modified.

Unnamed configuration attributes C2, C3 and C4 are not used by the *WibuBox* nor the driver. They can be set and reset individually and can be read from the application. The configuration attributes can be changed with **WKU**, **WKCRYPT**,

WKLIST and the *Control Panel Applet*. The user can change it and it should therefore not be used for security data.

2 WKCRYPT - Commandline programming

The tool **WKCRYPT**, that is refered to in this chapter, is obsolete and replaced by the *AxProtector*. The *AxProtector* is used for the automatic protection and does not support the commandline programming of *WibuBoxes*. So **WKCRYPT**must be still used in the future for this reason. A detailed description of **WKCRYPT**can be found in the external document *WibuKey - Guide for* **WKCRYPT**.

The **WKCRYPT** application allows *WibuBox* entries to be programmed, modified or erased with commandline options. **WKCRYPT** addresses the connected WibuBox with a special programming sequence.

The syntax for the programming commandline of *WibuBox* hardware is

WKCRYPT [*Options...*]

2.1 Programming *WibuBoxes*

/PA*BoxPort*

This option sets the port with the specified port value as the selected *WibuKey* interface for the subsequently used programming or listing commands. The port value must be entered directly behind the option without blank. Whether the specified interface may be used by *WibuKey* is established as soon as access for programming or listing is required. The selected interface remains valid until another is specified.

Port addressing

```
WKCRYPT /PA1
WKCRYPT /PAL2
WKCRYPT /PA#2
WKCRYPT /PAC1
WKCRYPT /PA#9
```

The first three examples address LPT1 to LPT3, the following two address COM1 and COM2. The complete description of the port value specification for the WibuKey interfaces may be found in the appendix B11.

/PC [M][*UserCodeNumber*]

This option deletes (*program clear*) all undesired entries in the selected *WibuBoxes* at the specified port with the specified FIRM CODE and USER CODE. Is no USER CODE specified, all entries with the current FIRM CODE get erased. If the option is specified as **/PCM**, a MASTER ENTRY will be cleared.

Deleting *WibuBox* entries

```
WKCRYPT /F10 /PAL2 /PQ3 /PC13
```

erases all BASE Entries 10:13 from the last three *WibuBoxes* at LPT2.

```
WKCRYPT /F10 /PAL2 /PQ3 /PC
```

erases all entries with FIRM CODE 10 from the same *WibuBoxes*.

/PE [*BoxEntryNumber*]

This option specifies a fixed *WibuBox* entry in the range of 1 to 10 which is used for the next *WibuBox* programming operations. The specification of a fixed entry permits the storing of an identical FIRM CODE/USER CODE entry several times in one *WibuBox*. If the **/PE** option is not specified or without parameter, the first free entry within a *WibuBox* is used to program a new entry. For the replacing, extending or modifying of a BASE ENTRY or an ADDED ENTRY, the first matching entry is used.

/PN

The **/PN** option (*program new*) programs a new entry into the selected *WibuBoxes* at the selected interface. The FIRM CODE and USER CODE must previously be specified by the options **/F** and **/U**.

Programming *WibuBox* entries

```
WKCRYPT /F10 /U13 /PN
```

programs the new entry 10:13 in the last *WibuBox* at LPT1.

```
WKCRYPT /F10 /U13++ /PQ3 /PN /U22 /PN
```

programs the last three *WibuBoxes* at LPT1 with the entries 10:13, 10:14, and 10:15. Each *WibuBox* is programmed with the entry 10:22.

***/PQ**Number*

This option sets the number (*program quantify*) of selected *WibuBoxes* which may be used at the selected interface for programming and listing commands. A number between `1` and `20` is allowed. The default value is `1`. Whether the specified number of *WibuBoxes* can be selected, is only established when the *WibuBoxes* are called for programming or listing. In case that a number of *WibuBoxes* is selected, they are counted from the free end of the plugged series. Their listing and programming though begins with the *WibuBox* nearest to the interface.

Listing *WibuBox* entries

```
WKCRYPT /PAL2 /PQ5 /L
```

lists the last five *WibuBoxes* at LPT2.

```
WKCRYPT /F10 /U20++ /PAC1 /PQ3 /L /PN /L
```

programs new entries with FIRM CODE and USER CODE `10:20`, `10:21`, and `10:22` in the last three *WibuBoxes* at COM1. The contents of the *WibuBoxes* are listed before and after the programming.

***/PR [M]**UserCodeNumber*

This option (*program replace*) replaces all entries in the selected *WibuBoxes* at the selected interface with the specified FIRM CODE/USER CODE. The current FIRM CODE/USER CODE must previously be specified by the options **/F** and **/U**. Should an entry with the current FIRM CODE and USER CODE already exist in one of the *WibuBoxes*, this option will be ignored. This which means that the old entry **will not** be cleared. Should the entry with the specified USER CODE not exist, a new entry will be generated.

If the option **/PRM** is specified, a MASTER ENTRY/USER CODE will be replaced.

Changing *WibuBox* entries

```
WKCRYPT /F10 /U13 /PR22
```

substitutes the entry `10:22` in the last *WibuBox* at LPT1 by `10:13`.

```
WKCRYPT /F10 /U13 /PAL2 /PQ3 /PR22
```

concerns the last three *WibuBoxes* at LPT2.

/PS [:*NetworkServerName*]

This option specifies a WkLAN server that is used for the next programming or list commands. To be able to change the contents of a *WibuBox* connected to a server, the right options for WkLAN must be set on the Server page of the *WibuKey Control Panel Applet* (see page 276). For the commandline option **/PS** the server name must be specified directly after the colon. It must be a valid IP address or a station name that is known by a DNS or WINS server. If all required communication demands are installed, the programming of a *WibuBox* over the Internet is possible.

The **/PS** option without argument specifies the use of the local subsystem.

Specifying a WkLAN server

```
WKCRYPT /PS:MyServer /L
WKCRYPT /PS:"My Server" /L
WKCRYPT /PS:"200.200.200.5" /L
WKCRYPT /PS /L
```

/PXC*CountValue*

This option adds a LIMIT COUNTER to the current selected BASE ENTRY. Such an entry is selected via the **/PE** option or via the specified FIRM CODE/USER CODE or FIRM CODE/ MASTER CODE specification.

LIMIT COUNTER setting

```
WKCRYPT /F10 /U13 /PXC1234
WKCRYPT /PE3 /PXC10
```

/PXD[C][S][D] [*HighValue*-]*LowValue*

This option bundles an ADDED DATA ENTRY to the selected BASE ENTRY. Such a BASE ENTRY is selected via the **/PE** option or via the specified FIRM CODE/ USER CODE or FIRM CODE/MASTER CODE. If the *HighValue* is not specified, it is set to 0.

With the **C**, **S** and **D** sub-options it can optionally be specified on which options the programming sequence depends:

- S (symbol) is specified if the programming sequence depends on the serial number of the *WibuBox*. This means that such a programming sequence can change only one specific *WibuBox*.
- C (symbol) is specified if the programming sequence depends on the programming counter in the *WibuBox*. This means that the programming can be done only for this counter value. After each programming of the *WibuBox* the programming counter is increased.
- D (symbol) is specified if the programming sequence depends on the data to be stored. Different data produce different programming sequences. This is an additional security mechanism that prohibits that during the remote programming the transfer value can be changed.

Within the **/PXD** option any combination of these flags can be used. The suboption **D** can only be used from *WibuBox* version 4 on.

ADDED DATA entry setting

```
WKCRYPT /F10 /U13 /PXD10
```

adds the ADDED DATA ENTRY 0-10 to the BASE ENTRY 10:13.

```
WKCRYPT /PE3 /PXD4-3
```

adds the ADDED DATA ENTRY 4-3 to the third BASE ENTRY.

```
WKCRYPT /F10 /U13 /PXDC10
```

programming sequence depends on the programming counter.

```
WKCRYPT /PE3 /PXDSCD4-3
```

The ADDED DATA ENTRY depends on all modify options.

/PXT[C][S][D]*DayDelta*|*DateValue*

This option adds an EXPIRATION DATE to the current selected BASE ENTRY. Such a BASE ENTRY is selected via the **/PE** option or via the specified FIRM CODE/USER CODE or FIRM CODE/MASTER CODE. For the date specifications *DayDelta* and *DateValue* see appendix B10.

The **C**, **S** and **D** sub options have the same meaning as described for the option **/PXD**. They are used when changing an EXPIRATION DATE Entry with the **/PMT** option.

The suboption **D** can only be used from *WibuBox* version 4 on. Using a version 4 or newer *WibuBox*, it is strongly recommended to use this option for security reasons.

EXPIRATION DATE setting

EXPIRATION DATE setting

```
WKCRYPT /F10 /U13 /PXT10
```

programs an EXPIRATION DATE to 10 days.

```
WKCRYPT /PE3 /PXTS30
```

programs an EXPIRATION DATE to 30 days. This entry can only be changed with a programming sequence depending on the serial number of the *WibuBox*.

```
WKCRYPT /F10 /U13 /PXT2011Dec31
```

sets the EXPIRATION DATE to the end of 2011.

```
WKCRYPT /PE3 /PXTD2011Mar31
```

uses the end of the first quarter of 2011 as EXPIRATION Date. The programming sequence to change this last entry depends on the value of the new EXPIRATION DATE to be programmed.

2.2 Modifying *WibuBox* contents

***/PMC**CountValue*

This option modifies the LIMIT COUNTER of the selected BASE Entry. Such a BASE ENTRY is selected via the **/PE** option or via the specified FIRM CODE/ USER CODE or FIRM CODE/MASTER CODE.

Changing a LIMIT COUNTER

Absolute value:

```
WKCRYPT /F10 /U13 /PMC1234

WKCRYPT /PE3 /PMC10
```

Delta value:

```
WKCRYPT /F10 /U13 /PMC+          +1

WKCRYPT /PE3 /PMC+=10            +10

WKCRYPT /PE3 /PMC—               -1

WKCRYPT /F10 /U13 /PMC-=316      -316
```

WKCRYPT checks such delta expression to ensure that the LIMIT COUNTER value is within the range of `0` to `1,048,575`. The relative changes are also supported by the Remote Programming. The end user needs the **WKU** or **WKU32**

application of version 2.20 or later. Older versions will report a communication error when programming a relative value. *WibuBoxes*, from version 4 on, support the relative changes of Limit Counters internally to improve the security for Remote Programming. For *WibuBoxes* of version 3 this is simulated by the *WibuKey* software.

/PMD [*HighValue*-]*LowValue*

This option modifies the ADDED DATA Entry of the currently selected BASE Entry. Such a BASE Entry is selected via the **/PE** option or via the specified FIRM CODE/ USER CODE or FIRM CODE/MASTER CODE. The ADDED DATA Entry must already exist. If the *HighValue* is not specified, it is not changed.

Changing an ADDED DATA entry

```
WKCRYPT /F10 /U13 /PMD10
WKCRYPT /PE3 /PMD4-3
```

/PMT*DayDelta*|*DateValue*

This option modifies the EXPIRATION Date of the current selected BASE Entry. This is especially useful to update the EXPIRATION DATE at the end user via Remote Programming. The arguments of this option are similar to the arguments of the **/PXT** option except that for **/PMT** there are no delta expressions available.

Changing an EXPIRATION DATE

```
WKCRYPT /F10 /U13 /PMT10
WKCRYPT /PE3 /PMTS30
WKCRYPT /F10 /U13 /PMT2011Dec31
```

/PU [*HighValue*-]*LowValue*

This option creates, modifies or clears the specified USER DATA Entry. Such a BASE Entry must always be selected via the **/PE** option. If the specified location is empty, a new USER DATA Entry is generated. Otherwise it is modified. If the *HighValue* is not specified, this value is not changed. For a new entry it is set to 0. If neither a *HighValue* nor a *LowValue* is specified, the selected USER DATA Entry is cleared.

Changing a USER DATA entry

```
WKCRYPT /PE1 /PU10-21
WKCRYPT /PE3 /PU4-3
WKCRYPT /PE9 /PU
```

2.3 Listing *WibuBox* contents

/L [*BoxPort*]

This option lists the contents of all selected *WibuBoxes* at the selected port on the monitor. A *WibuBox* is represented by ten fields for the ten entries. Each field contains the respective FIRM CODE and USER CODE or a FIRM Code and MASTER CODE. If an entry exists the serial number of the *WibuBox* is additionally listed.

The option displays the current *WibuBox* contents. The specification of a **/PC**, **/PR** or **/PN** option between two **/L** option calls, will display the *WibuBox* contents before and after the changes.

Listing *WibuBox* contents

```
WKCRYPT /PAC4 /PQ5 /L
```

lists the last five *WibuBoxes* at COM4.

```
WKCRYPT /F10 /U13 /L /PN /L
```

programs the entry `10:13` into the last *WibuBox* at LPT1. With the two **/L** options, the *WibuBox* contents will be displayed before and after the programming.

2.4 Modifying *WibuBox* configuration attributes

/PF*Configuration name list*

This option permits to modify the configuration attributes of the selected *WibuBoxes*. The configuration names must be specified without space character after the option; each preceding "+" sets the specified option, a "-" resets it. General configuration names (C1, C2, C3 or C4) can be specified. *WibuBox* specific options like **BD** or **UD** for the *WibuBox/P* or **MO** and **TM** for the *WibuBox/SP* are possible parameters.

If no argument is specified, this option displays the current state of the configuration attributes of the selected *WibuBoxes*.

Changing configuration attributes

```
WKCRYPT /PAL1 /PF+UD

WKCRYPT /PAC2 /PF+TM

WKCRYPT /PAL1 /PF+C1-C2+C4
```

2.5 Initializing *WibuBox* hardware

/PI

This option clears the complete contents of the selected *WibuBoxes* for which the user has the permission; other entries cannot be changed. USER DATA and all FIRM CODE/USER CODE or FIRM CODE/MASTER CODE entries for which the user has a `WKFIRM.WBC` file and a FIRM SECURITY BOX (FSB) are cleared by this operation. The use of this option in combination with **/PQ**, clears the contents of more than one *WibuBox*.

Initializing *WibuBox* hardware

`WKCRYPT /PQ3 /PI /L` initializes three *WibuBoxes*

2.6 Output of programming sequences

/PO ([*FormatLetter*][:*DestinationFile*]) | -

The programming sequence **WKCRYPT** transfers to the *WibuBox* can be displayed on the screen or written to a file. This is useful if the sequence is needed later to reprogram a *WibuBox* entry with the *WibuKey API*. The redirection of the sequence output to the screen is activated with the **/PO** option. The physical *WibuBox* programming is then stopped. The generation of a Remote Programming Update File will **not** be stopped by the **/PO** option.

When the **/PO** option is specified without argument the programming sequence will be printed on the screen.

If one of the format letters **A**, **B**, **C**, **D**, **L**, **M**, **N** or **P** follows the **/PO** option a destination file name needs to be specified. **WKCRYPT** writes the sequence according to the format letter to the given file. The format specification is compatible with the **/CO** option. If no format letter is specified the sequence will be written as binary data bytes. For multiple programming operations multiple programming sequences will be written to the same file.

 The **/PO** option is only supported by *WibuBoxes* of version 3 or higher.

The **/PO** mode is cancelled with the **/PO-** option. All following commands are programmed physically in the *WibuBox*.

To be able to use the **/POM** option the **/FM** option must be specified.

3 Remote programming

The principle of remote programming is explained on page 48. For remote programming the software developer and thr customer work together. The *WibuBox* stays at the user's site the whole time. No shipping of hardware is necessary at any time. Remote programming can be realized within a few minutes. See page 57 for the right preparation at the developer's site.

The following table shows the handling of data files.

Software Developer	Action	End-User
Context File: Generating a context file before the first software distribution at the developer's site.	Saving the context file, i.e. with the corresponding serial number at the developers' site.	No action.
Update File: Generating the update file after the order of the customer. It then contains the programming information for the new *WibuBox* contents.	→ Sending the update file to the customer by	Update File Updating the contents of the *WibuBox* interactively in the *WibuKey Control Panel Applet* with the update page. This file is suitable only for one specific *WibuBox*.
Modified Context File: Generating a modified context file together with the update file. This stays at the developer's site and can be used as the context file for changes in the future. The developer always has the actual contents of all *WibuBoxes* documented.	Saving the corresponding serial number at the developer's site.	No action.

If the software developer did not save a context file of the *WibuBox*, s/he delivered or if the context file was lost, remote programming is still possible. See the explanation on page 48.

3.1 Remote programming at the custome

The client can easily generate a remote update file (`*.rtc`) within the Windows Explorer by using the right mouse button:

❶ S/he looks for an appropriate folder within the Windows Explorer and clicks with the right mouse button into the Explorer Window of an open folder.

❷ S/he selects "New | WIBU Control File" from the context menue.

❸ From the following dialog box, s/he selects "Remote Programming Context File" and confirms with 'Create'.

❹ A file `MyWibuBox.wbc` is generated into the same folder.

❺ The client sends this file to the software developer.

❻ With this file, the software developer can now generate a remote update file by using the tools WkList or **`WKCRYPT`**.

❼ This file is sent to the client. By double-clicking the file the client can update his/her *WibuBox*.

3.2 Remote programming using the *WibuKey* symbol

For remote programming the customer has two special pages in the *WibuKey Control Panel Applet* at his disposal. These interactive pages help generate a context file and update a *WibuBox* with the update file. These two steps are described below.

3.2.1 Context file generation

❶ The *WibuKey Control Panel Applet* contains a page called *WibuBox* Context. Select this page to create a context file.

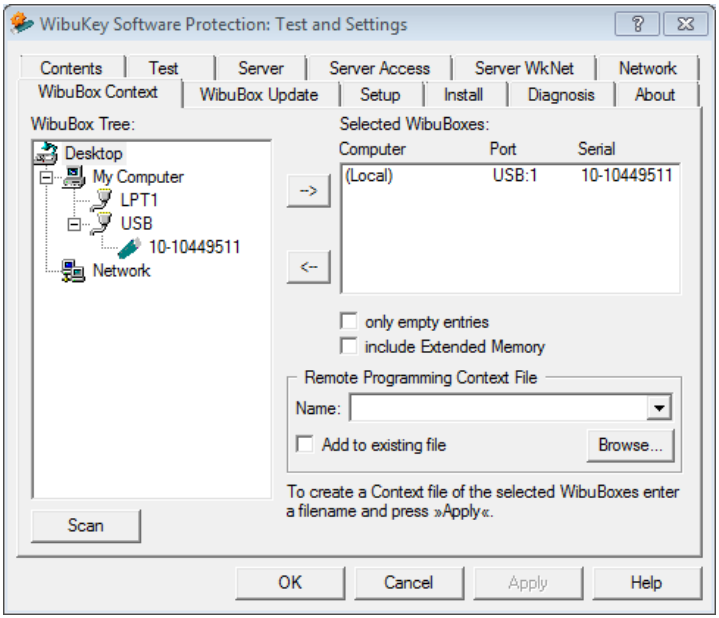


Figure 21: *WibuBox* Context page (*WibuKey Control Panel Applet* Advanced Mode)

❷ Use the *WibuBox* tree on the left side to select the *WibuBox* hardware, which is shown in the 'Selected WibuBoxes' section on the right side. Set the 'only empty entries' option to save the empty entries of the *WibuBox* instead of the contents.

❸ Specify a file name in the 'Remote Programming Context File' area. There is also a button to browse. A remote programming context file always has the file extension `*.rtc`.

❹ Press 'Apply' to create the context file.

3.2.2 Updating *WibuBox* contents

❶ The *WibuKey Control Panel Applet* contains a page called *WibuBox* Update (see page 284). Select this page to change the *WibuBox* contents with the update file.

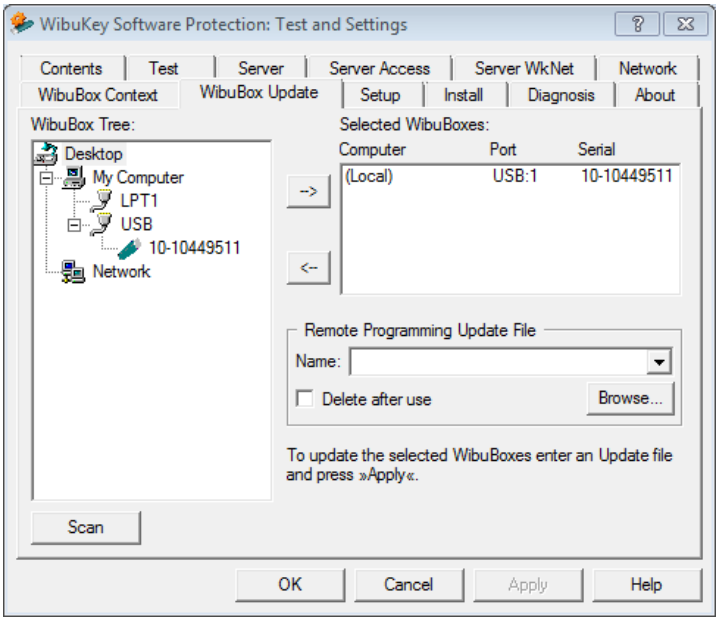


Figure 22: *WibuKey Control Panel Applet*: *WibuBox* Update page

❷ Choose the *WibuBox* to change with the update file. Only one specific *WibuBox* is suitable for the update file!

❸ Browse for the update file with the file extension `*.rtu`. The 'Delete after use' option can be set to delete the update file after the changing of the *WibuBox* contents. An update file is useless after the update is applied, so there is no reason to keep it around.

❹ Press 'Apply' to change the *WibuBox* with the specified update file.

Context file generation and updating *WibuBox* contents is done by `WkMacList` on the Macintosh. For that purpose, the user gets the `WkMacList` application without the file `Wkfirm.wbc` and without FIRM SECURITY BOX.

3.3 Commandline actions at the customer

It is possible to create a context file and update a *WibuBox* with an update file from the commandline. This is realized with the **`WKU`** utility program. Further functions of **`WKU`** are explained on page 293.

WKU can create a remote programming context file for one or more connected *WibuBoxes* with the **REMOTE GET** command:

```
WKU REMOTE GET ALL TO X.RTC
```

copies the contents of all connected *WibuBoxes* to the file `X.RTC`. To minimize the size of this file the ports or the connected *WibuBoxes* which contents should be stored can be selected.

```
WKU REMOTE l1:1 c1 GET ALL TO X.RTC
```

copies the contents of the first *WibuBox* at LPT1 **(l1:1)** and of all *WibuBoxes* at COM1 **(c1)** into the file `X.RTC`.

```
WKU REMOTE l1:2 GET 1,5 TO X.RTC
```

copies only the contents of the entries 1 and 5 of the second *WibuBox* at LPT1 to the remote programming context file.

For adding a new entry only a list of empty entries in the *WibuBox* is necessary. This can be saved with the keyword **EMPTY**:

```
WKU REMOTE * GET EMPTY TO X.RTC
```

The star (*) is a keyword to specify all found *WibuBoxes* and is the same like a space between the keywords **REMOTE** and **GET**.

The created context file can be sent in any form to the software developer. The software developer creates a remote programming update file and sends it back to his customer. This update file contains commands to modify the specific contents of the specific *WibuBox*.

To transfer a remote programming update information to a connected *WibuBox*, the command **SET** of **WKU** is used:

```
WKU REMOTE SET FROM X.RTU
```

searches at all ports for connected *WibuBoxes* which serial number and programming counter value is compatible with the values in the remote programming update File `X.RTU`. An optional list of *WibuKey* ports or specific *WibuBoxes* can limit the update process to a subset of connected *WibuBoxes*.

3.4 Actions at the developer's side

The software developer creates an update file for his customer and optionally a modified context file for himself. The creation of these files is done with the commandline options of the *WibuKey* program **WKCRYPT**.

3.4.1 Remote programming with `WkList`

It is possible in **WkList** (**WkMacList**) to display remote context files and create remote update files in an easy way.

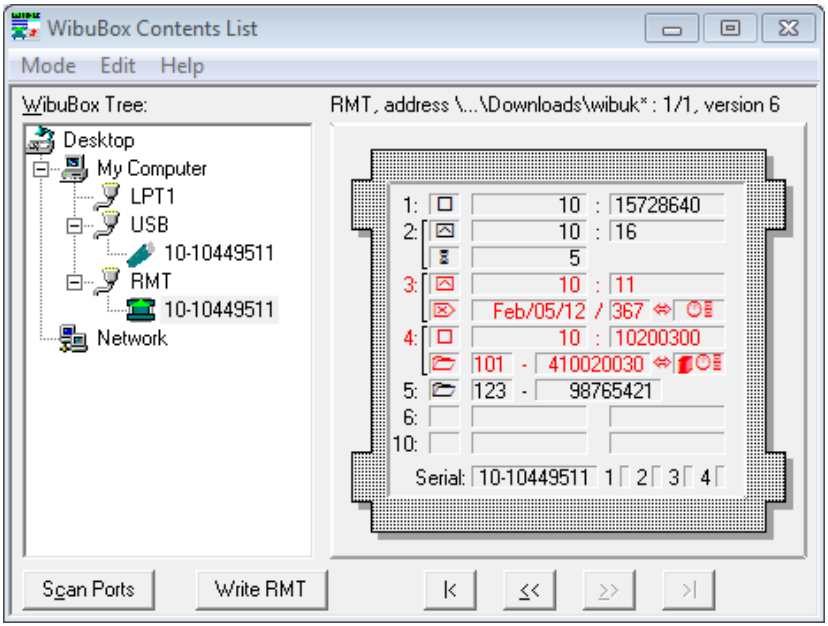


Figure 23: Remote programming with `WkList`

To start the programming select the 'List' Mode (default on startup) and select a `*.RTC` file via the menu 'Edit | Remote Context File…'. The contents of this file is displayed in a virtual port (RMT), the file name is displayed as box address above the contents.

Programming of a remote box is similar to programming a local box. All changes are done in memory until you press the 'Write RMT' button to save the remote programming update File. All changes are marked with red color.

In the sample below, you can see the contents of the `*.RTC` file '`TestKitBox.rtc`' with a modified LIMIT COUNTER entry (entry #2) and a new entry `10:272` (entry #6).

3.4.2 Commandline syntax remote programming (WKCRYPT)

/RC:[*RemoteProgrammingContextFile* [**.RTC**]]
/RM[A]:[*RemoteProgrammingModifiedContextFile* [**.RTM**]]
/RO[P/W]:[*File format of RemoteProgramming Files*]
/RU[A]:[*RemoteProgrammingUpdateFile* [**.RTU**]]

Commandline syntax for remote programming with **WKCRYPT**:

```
WKCRYPT [Option ...]:[ContextFile ...] [Option
...]:[UpdateFile ...] ...
```

3.4.3 Activating remote programming

For each call **WKCRYPT** is entered first. Then it is necessary to activate the remote context reading. The reading of a remote programming context file by **WKCRYPT** replaces the reading of one or more physically connected *WibuBoxes*. The activation is done by the **/RC** option with the remote programming context file as argument. The default file extension is .rtc.

To activate the reading of the remote programming context file, for example demo.rtc, use one of the following commandlines:

Context file creation

```
WKCRYPT /RC:demo

WKCRYPT /RC:demo.rtc
```

The **/L** option lists the contents of a context file. The contents are not changed by this option. It is possible to use this option after the creation of the update file.

Listing context file contents

```
WKCRYPT /RC:demo /L

WKCRYPT /RC:demo.rtc /L
```

3.4.4 Creating an update file

The creation of a remote programming update file by **WKCRYPT** is activated with the **/RU** option. This option has the remote programming update file as argument. The default extend for such a file is `*.rtu`. To cause the creation of the example update file `demo.rtu` use one of the following commandlines:

Update file creation

```
WKCRYPT /RC:demo /RU:demo /F10 /U13 /PN

WKCRYPT /RC:demo.rtc /RU:demo.rtc /F10 /U13/ PN
```

3.4.5 Creating a modified context file

WKCRYPT permits the creation of a modified context file. This file contains the new contents of the *WibuBox* after the reprogramming with the update File. The modified context file is useful for saving the new state of a *WibuBox* at the programmer's site. It simplifies the creation of another remote programming update file in the future. To create a modified context file, use the option **/RM**. The default file extension of such a file is `*.rtm`. The contents of a modified context file are absolutely compatible to the remote programming Context file with file extension `*.rtc`. The two different file extensions are used to avoid that an input context file is overwritten by a created modified context file. For the example, file demo use one of the following commandlines to create a modified context file:

Modified context file creation

```
WKCRYPT /RC:demo /RU:demo /RM:demo /F10 /U13 /PN

WKCRYPT /RC:demo.rtc /RU:demo.rtu /RM:demo.rtm /F10
/U13 /PN
```

3.4.6 File format of remote programming files

/RO[P/W]

WKCRYPT generates and reads RemoteProgrammingFiles (`*.RTC`, RemoteProgrammingContextFile; `*.RTM`, RemotePorgrammingModifiedContextFile, `*.RTU`, RemoteProgrammingUpdateFile) either in the Plain format (=Default value, programs up to version 2.53) or in the `*.WBC` format (version 3.0 or higher). The advantage is that `*.WBC` files have an additional file header and they can be allocated explicitly.

3.4.7 Additional options and tips

To add information to an existing update file the option **/RUA** is used which activates the append mode. A remote programming update file can contain up to 16 programming commands. Please note that replacing an existing entry by a

new entry via the **/PR** option internally needs two programming commands. The option **/RMA** can be used to add information to an existing modified context file.

Changes to the *WibuBox* contents via remote programming are done by any programming or listing operation of **WKCRYPT**. These are options like **/PN**, **/PC**, **/PR**, **/PX**, **/PI** or **/PM**. The **/PF** option cannot be used together with remote programming. This is not really a limitation, because the user can modify the *WibuBox* configuration directly via the **WKU** application. The option **/PQ** is used to read more than one *WibuBox* from a context file or to create update files or modified context files for multiple *WibuBoxes*.

Please note that within one commandline any new setting of the options **/RC**, **/RM**, **/RU** or **/PQ** terminates a started programming sequence. This termination creates the update and modified context file or executes the programming of the local *WibuBoxes*. The reading of the remote programming context file is then started again as well as the creation of the destination files.

Changing entries with remote programming

```
WKCRYPT /RC:demo /RU:demo /F10 /U13 /PMC10
```

creates a remote programming update file `demo.rtu` which changes the bundled LIMIT COUNTER of the entry `10`:`13` to `10` in the *WibuBox* described by the remote programming context file `demo.rtc`.

```
WKCRYPT /PAL1 /RM:save.rtm /F10 /U13 /PN /PXD3-5
```

stores a new entry `10`:`13` with an ADDED DATA Entry 3-5 into the *WibuBox* at LPT1 and saves the remote programming context information of this *WibuBox* in the file `save.rtm`.

Part VI Implementing network protection

This part describes the detailed implementation of network protection (see also on page 38 the network concept and the *WibuKey Network License Management).*

Network Specific Terms

In general, a network consists of a number of clients and one or more servers. The server is distinguished by the fact that files which have been stored on its disk may be called from all clients. The terms *client* and *server* need not necessarily exclude each other. For purposes of simplicity, the following description distinguishes between *client* and *server*.

- A *WibuKey* network client may be any computer on which the *WibuKey* software is installed. Operating systems are Windows NT/2000/XP, Vista, Windows 7, Windows 3.x/95/98/ME, DOS, OS/2, Mac OS or Linux.
- A *WibuKey* network server is a computer which has the *WibuKey* software installed and is additionally running a *WibuKey* network server process. All *WibuBoxes* which control network licenses are connected to this computer. A server can be any PC on Windows/95/98/ME/NT/2000/XP/Vista/Windows 7, Novell Netware, Mac OS or Linux.
  The computer where the *WibuKey Server* is running does not need to be the data server of the network. Any workstation can be *WibuKey Server* when a *WibuBox* can be connected and the operating system is supported by a *WibuKey Server*. One exception is Novell, where a NLM can only be started on the Novell server.
- `WibuKey.INI` is the main configuration file for the network access. This file has unique sections for the client and the server. Its network parameters are set via the *WibuKey Control Panel Applet* (see page 272).

1 WibuKey network technology

WibuKey offers a method to protect programs or data in the network requiring only a single W*ibuBox*:

- **WkLAN** is protocol based (currently TCP/IP): Protection requests from clients to the *WibuKey Server* are transferred by this protocol.

Please note that this guide will not deal in detail with **WkNet**, an older method although still supported by *WibuKey*. **WkNet**w is mentioned only where explicitely used in software GUIs or commands.

Implementing *WibuKey* using this older method is not recommended.

1.1 Features of WkLAN

WkLAN provides the following features:

- simple to install on client and server side.
- support of nearly every local *WibuKey/WibuBox* functionality, including remote programming of *WibuBoxes* via TCP/IP etc.
- fast encryption and decryption of data.
- support of many different FIRM CODE/USER CODE based licenses by one server.
- requires TCP/IP support of the network and a proper TCP/IP installation on the server and all client machines.

2 WkLAN – protocol-based protection

With **WkLAN**, the data transfer occurs synchronously between server and clients in the network: The API call is returned not before the answer is received from the server. Therefore the access to the server *WibuBox* looks for the client very similar to the access of a local *WibuBox*. The end user normally doesn't recognize the difference between local use and network use of *WibuKey* hardware.

WkLAN runs on the following operating systems on the client and/or on the server side:

- Windows NT/2000/XP/Vista/Windows 7
- Windows 95/98/Me
- Linux
- Apple Macintosh

For *WibuKey* version 2.50 or higher only the TCP/IP network protocol is supported. The support of the NetBEUI or the AppleTalk protocol is not planned because such protocols are being replaced more and more by TCP/IP.

WkLAN uses the following communication concept: A protected application accesses a *WibuBox* at the client side. This access calls the *WibuKey API* of the local calling driver on the client computer (`WKWIN.DLL`, `WKWIN32.DLL`). This driver sets the received parameters into one or more request datagrams which are sent via the network protocol to the server. At the server side, these datagrams are reconverted to an API call which addresses via the calling and kernel driver the local *WibuBox* on the server. The answer to this call (error code, encrypted data, contents of *WibuBox* etc.) is converted into an answer datagram which is sent back to the client. The driver at the client side stores back the received data into the protected application and terminates the execution.

WkLAN supports several servers with several *WibuBoxes* or the moving of a *WibuKey Server* and its *WibuBox* from one computer to another. Before a client communicates with a *WibuBox* of a specific server, it sends a broadcast message to all potential servers. The first server which has a suitable and free *WibuBox* network entry and has answered is used as the network server.

WkLAN is nearly completely transparent for the local use of *WibuBox* hardware. This makes it possible to start the protection of a program for a single computer. Without major changes it allows the extension of the local protection into a network based protection as soon as desired. It is very easy to search for a local *WibuBox* first, and if no suitable local *WibuBoxes* have been found, to browse the network for *WibuBox* hardware. With **WkLAN** nearly every function that accesses the *WibuBox* locally can access it over the network. All WIBU-API functions can be directed to a network server. It is possible to open a *WibuBox* over the network, to read its contents, to encrypt data or to reprogram entries. Even reading or resetting of diagnosis messages of the *WibuKey Server* is supported from any client. If a FIRM SECURITY BOX (FSB) is attached to a *WibuKey Server*, it is possible for a software developer to use his secure and unique FIRM CODE to encrypt programs or data or to program *WibuBox* hardware from any client in the network.

2.1 WkLAN configuration

The implementation of **WkLAN** includes the configuration of a *WibuKey Server*, the configuration of the clients and the protection of the desired software.

2.1.1 WkLAN server configuration

At the server side, the configuration for **WkLAN** is very easy. The *WibuKey Server* process is started. It searches for all locally attached *WibuBoxes* and makes them accessible over the network as *WibuKey* network clusters (see page 252).

For the server process to run, the required *WibuKey* drivers and the *WibuKey Control Panel Applet* (see page 272) must be installed.

It is a good idea to test the attached *WibuBoxes* locally with the *WibuKey Control Panel Applet* before you start the *WibuKey Server*.

The *WibuKey* server process is available in a number of variants:

- `WKSVW32.EXE` for Windows 95/98/Me/NT/2000/XP/Vista/Windows 7. On Windows NT/2000/XP/Vista/Windows 7 this server can be started as system service or as application. On Windows 95/98/Me the driver is started as application.
- `WKSVW16.EXE` for Windows for Workgroups 3.11. This is a 16-bit variant of `WKSVW32.EXE`. It supports **`WkLAN`** but the limited multitasking support of Windows 3.11 reduces the system throughput.
- `WKSVMAC` for Mac OS 8/9. This is the variant of `WKSVW32.EXE` as a Macintosh application.
- `WkSvMacX` for Mac OS X
- `WKSVNLM` for Novell. The current version 2.50 does not support TCP/IP or IPX/SPX. This is planned for a future version. For Novell servers, WkNet should be used.
- `WkSvLin` for Linux. The current version supports the TCP/IP protocol. The program can be started with user rights (not recommended) or Linux-like as a daemon (using runlevel-scripts).

The **`WkLAN`** part of the server can be used only when TCP/IP is installed on the server system. On Windows 95/98/Me/NT/2000/XP/Vista/Windows 7 or Apple Macintosh this is done by simply clicking an option in the network installation setting. On Windows for Workgroups 3.11, the Windows TCP/IP driver extension must be installed first.

Requirements for a proper test execution of a server:

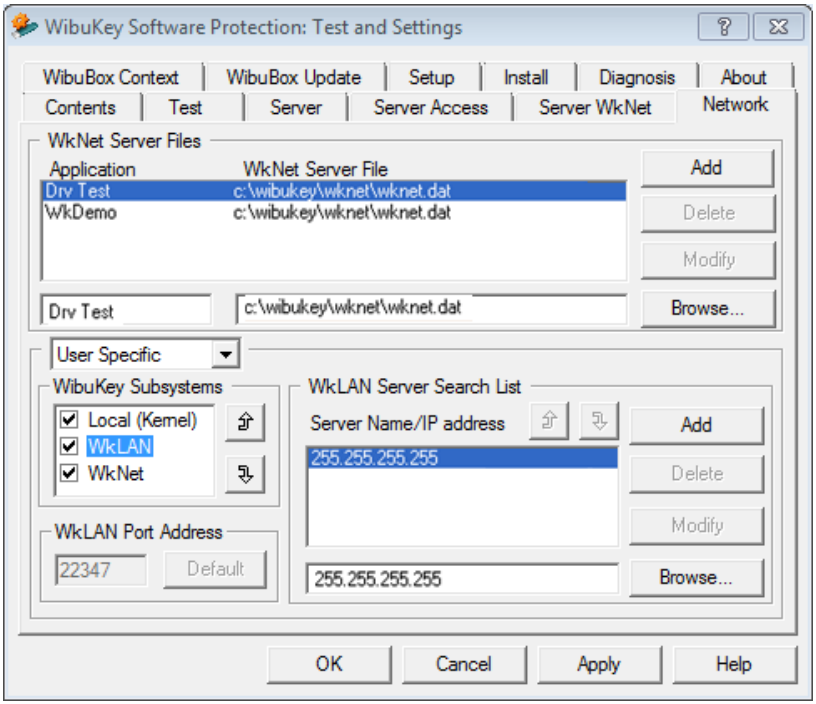


Figure 24: *WibuKey Control Panel Applet* - WkLan Port Address

- A connected *WibuBox* with one or more proper *WibuKey* Network Entries (page 39).
- The corresponding parameter settings are set in the *WibuKey Control Panel Applet* on the Server page (page 272). It can be specified whether **`WkLAN`** clients may reprogram *WibuBoxes* that are attached to the server or not. It is possible to control the access to a FIRM SECURITY BOX (FSB) that is connected to the server.

A timeout for a client application is stored in the `WibuKey.INI` file and may be handled manually by editing this file. It is stored under the topic **`[WkLAN Server].`** The variable name is Timeout. The value of the variable is time in minutes. If there is no entry in the `WibuKey.INI` or the value is 1440.

```
[WkLAN Server]
Timeout=30
```

The *WibuKey Server* version 2.50 lists all found network entries as clusters (page 252) in the server status text.

2.1.2 Adapting WkLAN to custom TCP/IP ports

By default a **WkLAN** client uses the TCP/IP port number 22347 to access a *WibuKey Server* via network. If this port number conflicts with port addresses of other TCP/IP applications, the port number may be changed on the *Network* page of the *WibuKey Control Panel Applet*.

The port number in the range of 1 up to 65,535 may be set in the edit box. The *Standard* button resets the value to the default value 22347.

Changing the **WkLAN** TCP/IP port number must be done by the *WibuKey Control Panel Applet* at server side and on all clients where a **WkLAN** server access is desired. This enables the communication between application and server.

2.1.3 Binding to specific IP addresses

If the computer on which the *WibuKey Server* runs has several network cards it might be desirable. Per default the server binds to the first found IP address. If it should bind to a specific one, this can be specified in the "BindAddress" Value in the registry (Windows) or *.INI file (Mac OS X, Linux), e.g. "BindAddress = 192.168.0.10". On Windows this value cannot be set in the control panel!

2.1.4 Access permissions for the clients

This feature is available in version 4.10 or higher on Windows. On Mac OS X and Linux it will be available in the next version.

There exist 5 permission levels:

Permission Level	Description
1	read content
2	read details (not implemented yet)
4	access license
8	cancel user
16	cancel all users (not implemented yet, treated as 'cancel user')

There is a new value in the registry called 'RemoteAccess'. The value 'Default' describes the permission for all clients which are not listed. Per default it is set to 31 which means that all clients have all permissions. You can add additional values to specify permissions for specific clients. The name must be the IP

address (the computer name only works for ‘access license’) and the value is the permission, e.g.

```
"192.168.0.20" = 7
```

This means that the client with the IP address `192.168.0.20` can only read the server's contents and access licenses, but cannot cancel users. If you set "Default" = 0 only the specified clients can perform any action.

2.1.5 WKLAN client configuration

At best, it is not needed to do any special configurations on the client side to access **WkLAN**. A client computer browses the local network for a suitable **WkLAN** server. The first server with the right entry is used.

All clients in a network need the following tasks:

- The *WibuKey* software must be installed (page 62).
- The TCP/IP network protocol must be installed or activated, exactly like on the server side.

The following items are optional for a simple local area network (LAN). They are required when the *WibuKey Server* is located in another sub-network of a complex, routed network:

- A **WkLAN** Server has to be found via **WkLAN** within a network system. Station names and IP addresses may be entered in the **WkLAN** Server Search List of the Network page of the *WibuKey Control Panel Applet* to search for a *WibuKey Server* via **WkLAN**. If a computer name is used, a name service like DNS (Domain Name Service) or WINS (Windows Name Service) has to be installed. These name services handle the relation between station name and IP address.
- Every server specification can contain broadcast or multicast information. The specification only has to be supported by the underlying TCP/IP implementation.
- If no server name or address is set in the **WkLAN** server search list table, the default setting `255.255.255.255` (LAN broadcast) is used. This broadcast addresses all computers in the local network but no requests are transferred by a router outside of this local network.
  The values in the edit box of **WkLAN** server search list are transferred to the list box by the 'Add' button. By the 'Delete' button, they can be removed and the 'Modify' button is for the modification of an existing entry. The network can be searched for **WkLAN** accessible *WibuKey Servers* by the 'Browse' button: All available **WkLAN** servers will be displayed in a list. The sequence of server addresses can be changed with the 'Up' and

'Down' buttons. The *WibuKey* drivers search within the list from top to bottom and use the first **WkLAN** server containing the desired entries.

In the figure on the left a **WkLAN** client would search for a **WkLAN** server on the computer with IP address `192.168.12.13`. Then it would search by broadcast in the sub-network `192.168.15.255`; and finally in the whole local network system by a local broadcast (`255.255.255.255`).

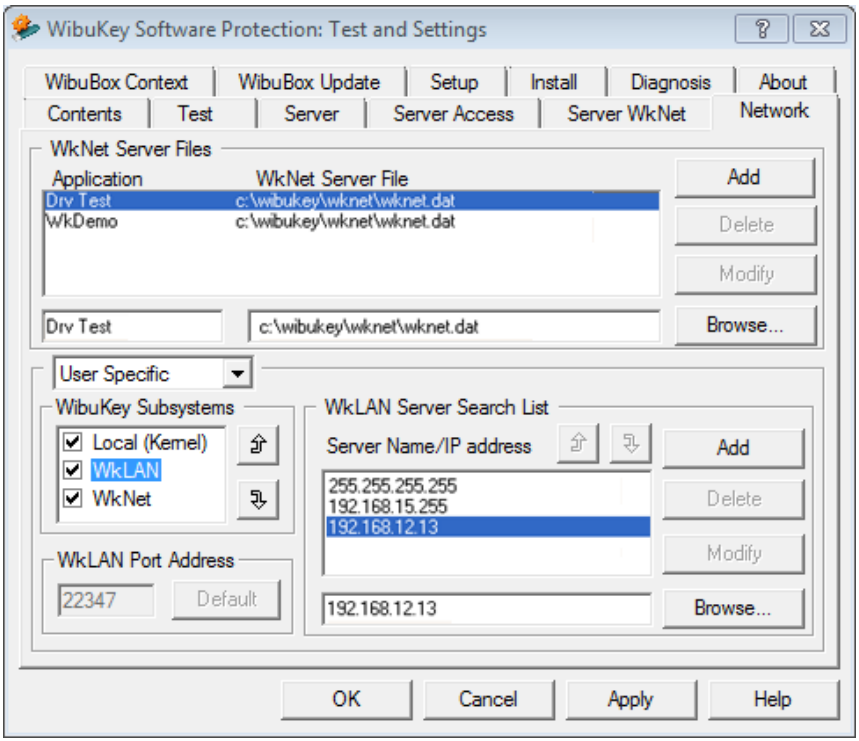


Figure 25: Control Panel Applet: Network page

2.1.6 Protecting software for WkLAN

An application for a **WkLAN** network can be protected explicitly or automatically. For explicit encryption see the Programmer Guide.

For automatic encryption use the corresponding *AxProtector* pages for license handling and runtime settings,for example, see the pages 'License handling' in the figure below. If the protected program is used in combination with a HLM file (page 260) it is advisable to deactivate the local subsystem.

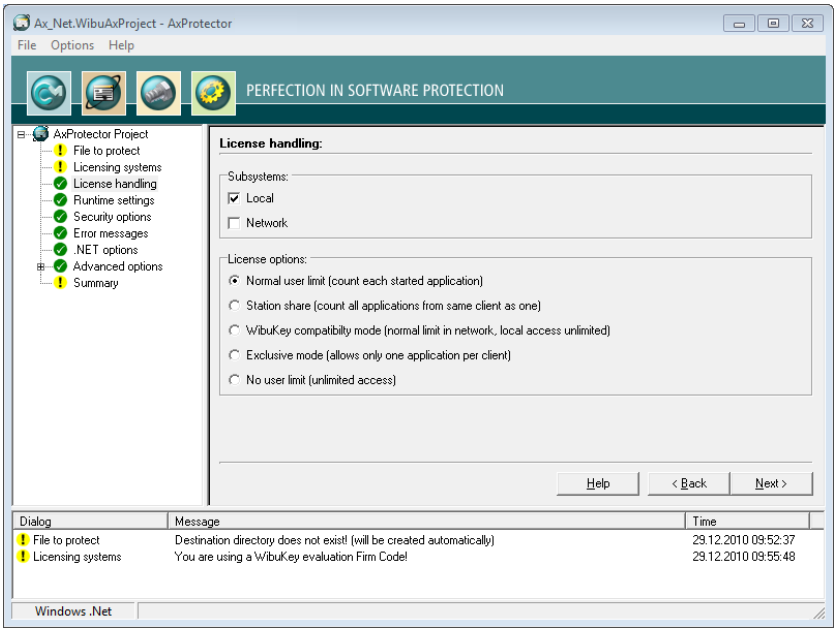


Figure 26: AxProtector settings – license handling example

DOS programs may not be protected automatically for **WkLAN**. Explicitly protected DOS programs may be used only on Windows NT/2000 by using `WKDOS.EXE`.

2.2 First evaluation test

After starting the *WibuKey Server* for **WkLAN** and after protecting your software for **WkLAN**, you can test **WkLAN** in your network. Use the following steps:

❶ Start **WKCRYPT**

❷ Execute the *WibuKey Control Panel Applet* on the client side and go to the *Contents* page. For a **WkLAN** test, no local *WibuBoxes* should be connected and displayed. Click the right mouse button on the network item in the *WibuKey* tree. It executes the *Scanning WibuBoxes* command.

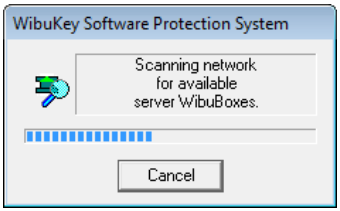


Figure 27: Scanning network

❸ A dialog box appears with a progress bar and a *WibuBox* network searching icon.
After terminating the scanning, the name of the **`WkLAN`** server computer should be listed in the *WibuKey* tree as a sub-item of the Network item. Select a *WibuBox* at one of the displayed server ports. The contents of this box are displayed.
In the screenshot beside the *WibuBox* at the COM2 interface of server `wibu1a.wibu.de` is selected. At the right side the contents appear in the symbol.

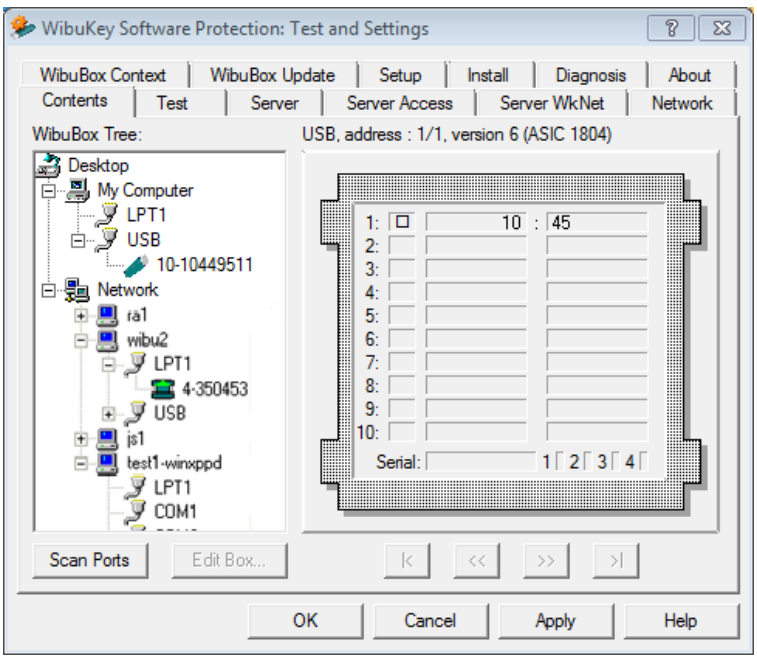


Figure 28: Contents of a *WibuBox* at the server

❹ You can now switch to the *Test* page of the *WibuKey Control Panel Applet* and execute some tests via the network. The test should be successful.

❺ Start an application which is protected for **`WkLAN`**. You can use for example the `WkClock` as sample application. After starting the application it may be looking for a server for about 10 seconds.

❻ If this search fails, check if the *WibuBox* at the server is connected properly, detected by the server and detected by the local *WibuKey Control Panel Applet*.

❼ If a suitable server is found, `WkClock` is protected by **`WkLAN`** which may be checked via the *About* command of this application.

❽ After starting `WkClock` successfully you can remove the *WibuBox* with the Network Entry from the network server. `WkClock` should display an error message box within 10 seconds that the *WibuBox* is no longer available.

3 How does the WibuKey Server work?

This chapter describes some details of the *WibuKey Server*. This information is independent of **`WkNet`** or **`WkLAN`** protection. It is useful for both technologies.

3.1 Starting the *WibuKey Server* on Windows

The `WkSvW32` *WibuKey* server application version 2.50 or later can be executed as system service on Windows NT, 2000/XP/Vista/Windows 7. A system service runs independently of a logged-in user as an application without a visible user interface. It is automatically started after the Windows system is started up, independent of a logged in user. When a user starts a service and logs off, the service remains active until the system is shut down.

Installation and starting of `WKSVW32.EXE` is very simple: Started as application on Window NT/2000/XP/Vista/Windows 7, an icon appears in the system tray. Open the context menu by right-clicking on the icon and select "start as

service". The icon will disappear for a moment (the application is closed) and reappear as soon as the service is started.

If the icon does not reappear the service could not be started for some reason. In this case you can try to start it in the service manager (Start | Settings | Control Panel | Administrative Tools | Services).

When `WkSvW32` *WibuKey Server* is started as an application program, the dialogs for installing, starting, stopping and uninstalling the service are available in the server control menu.

The starting and stopping of the server as service can be controlled by the *services* applet in the system control panel. Furthermore you can start the installed service with

```
net start wksvw32
```

and stop it with

```
net stop wksvw32
```

Please note that the start operation fails during the server is executed as a user-program.

If the *WibuKey* server process is started by the link in "Start | All Programs | WibuKey", it is automatically placed in the system tray. The default start option is ‘Start as Application’.

Installing, uninstalling, starting and stopping of the service will be supported with a future version of `WKU32.EXE` or `WIBUKE32.CPL`.

On Windows 95/98/Me the `WkSvW32` *WibuKey Server* can be used as a user application – services are not supported on these Windows variants.

3.2 Starting the *WibuKey Server* on Linux

WkSvLin can be started in two different ways, but only once a time.

Start with User rights

In a terminal window the **`WkSvLin`** can be started.

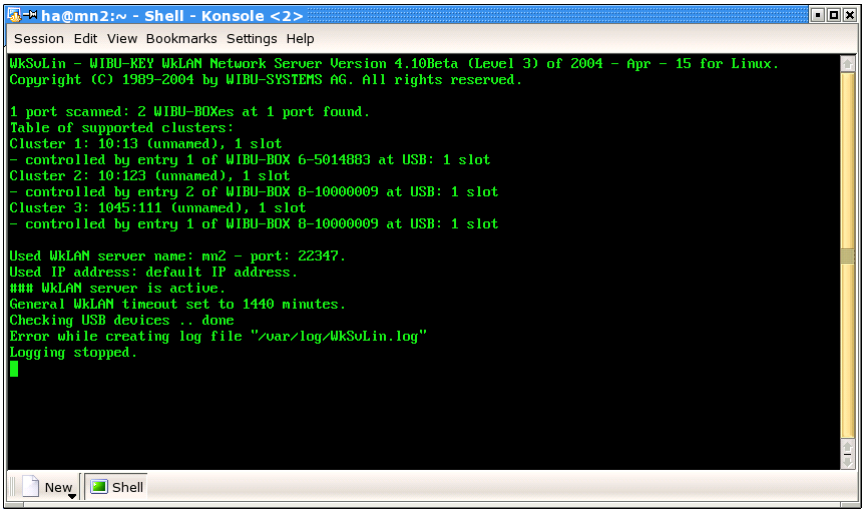


Figure 29: Start of WkSvLin

The output of the server is listed in the terminal window. The server can be terminated by entering “q”, or for getting information about the process by entering “i”.

Start as a daemon

The server process can be started and stopped via the runlevel scripts, too. To do so a corresponding link must be set at the runlevel entries. The daemon can be started manually by:

```
/etc/init.d/wksvlin start
```

Information about a running daemon can be retrieved by:

```
/etc/init.d/wksvlin status
```

```
/etc/init.d/wksvlin stop
```

The configuration file `WkSvLin.ini` can be found in the directory `/etc/wibukey`. This configuration file only should be modified when the server process is not running. If the server process is running as a daemon the logging to a log file always is activated.

3.3 Starting *WibuKey Server* on Mac OS X

The *WibuKey Server* for Mac OS X (WkSvMacX) can be started in 2 ways:

- manually using the "WkSvMacX Start.command" in the `/Applications/WibuKey` folder

- automatically during System Startup (daemon)

The first variant opens a new terminal window and accepts the same input as the Linux version.

The daemon variant is new with version 4.10 and installed by default, but must be activated!

To activate the daemon, open the **`WkConfig`** application and check the *Start Daemon at boot* on the Server page box. The next time the computer is booted, the *WibuKey Server* will be automatically started.

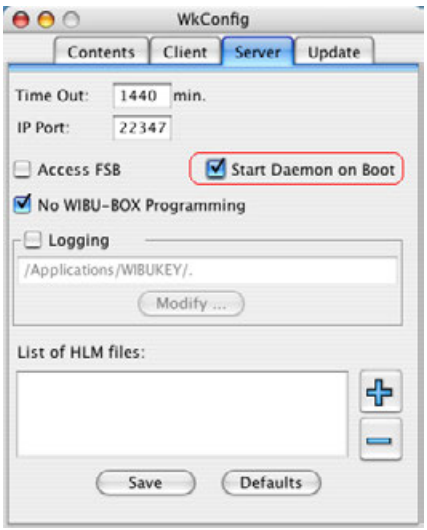


Figure 30: Start server process as a daemon

To start or stop the activated *WibuKey Server* you can also execute the following commands in a terminal window:

```
sudo SystemStarter start "WibuKey Server"
```

to start the activated daemon, and

```
sudo SystemStarter stop "WibuKey Server"
```

to terminate the activated daemon.

If the *WibuKey Server* is NOT configured to automatically start, invoking the SystemStarter sripts as shown above will result in an error message.

3.4 *WibuBox* network entries

The *WibuKey Server* analyses automatically all available and accessible *WibuBoxes* at its local computer and builds a list of one or more *WibuBox* Network Entries. Such entries may be used for **WkLAN** access or to control **WkNet** files.

In the simplest case, a single FIRM CODE/USER CODE entry or FIRM CODE/MASTER CODE Entry is accepted as *WibuBox* network entry with a network USER QUANTITY of 1. A higher network USER QUANTITY can be determined by a second entry with the same FIRM CODE (see page 39).

Things become complex when the scanned *WibuBoxes* contains several entries with the same FIRM CODE and USER CODE or MASTER CODE and several different ADDED ENTRIES, for example LIMIT COUNTERS and/or EXPIRATION DATES.

To handle the trivial and the complex *WibuBox* contents, following rules are established:

- All *WibuBoxes* which are accessible at a server computer are scanned beginning with LPT1 and ending with the highest supported *WibuKey* Port, for example USB.
- A *WibuBox* network entry which can be accessed via **WkNet** or **WkLAN** is always determined within one *WibuBox*.
- When a single *WibuBox* is analyzed, the entries are scanned from the first entry to the last entry.
- Every FIRM CODE/USER CODE or FIRM CODE/MASTER CODE is a candidate for a network base entry with three exceptions, explained in the next three rules. A network base entry is the base for a *WibuBox* network entry.

The following *WibuBox* entries are not candidates for a network base entry:

- When the value of the USER CODE Entry or MASTER CODE entry is in the range `15,728,640` (`0xF00000`) up to `16,777,215` (`0xFFFFFF`), this entry is reserved for Huge License Management (see page 44) and the entry with the next index is ignored as a candidate for a network base entry.
- A new USER CODE entry with an Added Entry (LIMIT COUNTERS, EXPIRATION DATE or ADDED DATA Entry) is found. Another USER CODE entry with the same USER CODE and a smaller index already exists in the *WibuBox*. Then the new entry is ignored as network base entry. This rule permits the usage of several USER CODE Entries with different Added Entries for example LIMIT COUNTER and EXPIRATION DATE and having a sum of network users by the multiple entries.

- A similar rule like the last avoids the creation of a network base entry from a USER CODE entry or a MASTER CODE entry, both with an Added Entry, when a compatible entry with smaller index already exists in the *WibuBox*. "Compatible" means that the USER CODE is matched by the MASTER CODE or both MASTER CODES are identical.

After a network base entry is determined, the **network USER QUANTITY** is calculated. Both values result in the *WibuBox* Network entry.

- For every network base entry, a second USER CODE (no MASTER CODE) entry with a higher index and the same FIRM CODE is searched. The USER CODE values can have a difference between 1 and 250 to the USER CODE or MASTER CODE value of the network base entry.
- If several of such entries exist, the entry with the lowest difference is used; the difference in the USER CODE is stored as network USER QUANTITY of the *WibuBox* network entry.
- If no such entry exists, the network USER QUANTITY of the *WibuBox* network entry is set to 1.

If several *WibuBox* network entries exist with identical USER CODE or MASTER CODE, they are combined into a single ***WibuBox* network cluster** and their single network count entries are added to one big cluster sum. Details about clusters are explained on page 252. Within a single *WibuBox*, to know the adding scheme, it is important to understand the following examples.

Network Cluster – Adding scheme for one *WibuBox*

Network Cluster – Adding scheme for one *WibuBox*

1: *WibuBox* has entries

1) `10:100` 1 user, smallest non-0-difference is `1` to `10:101`
2) `10:101` 1 user, smallest non-0-difference is `1` to `10:102`
3) `10:102` 1 user, no difference to any entry behind this entry

2: *WibuBox* has entries

1) `10:100` 1 user, no network entry for USER QUANTITY
2) `10:100` 1 user, no network entry for USER QUANTITY
3) `10:100` 1 user, no difference to any entry behind this entry

All together, there are 3 slots available for `10:100`

3: *WibuBox* has entries

1) `10:100` 100 users, smallest non-0-difference is `100` to `10:200`
2) `10:200` 1 user, no difference between `1` and `250`
3) `10:500` 1 user, no difference to any entry behind this entry

4: *WibuBox* has entries

1) `10:100` 200 users, smallest non-`0`-difference is `200` to `10:300`
2) `M10:255` 45 users, smallest non-`0`-difference is `255` to `10:300`
3) `10:300` 1 user, no difference to any entry behind this entry

A MASTER CODE is never used to calculate the USER QUANTITY.

Network Cluster – Adding scheme for one *WibuBox*

5: *WibuBox* has entries

1) `10:100` 1 user, no network entry for USER QUANTITY

> ... (first entry: EXPIRATION DATE does not influence calculation) 2) `10:100` 0 user, entry already in this *WibuBox* (1) and has added entry.... # ... (Added Data Entry)

3) `10:100` 1 user, no difference to any entry behind this entry

All together, there are 2 slots available for `10:100`

6: *WibuBox* has entries

1) `M10:255` 1 user, no network entry for USER QUANTITY

2) `10:100` 0 user, entry compatible with MASTER ENTRY in this *WibuBox* (1) and has added entry .... # ... (Added Data Entry)

All together, there is 1 slot available for `M10:255`

4 WibuBox network cluster

4.1 The basics

When a *WibuBox* has only one *WibuBox* network entry or every network entry has a separate FIRM CODE and USER CODE, the understanding of slot allocation is very simple. There are situations, where the allocations becomes more and more complex, like the next two examples demonstrate:

Network Cluster – Complex Slot Allocation

A *WibuBox* has following entries

1) `10:100` 5 users, smallest non-0-difference is `5` to `105`
2) `10:105` 1 user, no difference to any entry > `105`
3) `10:100` 5 user, smallest non-0-difference is `5` to `105`
4) `10:105` 1 user, no difference to any entry behind this entry

When a client accesses the *WibuKey Server* with FIRM CODE/ USER CODE `10:100`, it finds the *WibuBox* network entry `1/2` and allocates slot `1`. The next 4 clients accesses allocate slot 2 to 5. Now another client access `10:100` is coming. The *WibuBox* network entry `1/2` is full but the network entry 3/4 has 5 more slots. If these slots are allocated, the slot 1 is allocated again. The problem is: There are two different users in the network which have the same network user number. This is very confusing for the users themselves and for the network administrator. A solution is to combine the two *WibuBox* network entries to a single entry with 10 slots.

The *WibuKey Server* handles all *WibuBox* network entries as ***WibuBox* network clusters**:

- *WibuBox* network entries with identical FIRM CODE and USER CODES or FIRM CODE and MASTER CODES are combined to a single cluster. Identical network entries can be in a single *WibuBox* or in several *WibuBoxes* at the same or different ports.
- A *WibuBox* network entry with a MASTER CODE which is compatible to the USER CODE or MASTER CODE of other *WibuBox* network entries is bundled to the cluster as overflow cluster.

A future version of the *WibuKey Server* will support the combination of *WibuBox* network entries at different servers to solve large enterprise license management purposes.

When a client request is received, the *WibuKey Server* looks for a free slot in following search order:

- At first, it is looking for a free slot in the USER CODE Cluster of the required FIRM CODE and USER CODE. This cluster is called **BASE CLUSTER**.
- If no such nase cluster exists, it searches in the first MASTER CODE cluster which MASTER CODE is compatible to the required USER CODE.

If the base cluster is full, it looks in the list of overflow clusters which MASTER CODES are compatible. A slot index within the base cluster is allocated and a

different slot in the overflow cluster. The User is handled in two slots in two different clusters but is allocated only once.

4.2 Network clusters

The table of created clusters is displayed by the *WibuKey Server* version 2.50 or higher in the *WibuKey Server* Status Text.

- USER CODE Clusters are ordered by their FIRM CODE and USER Code. This increases the speed to find a specific entry by binary search.
- Moreover the single *WibuBox* network entries of a cluster are reordered: They are sorted by the number of users, the *WibuBox* network entry with the most users is coming first within a cluster. This guarantee that the slots of the biggest *WibuBox* network entries are allocated at first before another *WibuBox* network entry is allocated which increases the allocation speed.
- MASTER CODE clusters are ordered by their compatibility vice-versa. When a MASTER CODE is a full or partial part of another MASTER CODE it is placed before the other MASTER CODE. By this rule, the *WibuKey Server* tries to allocate a slot in a specific MASTER CODE cluster before it allocates a slot in a more general MASTER CODE cluster. The slots of the more general MASTER CODE cluster are “more expensive” because they are configured more generally than the other cluster.
- Within the sorted cluster list, each cluster specifies the number of slots or users to allocate. When a cluster has no overflow cluster, the number of slots is fixed. But when one or more overflow clusters exist, they are specified with their index and the slot size is increased to a maximum value.

Cluster in one *WibuBox* at the server

1: *WibuBox* entries:
1) `10:100` 5 users, smallest non-0-difference is 5 to `10:105`
2) `10:105` 1 user, no difference to any entry > `105`
3) `10:100` 1 user, no difference to any entry behind this entry
For `10:100`, 6 users are supported in a cluster; for `10:105` 1 user.

2: *WibuBox* entries:
1) `10:100` 5 users, smallest non-0-difference is 5 to `10:105`
2) `10:105` 1 user, no valid difference to any entry > `105`
3) `M10:511` 5 users as MASTER ENTRY to `10:516`
4) `10:516` 1 user, no difference to any entry behind this entry
For cluster `10:100`, at least 5, at most 10 users are supported

Cluster in one *WibuBox* at the server

For cluster `10:105`, at least 1, at most 6 users are supported.
For cluster `M10:511`, all USER CODES in the range 0 up to 511 are accepted, in the sum up to 5 users are permitted.

3: *WibuBox* entries:

1) `10:13` 2 users, smallest non-0-difference is 2 to `10:15`
2) `10:15` 5 users, smallest non-0-difference is 5 to `10:20`
3) `M10:15` 5 users as MASTER ENTRY to 10:20
4) `10:20` 245 users, smallest non-0-difference is 245 to 10:265
5) `M10:255` 10 users as MASTER ENTRY to `10:265`
6) `10:265` 1 user, no difference to any entry behind this entry.

Master cluster `M10:15` accepts up to 15 users because master cluster `M10:255` is compatible.
User cluster `10:13` accepts up to 17 users because it has both master clusters as overflow clusters.

4: *WibuBox* entries:

1) `10:13` 7 users, smallest non-0-difference is 2 to `10:15`
3) `M10:15` 5 users as MASTER ENTRY to `10:20`
4) `M10:17` 3 users as MASTER ENTRY to `10:20`
5) `10:20` 1 user, no difference to any entry behind this entry

The cluster 10:13 supports up to 12 users because of MASTER CODE 15.
The cluster `10:15` supports up to 8 slots because of MASTER CODE 17, e.g. for USER CODE 1 which matches 15 and 17.

The cluster `10:13` supports up to 12 slots because MASTER CODE 15 is compatible.

4.3 Cluster management

Every cluster has a unique FIRM CODE/USER CODE or FIRM CODE/MASTER CODE identification. To manage clusters more simply by the end user of protected software, clusters have names which are used in the **WkSvMon** application or in the *WibuKey Server* Status Text.

The name of a cluster is stored in the `WibuKey.INI file` (in future in the registry of Windows 95/98/Me/NT/2000/XP/Vista/Windows 7) and may be handled manually by editing this file. They are stored under the topic **[Net Cluster]** and the variable name is the cluster entry, leaded by an "M" for master clusters. The variable contents is the name of the cluster, for example:

```
[Net Cluster]
10:13="WkClock"
10:22=”Sample Application”
99:722="WkCrypt"
10:366=none
```

If "none" is specified without surrounded quotation, the cluster is locked against any access: It is simply ignored by the *WibuKey Server*.

Using "none" is a good idea to remove unused clusters from a cluster list which are never used by an application, for example the FIRM CODE/USER CODE which is used as network entry to calculate the USER QUANTITY of another entry but which is also interpreted as cluster.

The *WibuKey Server* reads the cluster name list from `WibuKey.INI` only during its start. To reflect any changes in the list to the *WibuKey Server*, it must be stopped and started again.

On the other hand, the *WibuKey Server* adds names in the list automatically and uses the stored names when it is started again: When a cluster has no name stored in `WibuKey.INI` and a request is coming from a **WkLAN** client, the application name of this client is interpreted by the *WibuKey Server* as cluster name and stored in to the `WibuKey.INI file`. This automatic name setting is never done if a name is already stored for a specific cluster. Therefore any manual entered names in the `WibuKey.INI` file are never change by the *WibuKey Server*.

5 Handling of network users

When several clients access a *WibuKey Server* they are allocating slots within the cluster of the requested FIRM CODE and USER CODE. Each of the clients receives a unique user number when the slot allocation was successful.

This chapter describes some details about the selection process of a specific user number and how this selection can be influenced by the software developer.

5.1 Sharing local and network access

Principally, a *WibuBox* at the network server may be accessed locally by a protected application without using the *WibuKey Server* application. But the *WibuKey* Kernel driver avoids the simultaneous use of a *WibuBox* network entry by the *WibuKey Server* and a locally executed application at the same time:

- Whenever the *WibuBox* network entry is used by the network it is locked against a local *WibuBox* access except the protected application does not want to allocate a network user or license. This avoids that the local access on the server system results in one additional license.
- When a *WibuBox* network entry is used to control a **`WkNet`** server file, this entry is permanently used by the *WibuKey Server* and locked against local access.
- When a *WibuBox* network entry is not used by a **`WkNet`** server file nor any allocated **`WkLAN`** slot, it may be used locally. In this case, the entry cannot be locked during the local use by any request from a network client to a slot of this *WibuBox* network entry: Every access will fail.

The last case may be a problem when software is used on the *WibuKey Server* and is the first allocating client in the network. The client accesses the *WibuBox* locally and not by the *WibuKey Server*. By this he locks the *WibuBox* Network Entry for all other client requests in the network. To avoid this, the local access to the *WibuBox* network entry should be disabled (page 283). The application is always addressing the active *WibuKey Server* by **`WkLAN`** or **`WkNet`**. The local access is no longer used.

5.2 *WibuKey Server* without user limitation

Sometimes a *WibuBox* at a server should be used without any limitation or licenses in the network. For example a software developer sells an application and permits his customer an unlimited number of users. This may be handled by disabling the network user count. Only **`WkLAN`** supports this possibility of unlimited users. The automatic encryption of executables by **`WKCRYPT`** permits this in the option dialog.

5.2.1 Unlimited access

When the user limitation in the *Network License Management* area is disabled, no slot is allocated. The first *WibuBox* network entry which is assigned to the used cluster is used for encryption of a network client request. This works also if the network entry is locked by this cluster.

5.2.2 User limitation

You can locally start an application as often as you want when using user limitation. The figure below from *Axprotector* shows how to define whether a protected application is to search for existing licenses locally in *WibuBoxes*, in the network, or both. Moreover, you can define the license allocation mode.

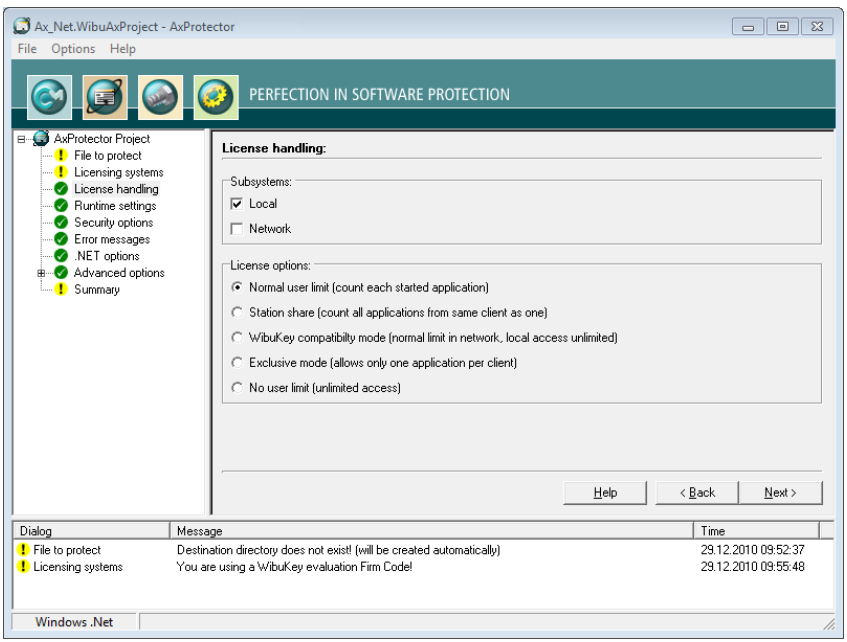


Figure 31: User limitation

If a *WibuKey Server* process is started and if the necessary *WibuBox* entry is used by the process via client accesses, the locally started application only can access the *WibuBox* via the server. In this case, the local use is limited by the number of slots of the used cluster. With the option of the 'Exclusive access mode' an application can be started only once. More simultaneous starts of the application are only possible when various *WibuBox* entries for the combination of FIRM CODE and USER CODE are available. Deploying a multi-user windows, a one-user license cannot be started several times.

With the 'Client specific user identification' a fixed slot number (or client number) can be allocated to every working place. The numbers are stored at the moment in the file `WibuKey.INI`. The option 'Auto cancel mode' can be used together with the 'Exclusive mode' and the 'Client specific user identification' within the network. If the current slot is reserved when starting an application, the server automatically cancels the client and allocates the slot by the new client. This way it is avoided that entries, which are blocked constantly by not properly terminated programs, cannot be used anymore by other programs.

5.3 Setting of a fixed slot by a client

A *WibuKey* protected client application may specify a specific user index value which is used to allocate a specific slot within a cluster. Optionally, if the slot is allocated, it is cancelled automatically and allocated again. If this option is not set, the allocation fails. The Auto-Cancel option is useful to cancel old network licenses which had not freed their slot properly because by a system or application crash this was not done.

The setting of explicit network user indices is currently not supported by the automatic encryption of executables. This is planned for the future.

There are two variants to specify the user index for a specific cluster:

- The user index is stored in the `WibuKey.INI` file (in future the registry of Windows 95/98/Me/NT/2000/XP/Vista/Windows 7) and the end user can modify these values by editing `WibuKey.INI`. This is not implemented for DOS programs.
- The numbers are stored application dependently within `WibuKey.INI` under the **`[Net User]`** in following variables:
- `ApplicationName = UserIndex`
- The user index of the first slot is 1; the application name must be identical with the name which is used to access a **`WkNet`** file.

The setting of the user number for a specific application may be controlled in future by the *WibuKey Control Panel Applet* on the network page.

- The user index is stored in the application itself and may be selected by the end user by an application specific dialog or other setting. This handling is available for DOS programs.

6 Huge license management

As explained shortly on page 44, **Huge License Management** (HLM) is used to protect many licenses of a software product. Every license is controlled by a different cluster with a different USER QUANTITY. A *WibuBox* may support up to 5 of such clusters. For more clusters several *WibuBoxes* are required or one or more **HLM Control Files**. Such a control file can support several thousands of clusters if required. Each cluster may have up to 250 slots per cluster at most. The limitation is only done by the required memory and the power of the *WibuKey Server*, not principally by limitations of the used *WibuBox*. Each of these HLM clusters has a fixed name as description which is stored into the HLM control file.

HLM is fully compatible to the *WibuKey* network license management. The *WibuKey Server* reads the *WibuBox* network entries into a network cluster table and adds all clusters which are read from the opened and decrypted HLM control files. A network client accesses a HLM cluster in the same way as a *WibuBox* networkentry based cluster.

HLM can be principally used with **`WkNet`**. Because for each HLM cluster a separate **`WkNet`** file is required, this combination is limited. It is possible for example, to add the **`WkNet`** support only to some HLM clusters of a HLM control file.

A single HLM file requires two entries in a *WibuBox*:

- A BASE ENTRY to decrypt the encrypted HLM vontrol file. This must be compatible with the BASE CODE which was specified by the creation of the HLM control file (see page 261). A normal *WibuBox* network entry could be used as HLM decryption entry to increase the number of maximum network users. To avoid this, the network entry for the encryption of a HLM control file must be in the range `15,728,640` (`0xF00000`) up to `16,777,215` (`0xFFFFFF`). This USER CODE range must not be used for normal *WibuBox* network entries and allows therefore a separation between HLM and *WibuBox* network license management.
- The next entry in the *WibuBox* must be a MASTER ENTRY. It must be compatible to several or all HLM clusters of the HLM control file. This entry is used to decrypt the specific HLM clusters of the file individually and to handle encryption and decryption operations coming from a network client. To reduce the number of matching bits in this MASTER ENTRY, the number of usable HLM clusters of a specific HLM control file may be reduced by a specific *WibuBox* and can be increased, for example, by remote programming (see page 226).

Both entries are not candidates for a normal *WibuBox* network entry which avoids the mixing of HLM and *WibuBox* based network license management.

After setting a HLM file at server side and starting the *WibuKey Server*, all available HLM clusters are listed with the *WibuBox* controlled clusters in a single cluster list in the *WibuKey Server* status text. The HLM clusters specify their HLM control file instead of specifying the controlled *WibuBox* entry.

6.1 Creating a HLM control file

The program **WKCRYPT** allows to generate a HLM control file by specifying the **/NET** mode option. Following settings must be done:

- The FIRM CODE and USER CODE of a HLM control file which are used to encrypt the file. They must be found by the *WibuKey Server* in the *WibuBox*. The USER CODE must be in the range `15,728,640` (`0xF00000`) up to `16,777,215` (`0xFFFFFF`).
- Optionally an extended name of the HLM control file.
- For each HLM cluster (license) the FIRM CODE and USER CODE and the number of slots which are controlled by this cluster.
- Optionally each HLM cluster may receive an individual name which is used by the *WibuKey Server*.

The current implementation of HLM requires an identical FIRM CODE for all clusters of a single HLM file. Usually the HLM file encryption and the clusters use the same FIRM CODE. For special purposes both codes may differ.

A HLM control file has typically the extension `*.WBB`: This is the abbreviation of "WIBU-Binary" and is a universal binary data exchange format of Wibu-Systems. It contains a header which explains the meaning and the version of the binary file and contains a lot of CRC check sums which avoid to accept a binary file which is manipulated by adding or modifying single bytes.

The HLM control file generation syntax is:

WKCRYPT /NET `[Option…] HlmFileName[.wbb]`

The following options are available for **WKCRYPT /NET** to create a HLM control file:

Option	Description
/F*FirmCode*	specifies the FIRM CODE for the HLM file or the FIRM CODE for the next generated HLM cluster or license. Currently HLM supports only one FIRM CODE for all licenses. Therefore the FIRM CODE must be specified only once for a complete creation of an HLM control file.

Option	Description
/GH	generates a HLM control file. The last settings in the commandline of FIRM CODE (**/F**) and USER CODE (**/U**) are used to encrypt the complete HLM file; the last setting of the Name (**/N**) are used as extended name of the HLM control file.
/GL	generates a single HLM cluster (license) which bases on the last settings in the commandline: FIRM CODE (**/F**), USER CODE (**/U**), maximum number of users (**/Q)**, and name of the license (**/N**).
/N:*Text*	allows the specification of a name which is set as extended HLM control file name or used as cluster name of the next created HLM license.
/Q*Number*	specifies the maximum number of users for the next created HLM cluster. This value is currently limited to `1,000`.
/U[**M**]*UserCode*	specifies the USER CODE for the encryption of the HLM file or the USER CODE for the next generated HLM cluster. The **/UM** option specifies a MASTER ENTRY which is only permitted for the next generated HLM cluster. The USER CODE for the encryption of the HLM file must be in range `15,728,640` (`0xF00000`) up to `16,777,215` (`0xFFFFFF`).
/V	activates the extended output mode (*verbose*), which issues more information on the monitor during the execution of the process.
/XT*Delta*	sets an EXPIRATION DATE which expires after the specified number of days since today for the next created HLM clusters.
/XT*DateValue*	sets an EXPIRATION DATE for the next created HLM clusters. The date expires at the specified day. The *DateValue* syntax is described in the appendix B10.
/XT	disables the setting of an EXPIRATION DATE for the next created HLM clusters.
/?	prints a brief description of the program, and all options are permitted after the `/NET` option.

Creating a HLM control file

Following WKC file creates a HLM control file with name sample.wbb, extended name “HLM Sample File”, and 5 licenses with FIRM CODE 10 and USER CODE 40 up to 44, each with 10 to 20 users and with different cluster names. License 41 and 42 will be expired in 10 days, License 44 at end of 1999.

```
/net /f10 /u0xF00000 /n:"HLM Sample File" /gh
/u40 /q20 /n:"Addition" /gl
/u41 /q14 /xt10 /n:"Multiply" /gl
/u42 /q12 /n:"Division" /gl
/u43 /q16 /n:"Subtraction" /gl
/u44 /q10 /xt99Dec31 /n:"Square Root" /gl
sample.wbb
```

Creating a HLM control file with overflow cluster

This WKC file example creates the same HLM control file as the example above but additionally an overflow cluster with MASTER CODE 63 and 10 users. The overflow cluster is compatible to all other USER CODE Clusters.

```
/net /f10 /u0xF00000 /n:"HLM Sample File" /gh
/u40  /q20 /n:"Addition"    /gl
/u41  /q14 /n:"Multiply"    /gl
/u42  /q12 /n:"Division"    /gl
/u43  /q16 /n:"Subtraction" /gl
/u44  /q10 /n:"Square Root" /gl
/um63 /q10 /n:"Overflow"    /gl
sample2.wbb
```

6.2 Setting a server-sided HLM control file

A HLM control file must be set manually in the *WibuKey Control Panel*. Go to page 'Server' and select the HLM control file by using the button '…'.

Check the selected HLM control file with the 'Test' button. If no errors can be found, you can add the HLM control file to the list with the 'Add' button. Then click the 'Apply' button. The server only reads the file when it is started. When modifying the file you have to restart the server.

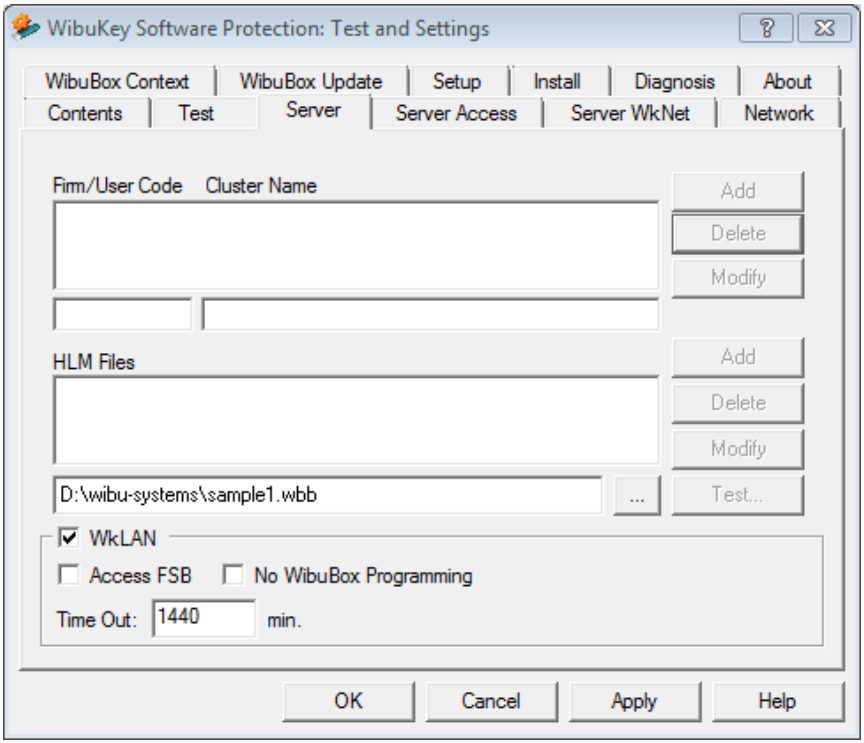


Figure 32: Setting a server-sided HLM control file

6.3 Mixing of HLM and *WibuBox* network clusters

Principally, HLM Clusters and *WibuBox* controlled clusters may be mixed. They are listed as clusters after the *WibuKey Server* is started. Because of security reasons, they cannot be mixed. Moreover a HLM file with a specific FIRM CODE and USER CODE may be specified only once. Following rules control the HLM control file reading by the *WibuKey Server*:

- Every HLM control file must be protected by a specific FIRM CODE/USER CODE entry in a *WibuBox*. If several HLM files are protected with the same codes, only the first read HLM file is used. This avoids that multiple copies of a specific HLM control file are set at one *WibuKey Server* and that the number of USER QUANTITIES of all set HLM control files is added to a big sum value. This would crack the pre-defined USER QUANTITY.
- Several HLM control files which are encrypted by different FIRM CODE and USER CODES are accepted. If they have clusters with the same FIRM CODE

and USER CODE or FIRM CODE and MASTER CODE, these clusters are combined and their slot size is added to a sum value.

- HLM clusters of different HLM control files which have the same FIRM CODE and USER CODE are combined to a single cluster. The number of slots or users is added for this cluster. Within a single HLM control file each cluster is permitted only once. This avoids the cracking of such a file by adding the file contents again at the end of this file.
- HLM MASTER CODE clusters which are compatible to HLM USER CODE clusters are used automatically as overflow clusters for the compatible HLM clusters.
- HLM clusters and *WibuBox* network entry clusters are strictly separated by the *WibuKey Server*. They are never combined to clusters with more slots nor are they overflow clusters vice versa. This avoids the cracking of the secure *WibuBox* controlled clusters by the less secure HLM clusters.

When a cluster with the same FIRM CODE and USER CODE or FIRM CODE and MASTER CODE is specified by one or more *WibuBox* network entries and also by one or more HLM control files, all HLM clusters are ignored: *WibuBox* network entry clusters overwrite HLM control file clusters automatically.

6.4 HLM in the extended memory

This chapter refers to the possibility to store the HLM information in the EXTENDED MEMORY of the *WibuBox/+* variants.

Only a few steps are necessary to realize a license management in the network for your software with one *WibuBox*. Either by the simple model of two entries per license in the *WibuBox* or, essentially simple and more flexible, by the Hugh License Management (HLM). HLM manages the licenses in an encrypted binary file that is updated by the *WibuKey Server* process.

The encrypted HLM file normally is installed with the *WibuKey* components and must be introduced to the *WibuKey* server process by a registry entry. If a *WibuBox* with extended memory is used, the HLM binary file can be copied to this memory, so it can be transported and installed easily. The *WibuKey Server* process reads this information at its start-up and provides all enabled licenses in the network.

6.5 The *WibuBox* entries

With HLM the first entry in the *WibuBox* must have a USER CODE that is higher than `15728640` (`0xf00000`). The second entry controls the enabling of the HLM licenses, it is realized by a MASTER ENTRY whose USER CODE mask is

combined by a binary AND combination with the USER CODE of the protected application..

In following sample we will use the evaluation FIRM CODE `10` and the USER CODE `15728640` for the first entry and a MASTER ENTRY of `10`:`10` for the second:

```
wkcrypt /pau /pi /f10 /u15728640 /pn
```

Because the HLM information only can be programmed in the commandline using the tool **WKCRYPT**, we program the entries for this samples in the commandline, too. So the programming of the *WibuBox*, here we use a *WibuBox/U+*, and the creating of the HLM information is done in one commandline.

Now a part of the unprotected, extended memory must be prepared, that means formatted, for HLM:

```
wkcrypt /pau /pxmf:w30
```

This commandline formats 30 pages of the memory. Now the different licenses must be created and transferred to the memory:

Three licenses are defined:

Main program:
FIRM CODE = `10`
USER CODE= `2`
Number of licenses = `1`

ModulA:
FIRM CODE = `10`
USER CODE= `4`
Number of licenses = `5`

ModulB:
FIRM CODE = `10`
USER CODE= `8`
Number of licenses = `10`

```
wkcrypt /pau
        /net /f10 /u15728640 /n:"MemoryHLM" /gh
        /u2 /q1 /n:"Main program" /gl
        /u4 /q5 /n:"ModulA" /gl
        /u8 /q10 /n:"ModulB" /gl
        /ph
```

The deciding option is the last one. The option **/ph** causes the transfer of the information to the *WibuBox* instead to a file.

Of course the HLM information in the *WibuBox* can be transferred by remote programming, too. Instead of the port, where the *WibuBox* is connected to, you specify the remote context file; the programming is re-directed to a remote

update file. In the following please find the complete commandline for the remote programming:

```
wkcrypt /rc:wibukey.rtc /ru:wibukey.rtu
        /f10 /u15728640 /pn /f10 /um10 /pn
        /pxmf:w30 /net /f10 /u16001000 /n:"MemHLM" /gh
        /u2 /q1 /n:"Main program" /gl
        /u4 /q5 /n:"ModulA" /gl
        /u8 /q10 /n:"ModulB" /gl
        /ph
```

This is the output of the commandline with the direct programming of the *WibuBox*:

```
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: Entry 1 (contents 10:15728640) new programmed.
WIBU-BOX 1: Master Entry 2 (contents 10:10) new programmed.
WIBU-BOX 1: ExtMem area (512 pages) formatted.
WIBU-BOX 1: Start Writing ExtMem User WIBU page 0.
WIBU-BOX 1: Start Writing ExtMem User WIBU page 1 to 6.
HLM data of "MemoryHLM" into WIBU-BOX ExtMem written (3
licenses).
```

If the *WibuKey Server* process is started now, the following output should appear:

```
WkSvW32.exe - WIBU-KEY WkLAN/WkNet Network Server.
Version 5.01Beta (Level 2) of 2005-Aug-25 for Win32.
Copyright (C) 1989-2005 by WIBU-SYSTEMS AG. All rights
reserved.

4 ports scanned: 2 WIBU-BOXes at 2 ports found.
HLM file <8-10001978 block 0> successfully read: 2 of 3
licenses accepted.
Table of supported clusters:
Cluster 1: 10:2 (Main program), 1 slot
- HLM controlled by file <8-10001978 block 0> and entry 2
of WIBU-BOX 8-10001978 at USB
Cluster 2: 10:8 (ModulB), 10 slots
- HLM controlled by file <8-10001978 block 0> and entry 2
of WIBU-BOX 8-10001978 at USB

Used WkLAN server name: COMPUTER - port: 22347.
Used IP address: default IP address.
### WkLAN server is active.
General WkLAN timeout set to 1440 minutes.
```

```
X.XX XX:XX:XX:XXX: WkSvW32.exe is running.
```

As you can see the licenses are read automatically. Because the USER CODE mask of the MASTER ENTRY does not cover ModulA (USER CODE = 4, 5 licenses /10 AND 4 <> 4), this module is not enabled.

The *WibuBox* we are using in the sample now could be connected to any workstation or server in the network and the *WibuKey Server* would provide the licenses without any further installation.

7 WibuKey network tools

7.1 *Control Panel Applet*

For the network use there are two pages available in the *WibuKey Control Panel Applet*. They help defining the right parameters on the client as well as on the server side.

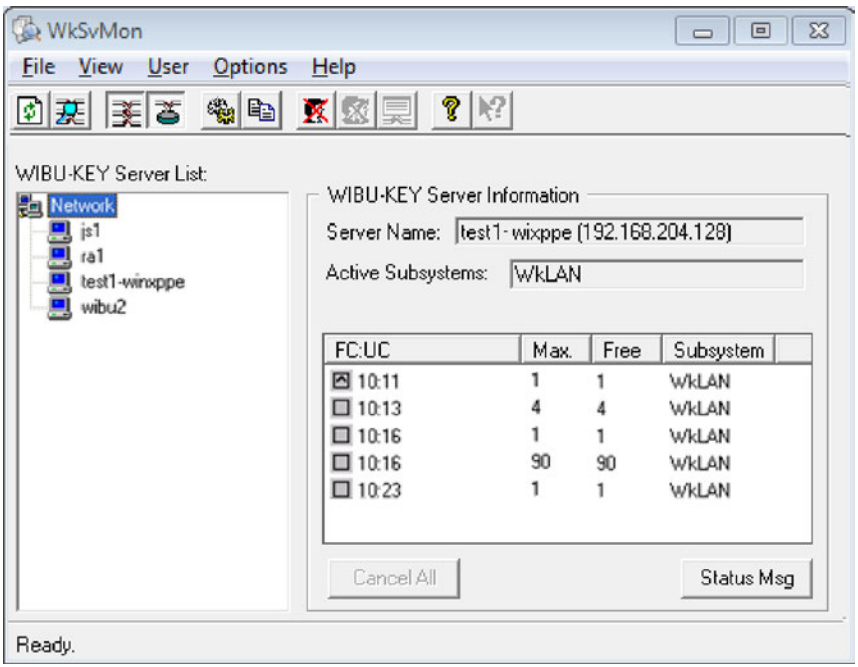


Figure 33: *WibuKey Network Server Monitor*

- For the server settings see the Server page on page 276.
- For the client settings see the Network page on page 281.

7.2 *WibuKey Network Server Monitor*

Wibu-Systems offers a *Network Server Monitor* `WKSVMON.EXE`. This is a 32 bit program for Windows 95/98/Me/NT/2000/XP/Vista/Windows 7. The Server Monitor is the main tool to administer *WibuKey* in a network.

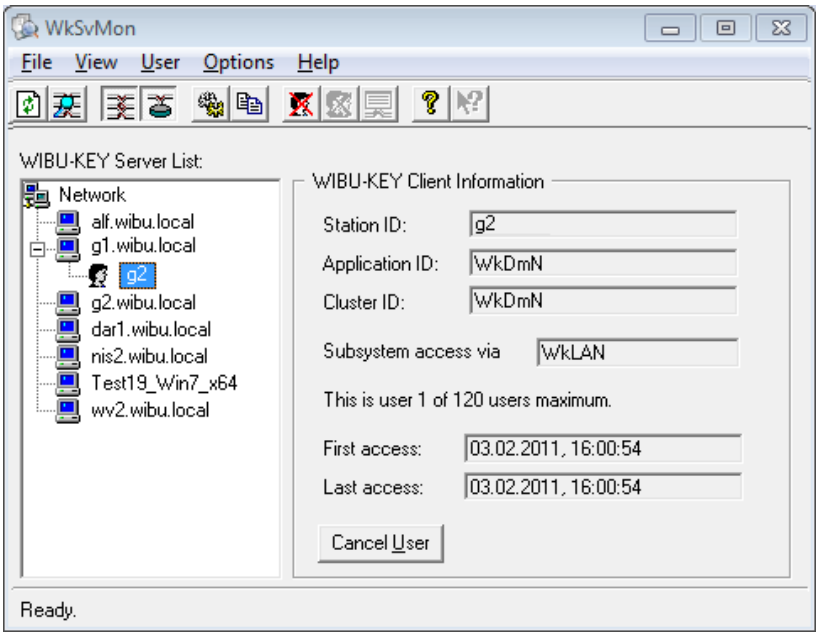


Figure 34: Server Monitor showing user `test1 -winxppe`

In the 'Network Info' area each user can see the user-ID, the login time and the time of the last server access for any chosen server displayed in the *WibuKey Server* List.

In the 'File' menu a `WkNet` file can be opened or closed.

In the 'View' menu it is possible to search the network for servers or to refresh the server list. It is also possible to choose only `WkLAN` servers or only `WkNet` servers. The default setting shows `WkLAN` as well as `WkNet` servers. Via the 'Server Status' option the server status text is displayed.

The 'Option' menu offers the possibility to make general settings for the Server Monitor like the option 'Browse network at startup'.

In the 'User' menu it is possible to disconnect any chosen user from the server. Another option is to cancel all users at once. When a user is selected in the *WibuKey Server List* it is possible to cancel it with the 'Cancel User' button.

The server status can be shown for a chosen server via the 'Status Msg' button. This is important if a server is running as service on Windows NT. The server monitor then offers the only method to view the server status.

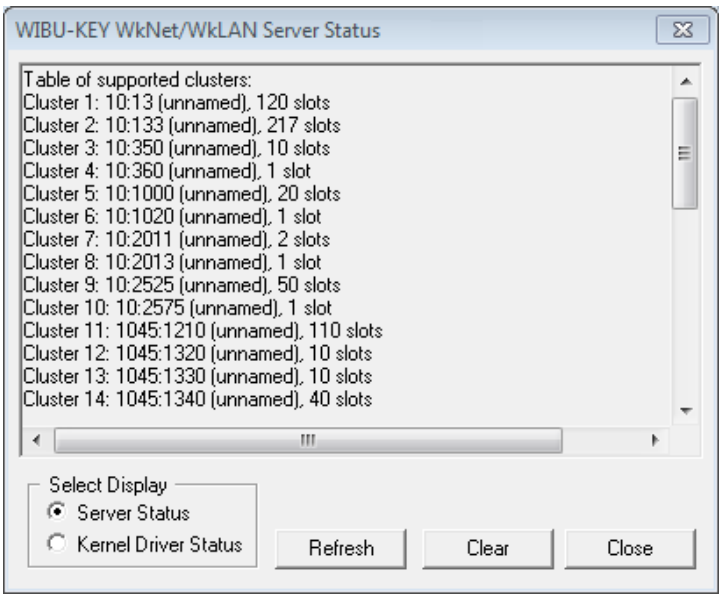


Figure 35: Server Monitor status window

Beside the menu bar there are symbols with the same options to allow an easy and fast access to the main menu options.

7.3 Server process HTTP interface

The Mac OS X and Linux versions of the *WibuKey Server* contain a web interface, which allows local and remote access to some functions similar to the **`WkSvMon`** application for Windows.

To start the *WebAdmin* an Internet Browser (e.g. IE, Mozilla or Safari) must be opened and the IP address or hostname must be specified with the *WibuKey Server* port (default is 22347).

: http://localhost:22347 or http://127.0.0.1:22347 would open a *WebAdmin* on the same computer displaying the start page.

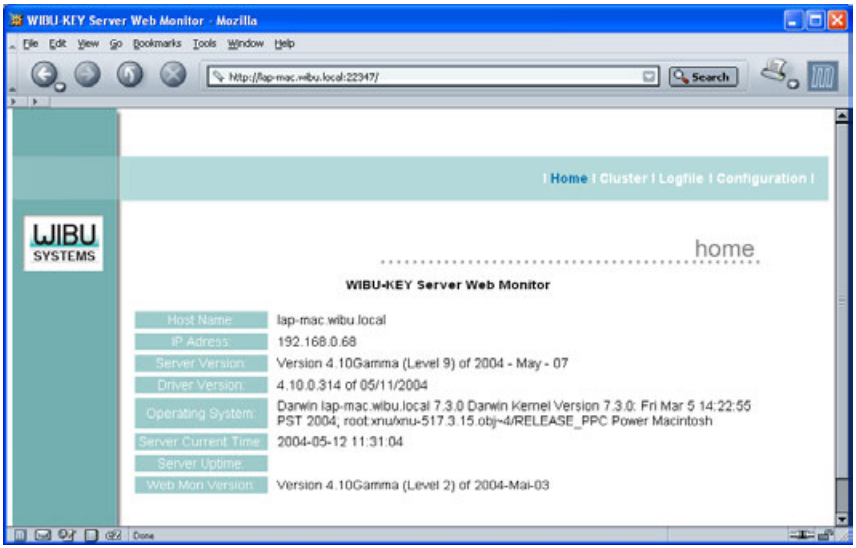


Figure 36: HTTP-Interface

Then navigate to the *Cluster* page to display available and used *Network Clusters*, where it is also possible to cancel connected users.

If the remote cancellation of connected users should be disabled, the following settings can be made in the `WkSMacX.ini` /`WkSvLin.ini file`:

```
[HTTP]
RemoteCommand=0
```

Part VII Controlling and configuring *WibuKey*

Wibu-Systems offers useful tools for software developers and their clients. A lot of the settings necessary for *WibuKey* can be done interactively. This is very comfortable and helpful in finding the right settings for every application.

1 *WibuKey Control Panel Applet*

The main instrument for controlling the access to *WibuBoxes* is the ***WibuKey Control Panel Applet*** in the control panel of Windows. The *Control Panel Applet* is for software developers as well as end users of applications protected with *WibuKey*. One module variant is available:

- `WIBUKE32.CPL` for Windows 95/98/Me/NT/2000/XP/Vista/Windows 7.

There are several pages with the following functions:

- Contents: Displays connected *WibuBoxes* (local and network).
- Test: Tests *WibuBox* functionality.
- Server: Configures settings for the *WibuKey* network server process.
- Network: Sets parameters for clients in a network system.
- WibuBox Context: Stores the contents of *WibuBoxes* in a remote programming context file.
- WibuBox Update: Updates the contents of WibuBoxes with a remote programming update file.
- Setup: Specifies interface and kind of configuration for the driver to search for *WibuBoxes*.
- Install: Installs, deinstalls or updates *WibuKey* drivers.
- Diagnosis: Displays the diagnosis messages of the *WibuKey* drivers.
- About: Shows the version of the *Control Panel Applet* and *WibuKey* driver.

The *WibuKey Control Panel Applet* by default starts with the basic tabs, which provide the most common functionality for the end-user. All additional tabs can be accessed by specifying the "Advanced" option in the Window menu of the applet.

Click on the  symbol and activate the "Advanced" Option item.

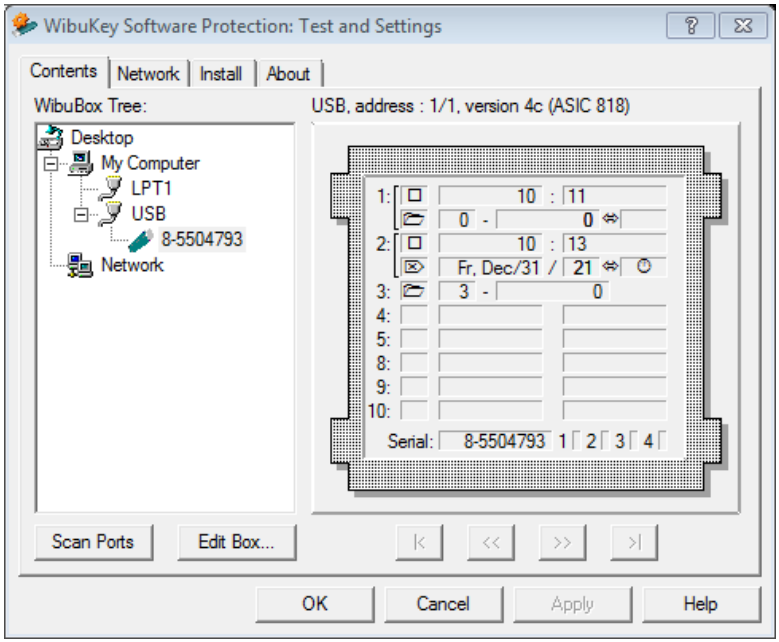


Figure 37: *Control Panel Applet*: Basic tabs

1.1 Contents page

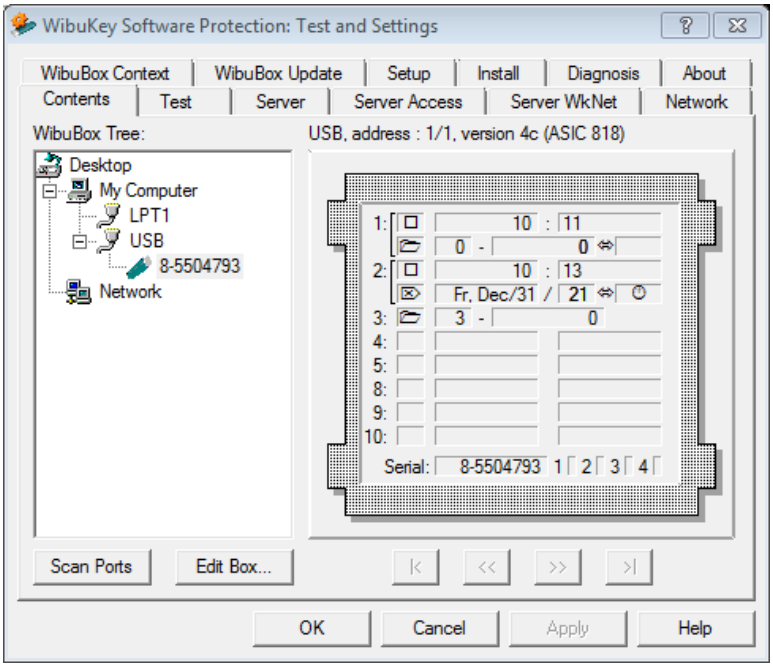


Figure 38: *Control Panel Applet*: Content Page (Advanced Mode)

The Contents page is the most important page of the *Control Panel Applet*. It lists the *WibuBoxes* that are connected either locally or in the network.

Via the port list, a *WibuBox* can be selected and User Data entries can be modified with the *Edit Box* button. Newly connected *WibuBoxes* and any changes to contents of an already connected *WibuBox* are displayed with the *Scan ports* button.

The display is visually equivalent to **`WKLIST`** which is used for the programming of *WibuBoxes*. For detailed explanations of **`WKLIST`** and the different entries see page 209.

The *WibuKey* icon is automatically with the Runtime Kit at the user's site. It is not possible to modify entries other than User Data entries with the *WibuKey Control Panel Applet*.

1.2 Test page

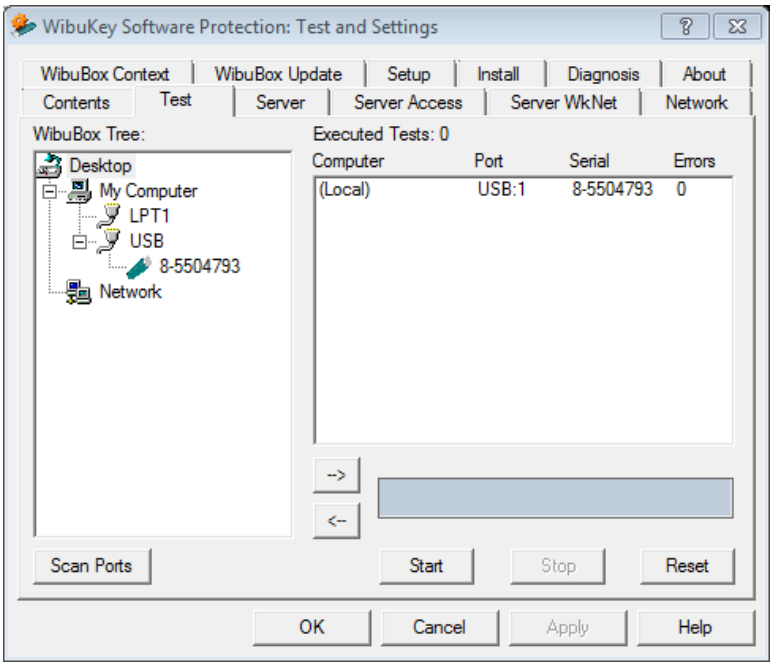


Figure 39: *Control Panel Applet*: Test page

This page checks the function of connected *WibuBoxes*. By default all *WibuBoxes* are displayed in the test list on the right side of the page. With the arrow keys, the marked *WibuBoxes* can be removed from the test window or additional *WibuBoxes* from the *WibuBox* tree can be transferred to the test list.

By pressing the 'Start' button the test procedure starts. It can be stopped with the 'Stop' button. The 'Reset' button resets the error counter to zero. The functionality test of selected *WibuBoxes* is done by transferring randomized data sequences. Errors during the test are listed in the last column of the test list. The test itself runs continuously. It stops automatically when the selection of ports to be tested is changed. The 'Scan Ports' button scans for newly connected *WibuBoxes*. It resets the test counter and the error counter in the Test Results list.

If *WibuBox* hardware is removed or connected, it is necessary to refresh the *WibuBox* tree with the 'Scan Ports' button. The test can then be restarted.

1.3 Server page

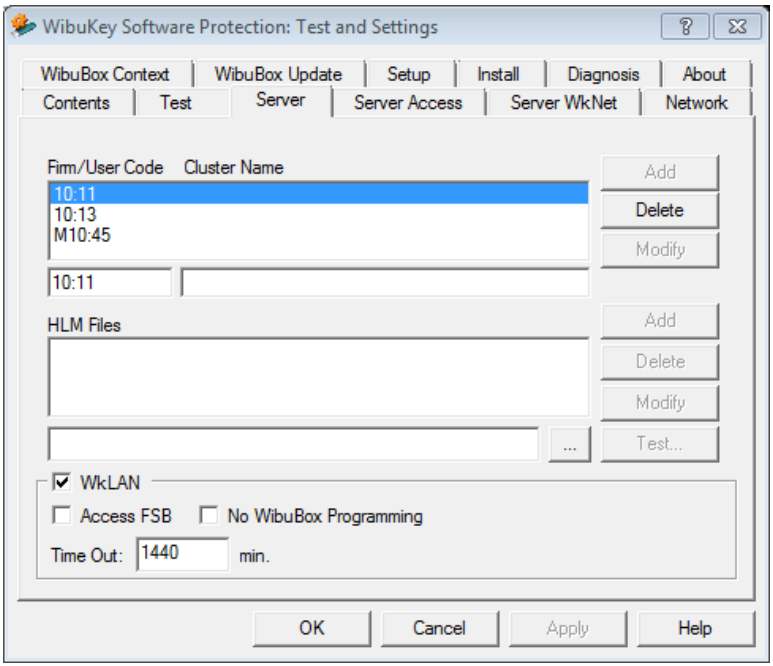


Figure 40: *Control Panel Applet*: Server page

The Server page allows the specification of cluster names, HLM files and **`WkLAN`** configuration.

1.3.1 Cluster Name

The field Cluster Name defines for a combination of FIRM CODE/USER CODE a name of a cluster. You can add, modify or delete cluster names, although only one cluster name is possible for a FIRM CODE/USER CODE.

1.3.2 HLM Files

The field HLM Files allows the definition of HLM files. You can enter a file name, or browse with the '…' button for existing files. To test whether the specified files is valid you may press the 'Test…' button.

1.3.3 WkLAN Configuration

The CheckBox **WkLAN** activates the **WkLAN** server process. When activating the CheckBox it is not possible anymore to access the server via TCP/IP. On the server page you can determine three parameters for **WkLAN**:

- *Access to FSB*. If this option is active, access to FIRM CODES in the range of 90-99 is possible from the network. This range is reserved for FIRM SECURITY BOXES (FSB). This option has to be activated when a FIRM SECURITY BOX is connected on a **WkLAN** server and every user should have access to this FSB via the network to program *WibuBoxes*.
- *No WibuBox programming* can be activated to avoid the reprogramming of the *WibuBoxes* that are connected to the **WkLAN** server from other stations.

The 'Timeout-Check-Box' specifies the timeout value in minutes. It frees automatically an allocated slot in the **WkLAN** client list if the allocating process has not accessed its slot in the specified time. If this entry is not set, a default timeout value of `1440` (24 hrs) is used.

1.4 Server access page

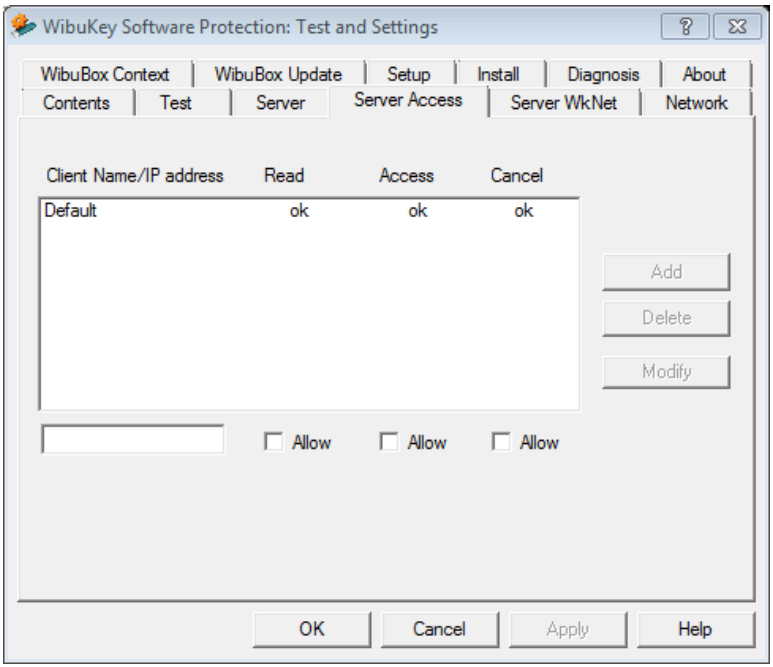


Figure 41: *Control Panel Applet*: Server Acces page

At this page client computers can be specified, by name or IP, which are allowed to access or cancel a license, respectively to read the *WibuBox* information.

1.5 Server `WkNet` page

The Server WkNet page allows the parameter setting for the server process of a network system.

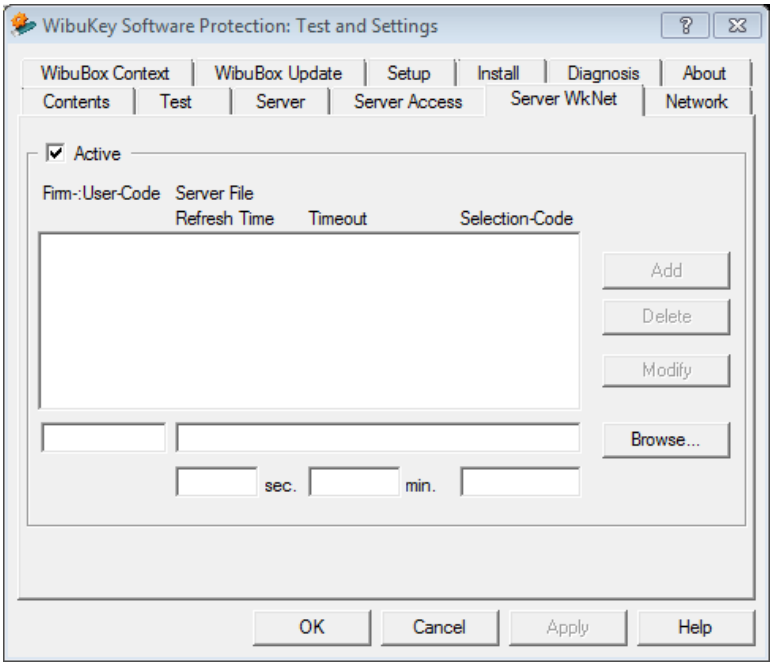


Figure 42: *Control Panel Applet*: Server WkNet page

1.5.1 `WkNet` configuration

In the **`WkNet`** section of this page the following parameters for the WkNet-Server can be set:

- The field 'Firm /User Code' contains encryption parameters for this WkNet process. The corresponding entries need to be programmed in a *WibuBox* connected to this station.
- The 'Server File' is the file which handles the communication between **`WkNet`** server and workstation. This file doesn't need necessarily to be stored locally on the **`WkNet`** server. It can be stored on any station within the network system. It is only necessary that the **`WkNet`** server as well as all workstations can perform read and write operations on this file. The Windows dialog for searching a file can be opened via the 'Search' button.
- The 'Refresh Time' specifies the time in seconds for the actualization of the contents of the **`WkNet`** server file. The values can be 10 seconds up to ca. 1 hour. The default value is 10 seconds.

If the value of the refresh time has been changed at the **WkNet** server, it is necessary to adapt the corresponding applications to the changed value as well.

- The value 'Timeout' specifies the time in minutes after which the server process cancels clients that have not accessed the server file in this time. The default value is 30 minutes. The value range of timeout depends on the selected refresh time. The timeout value has to be a minimum of three times and a maximum of 249 times the refresh time value. If this value isn't set no entries in the server file will be canceled.
- The 'Selection Code' is a further parameter that is needed for the encryption. This value is not being stored in the *WibuBox*. Because this value is used in the application, it is given by the software developer. The default value is 1234.

A **WkNet** server can control several server processes simultaneously. The values in the edit boxes are added to the list with the 'Add' button. The entries can be removed with the 'Delete' button. The 'Modify' button changes the currently selected entries.

1.6 Network page

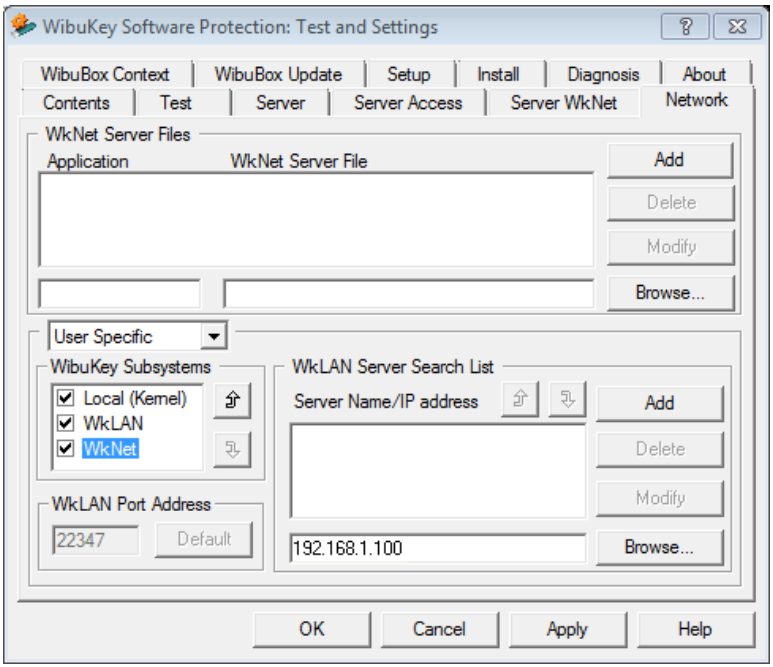


Figure 43: *Control Panel Applet*: Network page

On the Network page, client stations are configured to communicate with a **`WkNet`** or **`WkLAN`** server. The upper section of this page shows the settings for **`WkNet`**. On the bottom right side there is a search list for **`WkLAN`**. On the bottom left side it is possible to generally limit the search to certain *WibuKey* subsystems.

1.6.1 `WkNet` settings

To allow an application to access a *WibuBox* that is connected to a **`WkNet`** server, the application has to know which server file is necessary for the communication. The application has to provide the *WibuKey* drivers with the name of the application. The Network page allows to set the link between an *application* name and the corresponding **`WkNet`** server file. The values in the input windows can be transferred to the list with the 'Add' button. The *Browse* button removes entries and the 'Modify' button is for the modification of entries.

With the 'Search' button it is possible to open a Windows dialog for searching the **WkNet** server file.

Setting of WkNet server files for two applications

```
DrvTest  H:\WibuKey\WKNET\WKNET.DAT
RunMe    DOS:\TEST\WKNET.DAT
```

In the first sample the application with the name *DrvTest* is attached to the **WkNet** server file H:\WibuKey\WKNET\WKNET.DAT.
The second sample attaches the application *RunMe* to the file DOS:\TEST\WKNET.DAT on a Novell server.

1.6.2 WkLAN settings

A **WkLAN** server has to be found in a network system for **WkLAN**. This is comparable with **WkNet** where it is necessary to know which **WkNet** Server File is attached. Station names and IP addresses can be entered in the **WkLAN** server search list for searching for a **WkLAN** server. If a station name will be used a name service like DNS (*Domain Name Service*) or WINS (*Windows Name Service*) has to be installed. These name services handle the relation between station name and IP address.

Every server specification can contain *Broadcast* or *Multicast* information. The specification only has to be supported by the underlying TCP/IP implementation. If no server is set the default setting 255.255.255.255 (*LAN Broadcast*) is used. This *Broadcast* is transmitted only in the local network system and is not transferred by a router.

The values in the text windows will be transferred to the list with the 'Add' button. With the 'Delete' button they can be removed and the 'Modify' button is for the modification of entries. The network can be searched for **WkLAN** server with the 'Search' button. All available **WkLAN** server will be displayed in a list. The sequence of server addresses can be changed with the 'Up' and 'Down' buttons. The *WibuKey* drivers search within the list from top to bottom and use the first WkLAN server containing the searched entries.

TCP/IP Server Addresses

```
200.200.200.13
200.200.200.147
255.255.255.255
```

These settings search first for a WkLAN server on the station with IP address 200.200.200.13, then on the station 200.200.200.147 and finally in the whole local network system.

1.6.3 Subsystem settings

Beyond the settings for searching **WkLAN** servers the Network page can be used for the setting of *WkLAN port addresses* for the communication with **WkLAN** servers. The **WkLAN** management uses the default TCP port number 22347 for the access to a **WkLAN** server. If this number is already used by another TCP server it can be changed here. But this change has to be done also for the **WkLAN** server simultaneously. Otherwise there will be no communication between application and server.

WibuKey has two different subsystems:

- Kernel, the access to local *WibuBoxes*,
- **WkLAN**, the protocol based access to *WibuBoxes* via network systems and

The Network page offers the possibility to select subsystems to be used and in which order these subsystem should be searched. A check box before of the option means that this subsystem is active for searching. The sequence for searching the subsystems can be changed with the 'Up' and 'Down' buttons.

1.7 *WibuBox* context page

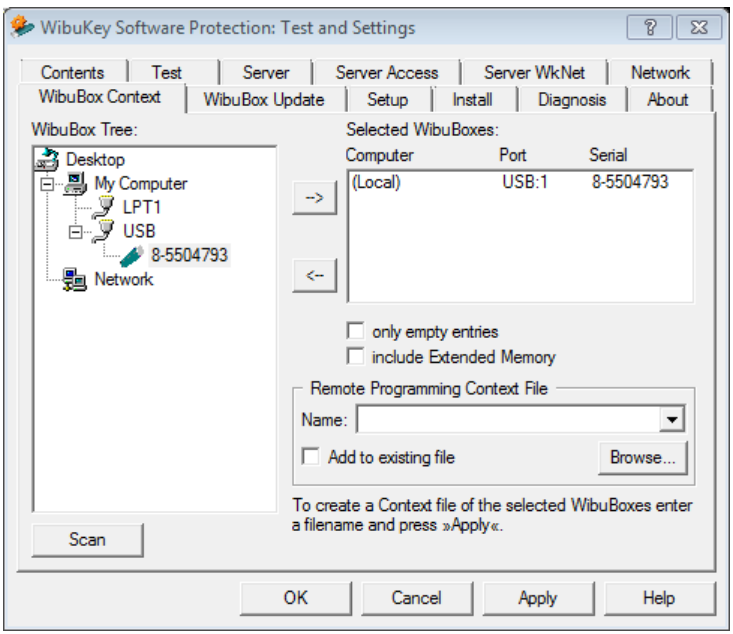


Figure 44: *Control Panel Applet*: *WibuBox* Context page

The *WibuBox* Context page is used to save contents of *WibuBoxes* in a remote programming context file. This page is one of the interfaces used by the software customer to create context files. The principle of remote programming is described on page 48, the realization on page 226.

By default all connected *WibuBoxes* are displayed in the window on the right side. *WibuBoxes* can be removed from the window and added from the *WibuBox* tree to the window with arrow keys. If one or more *WibuBoxes* are listed in the window and a correct file name has been entered the 'Apply' button appears activate. Pressing the 'Apply' button, the contents of the selected *WibuBoxes* will be encrypted and written to the specified file. If the 'only empty entries' option is activated a list of still available entries will be stored. The size of the context file is significant smaller. Via the option 'Add to existing file' it is possible to append data to an existing context file.

After this process has been finished successfully a message with information concerning the number of written bytes is displayed.

1.8 *WibuBox* update page

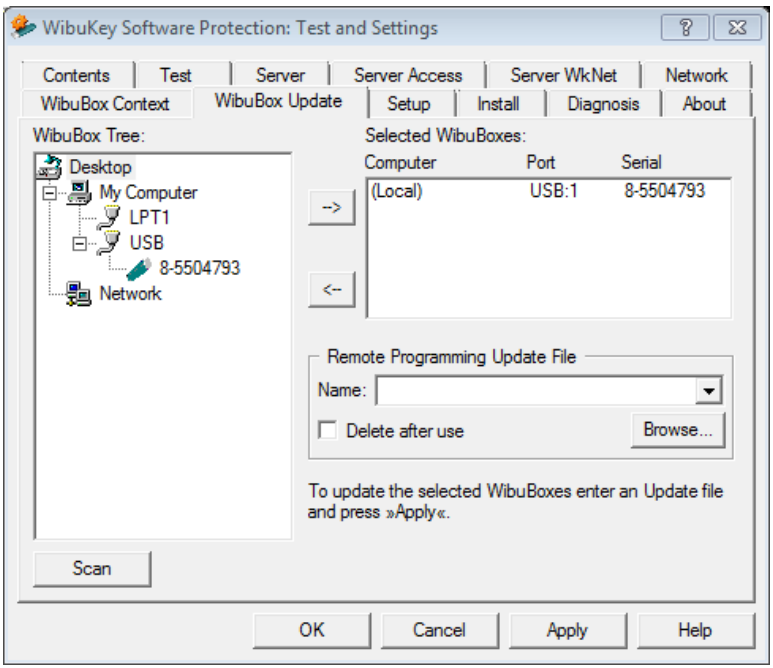


Figure 45: *Control Panel Applet*: *WibuBox* Update page

The *WibuBox* Update page is used to read remote programming update files in order to reprogram *WibuBoxes*. For additional information concerning the principle of remote programming see page 57.

By default all connected *WibuBoxes* are displayed in the window on the right side. As it can be done on the *WibuBox* Context page, the arrow keys allow to remove *WibuBoxes* from the window or to add *WibuBoxes* to the list. If one or more *WibuBox*(*es*) are selected and a proper update file is chosen with the 'Browse' button, the button 'Apply' will appear active. Pressing 'Apply' the remote programming update file is read and the corresponding *WibuBoxes* are reprogrammed. The activation of the 'Delete after use' option deletes the remote programming update file after the successful programming of *WibuBoxes*. It is not necessary to save the update file as it is useless after the reprogramming of the *WibuBox*.

A *WibuBox* with a determined remote programming update file can be reprogrammed **exactly once**. A repetition of this process is **not** possible.

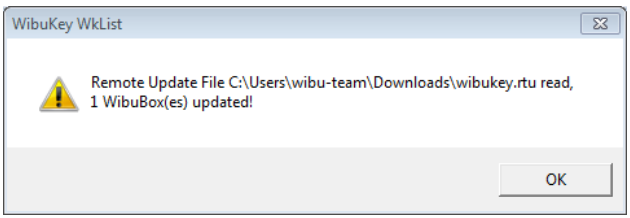


Figure 46: *Control Panel Applet*: WibuBox Update

A message with the information how many *WibuBoxes* are reprogrammed will be displayed after the successful process.

1.9 Setup page

This page implements all settings concerning the search for *WibuBoxes* and the configurations of every interface. If all *WibuKey* drivers have been installed properly and no *WibuBoxes* are found the settings on this page should be checked in the first step.

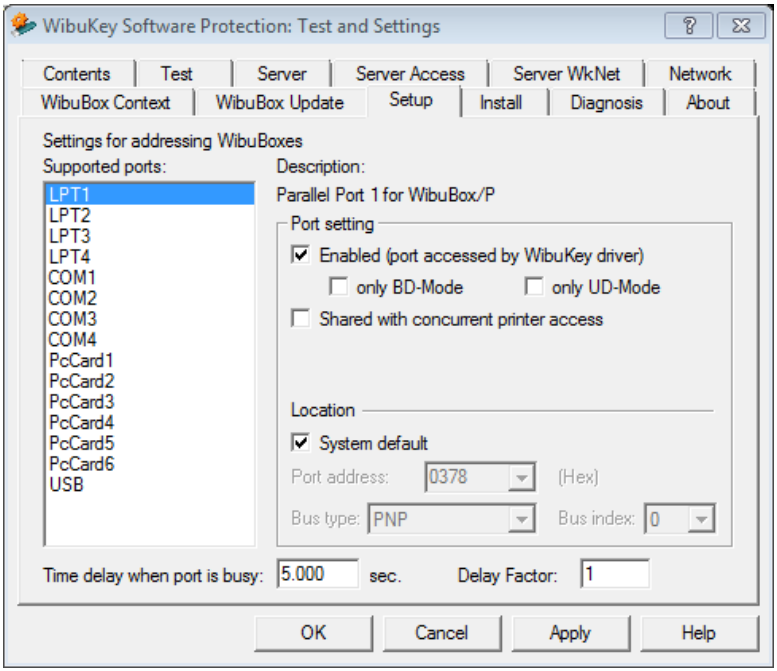


Figure 47: *Control Panel Applet*: Setup page

There are eight types of *WibuBoxes* listed on this page: The *WibuBox/P* and *WibuBox/RP* for parallel ports, the *WibuBox/SP* and *WibuBox/ST* for serial ports, the *WibuBox/U* and *WibuBox/RU* for USB bus, the *WibuBox/M* as PC Card and the *WibuBox/CI* as slot card.

1.9.1 General settings

The most important setting, available for every *WibuBox* variant, is the *Enabled* option. It enables the access on the corresponding interface.

Another option concerning all interfaces is 'Time delay when port is busy'. If there are problems with the communication, it is possible to enter values between `100` and `30,000` to slow down the access.

1.9.2 Special settings for *WibuBox/P* and *WibuBox/RP*

In the area 'Location' you can determine for the parallel ports the 'Port address' and for Windows NT/2000/XP/Vosta/Windows 7 the 'Bus type' and the 'Bus index'. By default the option *System default* is activated. Thus, the *WibuKey* drivers take the corresponding values of the operating system. In certain cases

it is possible that the operating system does not recognize the correct port or, for other reasons, sends a wrong or no address to the *WibuKey* drivers. In this case, you can enter the port address manually. Typical port addresses are `0x3BC` for LPT1, `0x378` for LPT2, and `0x278` for LPT3. Often the addresses for LPT1 and LPT2 are exchanged. Many state-of-the-art BIOS versions display, when starting the computer, the port addresses in a window.

In the area 'Port setting' you find the following options:

- 'Shared with concurrent printer access' manages the concurrent access of several applications to a parallel port. Normally, every application should ask the operating system for permission before accessing a port. If already another application accesses this port, the access is refused. The *WibuKey* drivers then wait for the 'Time delay when port is busy' and try again to access the port. If the access is refused again, the application gets a corresponding error message. If the access is permitted, the *WibuKey* driver allocates the port as long as the communication with the *WibuBox* lasts and then deallocates it. There are various applications wich don't stick to this rule: these applications allocate the port but don't deallocate it. Therefore you have the option 'Shared with concurrent printer access', in order to be able to communicate with the *WibuBox* despite such programs. If this option is activated, the operating system is not requested, but the driver accesses directly the port. Access conflicts and transmission errors can occur. Therefore, this option is deactivated.
- 'Only UD mode' determines, that the driver only searches within the unidirectional mode. That means, only *WibuBox/P* and *WibuBox/RP* are searched for, whose configuration attributes are set on unidirectional. *WibuBoxes* in bidirectional mode are not found. By default, this option as well as the option *only on BD mode* are deactivated. Thus, the driver searches in both modi for WibuBoxes.
- 'Only BD mode' determines that the driver only searches within the bidirectional mode. That means, only *WibuBox/P* and *WibuBox/RP* are searched for, whose configuration attributes are set on bidirectional. *WibuBoxes* in unidirectional mode are not found.

1.9.3 Special settings for *WibuBox/SP* and *WibuBox/ST*

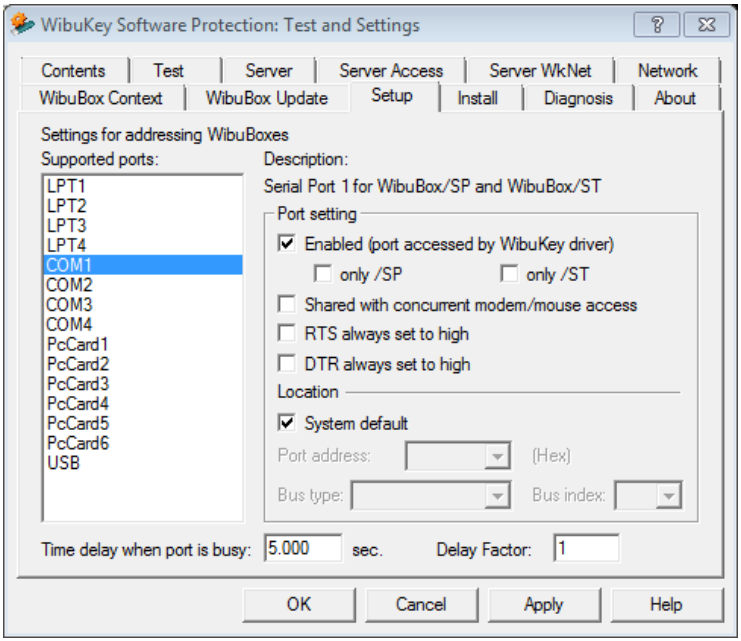


Figure 48: *Control Panel Applet*: Special Settings

In the area 'Location' you can determine the 'port address' for the serial port and the 'Bus type' and the 'Bus index' for Windows/NT/2000/XP/Vista/Windows 7. By default, the option 'System default' is activated. Thus, the *WibuKey* drivers take the corresponding values of the operating system. In certain cases it is possible that the operating system does not recognize the correct port or, for other reasons, sends a wrong or no address to the *WibuKey* drivers. In this case, you can enter the port address manually. Typical port addresses are `0x3F8` for COM1, `0x2F8` for COM2, and `0x2E8` for COM3, and `0x2E0` for COM4. Many state-of-the-art BIOS versions display, when starting the computer, the port addresses in a window. Mostly, you can modify the port addresses also in the BIOS setup.

In the area 'Port setting' you find beside *'Enabled'* the following options:

- The option 'RTS always set to high' specificates, that the *WibuKey* driver sets the line RTS of the serial port always to high. Normally, this option is not avtivated. It should only be activated when a *WibuBox/SP* or a *WibuBox/ST* couldn't be detected at the serial port. One of the reasons can be a weak power supply. Via the activated RTS line, a *WibuBox/SP* or a *WibuBox/ST* has another line at its disposal for its power supply. In this case, the *WibuBox* will often be found again.

- The option 'DTR always set to high' specifies, that the *WibuKey* driver always sets the line DTR of the serial port to high. This option performs the same task as 'RTS always set to high' for another line. By default, this option is deactivated.
- The option 'only/SP' or 'only/ST' finds only *WibuBox/SP* and *WibuBox/ST* respectively.
- The option 'Shared with concurrent modem/mouse access' manages the concurrent access of several applications to a parallel port.

1.9.4 Special settings for *WibuBox/U* and *WibuBox/RU*

Under Windows 98/Me and Windows 2000/XP/Vista/Windows 7, the access to a *WibuBox/U* or *WibuBox/RU* can only be activated or deactivated.

1.9.5 Special settings for *WibuBox/M*

Under Windows 95/98/Me and Windows NT, the access to a *WibuBox/M* can only be activated or deactivated. Under Windows NT, the Award Cradware 6.00,009 is necessary.

1.9.6 Special settings for *WibuBox/CI*

On PC the *WibuBox/CI* will be recognized as well as a *WibuBox/P*. Since the BIOS doesn't recognize the *WibuBox/CI* as a parallel port, the port address must be entered manually under on of the parallel ports. The port address can be specificated via the jumpers on the card. By default, the address is `0x330H`. There are 16 possibilities within the range of `0x230` to `0x33C`.

1.10 Install page

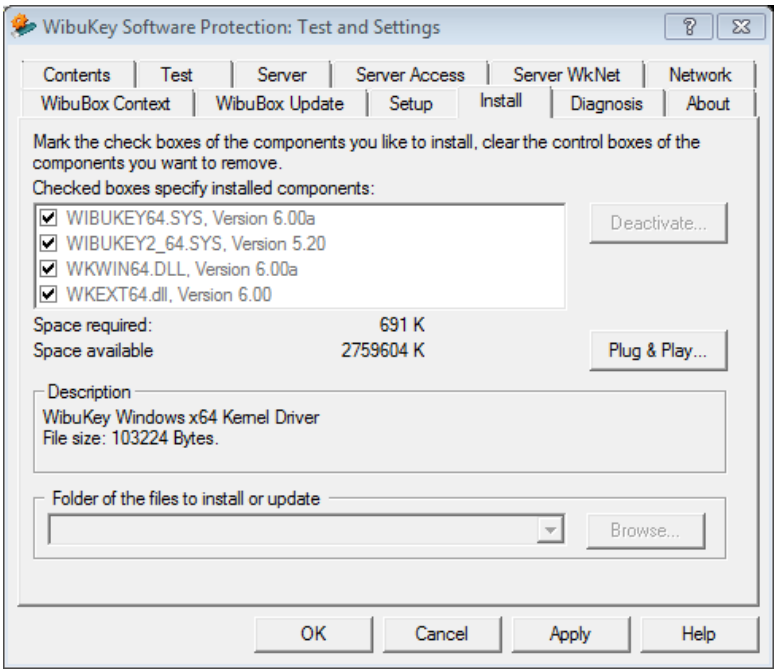


Figure 49: *Control Panel Applet*: Install page

This page installs, deinstalls or checks the *WibuKey* drivers. A description of which drivers are necessary for which operating systems can be found in detail on page 302. At this point only the functionality of the Install page is explained.

The list of installed components shows all driver components for the current operating system. If a component is installed correct, it is marked by a checkbox and the software version is displayed. Below there is a short description of this driver component together with the file size. The memory requirements of all drivers and the free memory on the system disk are also displayed. If one component is not checked, it is either not installed in the right way or not installed at the right place.

Plug&Play

With the 'Plug & Play' button, not visible for Windows NT 4.0, you can reset the plug&play settings for your *WibuBox/U* or *WibuBox/M*. You have to remove the box from the port before pressing the settings can be refreshed. After this step the next time you plug in your *WibuBox* the plug&play mechanism will recognize the hardware and install the *WibuBox* driver.

1.10.1 Activation / deactivation of the *WibuKey* Kernel driver

This dialogbox appears when *activating* the *WibuKey* kernel driver (`WIBU.KEY.VXD` or `WibuKey.SYS`) on the Installation page. There are three possibilities to start the kernel driver. The *deactivation* of the kernel driver is realized automatically.

1.10.2 Installation / update of *WibuKey* drivers

For the installation or update of the current drivers with the latest drivers it is necessary to activate the corresponding checkboxes and to enter the path to the source files. The source files may be packed with the Microsoft program Compress. The installation of the corresponding drivers happens through the *Apply* button. Any problems will be displayed in a corresponding dialog.

All drivers are installed properly, but only the English and German language files are being installed. All required language files are installed by using the *WibuKey* Runtime Kit or the *WibuKey* Driver Update Kit.

1.10.3 Removal of *WibuKey* drivers

The installed *WibuKey* drivers can be removed by clearing the concerning checkboxes and pressing the 'Apply' button. The driver files can only be removed when no application besides the *Control Panel Applet* itself is currently using them.

1.11 Diagnosis page

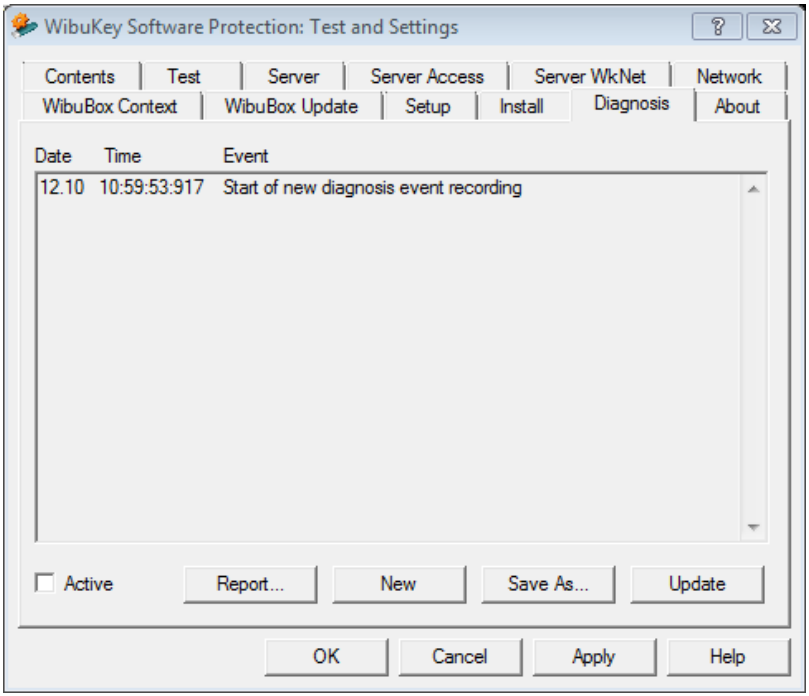


Figure 50: *Control Panel Applet*: Diagnosis page

The drivers of *WibuKey* can return information about problems concerning the access to *WibuBoxes*. This information is a status report about driver's internal processes. Usually this diagnosis mode is not activated and no messages will appear. The diagnosis can be activated via this page.

To activate the diagnosis mode the 'Active' option in the text field on the left below side must be switched on and the 'New' button must be pressed. All following operations will be controlled and error messages will be recorded. To read the diagnosis record the 'Update' button must be chosen. Diagnosis events can be stored via the 'Save' *as* button to an external file. This file can be send by mail or fax to the software developer or to Wibu-Systems, together with a MSD diagnosis file.

1.12 About page

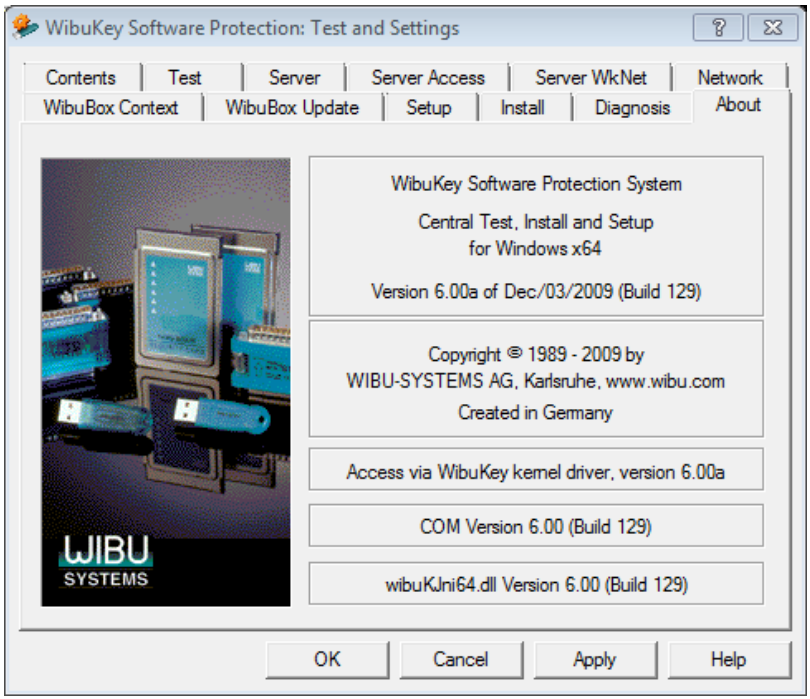


Figure 51: *Control Panel Applet*: About page

This page displays the version of the *WibuKey Control Panel Applet* and the version of the *WibuKey* drivers which access the *WibuBox* hardware. Moreover, the version of the used (Kernel) driver and the installed COM version are displayed.

2 WKU commandline

WKU is the universal *WibuKey* console application for the *WibuKey* end user. Most of the functions of the *WibuKey Control Panel Applet* can be realized with **WKU**. The program is available as

- `WKU.EXE` for DOS and as
- `WKU32.EXE` for Windows 95/98/Me/NT/2000/XP/Vista/Windows 7.

WKU.EXE can be used on Windows NT and Windows 95/98/Me in combination with the installable DOS driver **WKDOS.EXE**. It is an ideal test program to find out whether the access from DOS programs via the kernel driver to the *WibuBox* works well.

WKU has a lot of features, which are also available in the *WibuKey Control Panel Applet*. Such features are:

- Listing of all *WibuKey* ports and number of connected *WibuBoxes* if **WKU** is called without argument.
- Displaying of *WibuBox* contents for one or more *WibuBoxes* at one or more ports with the `WKU List` command.
- Checking the encryption functionality of all or of specified *WibuBoxes* in greater detail by a specific number of checks with the `WKU TEST` command,
- Listing and Modifying of USER DATA entries in the *WibuBox* with the `WKU DATA` command.
- Changing of configuration attributes for a *WibuBox* with the `WKU CONFIG` command,
- Creating a remote programming context file and updating a *WibuBox* with a remote programming update file to change the contents of the *WibuBox* with the `WKU REMOTE` command.
- Reading and saving diagnostic results of the resident DOS driver `WKDOS.EXE` with the `WKU DIAGNOSE` command.
- Searching for **WkLAN** server on the local network with the `WKU NETWORK` command.
- Resetting the driver or giving *WibuBoxes* free which are locked permanently with the `WKU RESET` command.
- Testing and port setting of the physical ports LPT1 to LPT4 and COM1 to COM4 with the `WKU PORTS` command.
- Installing the kernel driver `WibuKey.SYS` on Windows NT with the `WKU32 INSTALL` command.
- Deinstalling the kernel driver on Windows NT with the WKU32 DEINSTALL command. This is the counterpart to `WKU32 INSTALL`.
- Starting the Kernel driver on Windows 95/98/Me/NT/2000/XP/Vista/Windows 7 with the WKU32 START command.

- Stopping the Kernel driver on Windows NT with the `WKU32 STOP` command.
- Displaying the version information of the Kernel driver on Windows 95/98/Me/NT/2000/XP/Vista/Windows 7 with the `WKU32 CHECK` command.

For more details about the use of **WKU** see the online help of this application. A general help can be displayed via the command:

```
WKU /?
```

A subcommand specific help can be printed by entering the subcommand name, followed by the help option **/?**, for example:

```
WKU TEST /?
```

2.1 Port address setting

The **PORTS** command is used for the correct interrogation and addressing of ports to which *WibuBoxes* are connected. **WKU PORTS** is especially needed, if the port addresses cannot be correctly recognized by some BIOS variants. In such cases, the printer output won't work at such interfaces.

With **WKU PORTS**, the addresses determined by the BIOS can be issued. For this purpose, **WKU PORTS** is simply entered without any argument. If the specified values are identical with and without a connected BOX after booting, the BIOS has recognized the interfaces correctly. If they are not identical, **WKU PORTS** must be used to subsequently recognize the interfaces and enter them in the BIOS-RAM. Only then they will be available for printing or similar functions and can be recognized by the *WibuKey* drivers. This concerns mainly parallel interfaces (LPT). It is expedient to enter such a **WKU PORTS** start in `AUTOEXEC.BAT`.

The following syntax is used to set **WKU PORTS**:

WKU PORTS ((**LPT**|**COM**) [=(*HexAddress*|*|**0**)[,]…])…

If just **LPT** or **COM** is specified without additional arguments, **WKU PORTS** searches for interfaces at the following addresses and if the result is positive, sets them in the BIOS-RAM:

Port	Default Address
LPT1	0x3BC
LPT2	0x378

Port	Default Address
LPT3	0x278
COM1	0x3F8
COM2	0x2F8

If the equal sign is used, **WKU PORTS** searches for interfaces at the specified addresses and sets them in the BIOS-RAM. The addresses must be in the sequence LPT1 to LPT4, or COM1 to COM4.
Addresses are fundamentally to be entered in the hexadecimal notation but as pure numbers without any leading prefix such as "0X" or trailing "H" (see examples below). The address entry "0" will deactivate the corresponding interface.
If an asterisk (*) is entered, instead of an address, the default address from the table above will be used.

The option **/?** will supply a brief description of the program, together with examples on the monitor.

WKU PORTS only runs on DOS and in AUTOEXEC.BAT on Windows 95. For OS/2 the correct addresses of these interfaces must be entered in the environment.

Port address setting

```
WKU PORTS LPT
```

WKU checks and sets the parallel interfaces to the default addresses.

```
WKU PORTS LPT=378,278 COM=2E0,2F8
```

WKU checks and sets LPT1 to the address 378H, LPT2 to 278H, COM1 to 2E0H and COM2 to 2F8H. Other possible interfaces (LPT3 etc.) are deactivated.

```
WKU PORTS LPT=*,0,27C COM=2E8,*,0,2E0
```

WKU checks and sets LPT1 to the default address 3BCH, LPT3 to 27CH, COM1 to 2E8H, COM2 to the default address 2F8H and COM4 to the address 2E0H. COM3 and LPT2 are deactivated.

WKU PORTS normally operates only with LPT1 to LPT3, except if four addresses are specified in the argument following LPT. The set LPT4 value can be read back via the entry

WKU PORTS LPT=*,*,*,*

The PC continually supports only LPT1 to LPT3. The use of LPT4 leads to difficulties with IBM-PS/2 systems and a number of other systems which employ ESDI or SCSI-Disk adapters.

2.2 *WibuBox* diagnosis

A *WibuBox* can be checked independent of a protected program. The **WKU** console application is used for diagnosis on DOS. **WKU** sends any randomized sequences to the *WibuBox*. The test program analyses the received data carefully. The result is a text file which lists the communication problems between protected Windows or DOS applications and *WibuBoxes*. Such files should be sent for analyzing to Wibu-Systems.

```
WKU DIAGNOSE LIST
```

lists the data on DOS only at screen during **WKDOS** is installed and the buffer is available.

When the output overruns the height of the screen, the output can be stopped page by page via the option **/PAUSE**. With the subcommand **WKU DIAGNOSE RESTART** the diagnosis buffer is cleared and the diagnosis option is started again. Is the subcommand **LIST** specified with a file name, the diagnosis results are stored to this file:

```
WKU DIAGNOSE LIST WIBU.RPT
```

***WibuBox* diagnosis with WKU**

If a user has problems on DOS with a *WibuBox* at LPT2, the following command sequence can be used to record diagnostic data:

```
WKDOS /SD /M
WKU TEST LPT2
WKU DIAGNOSE LIST WIBU.RPT
MSD /P MSD.RPT
WKDOS /R
```

The generated files `WIBU.RPT` and `MSD.RPT` should be copied to a floppy disk and sent to Wibu-Systems.

For `WKDOS` options and their syntax enter in a DOS box

```
WKDOS /?
```

2.3 Kernel driver installation

WKU32 can install and start the Kernel driver of Windows NT. **WKU32** as a console application is an alternative to the *WibuKey Control Panel Applet*. **WKU32** can also be used to manage kernel driver remotely over the network.

The installation or deinstallation of the WibuKey.SYS kernel driver on Windows NT can only be realized by a user with system administrator rights. Normally a Windows NT administrator or domain administrator has the proper user rights to install or uninstall a kernel driver. The Windows NT User Manager can be used to give this user right to other users. The exact name is Load and unload device drivers or *SeLoadDriverPrivilege*.

The Kernel driver commands of WKU32 have the following syntax:

```
WKU32 INSTALL KERNEL DRIVER [AUTOMATIC|MANUAL]
```

This command (re-)installs the WibuKey.SYS kernel driver which must reside in the driver folder of Windows NT.

If AUTOMATIC has been specified or no specification is given, the driver will be installed for automatic start. This will be done automatically if the system is booted. This variant increases the timing delay of the system startup slightly. It is suitable for users, who do not have the user rights to install or to start a kernel driver.

Specifying the option MANUAL, the driver will be installed for a manual start. This done if the first application wants to access the *WibuBox*. This variant requires that the user who has called the *WibuKey* software has user rights to install and to start a Windows NT kernel driver.

```
WKU32 DEINSTALL KERNEL DRIVER
```

This command deinstalls an installed driver from the Kernel memory of Windows NT without deleting the driver file in the driver folder. The file can be installed later with the INSTALL command.

```
WKU32 START KERNEL DRIVER
```

This command starts the Kernel driver. Normally the Kernel driver is started during system startup or if an application tries to access it. Nevertheless, in some case it might be useful to start the Kernel driver manually.

```
WKU32 STOP KERNEL DRIVER
```

This command stops the Kernel driver. If any application is still accessing the Kernel driver it can't be stopped and a warning message will be displayed.

```
WKU32 CHECK KERNEL DRIVER
```

This command starts the Kernel driver and displays its version number. This command works for both the Kernel driver on Windows 95/98/Me/NT/XP/Vista/Windows 7.

AT ComputerName

With the **AT** command all Kernel driver commands except **CHECK** can be redirected over the network to another workstation. **AT** can be specified before the main command:

```
WKU32 AT ComputerName STOP KERNEL DRIVER
```

or after the main command:

```
WKU32 DEINSTALL AT ComputerName KERNEL DRIVER.
```

2.4 Changing configuration attributes

To change configuration attributes the **WKU CONFIG** subcommand should be used.

WibuBox/P

To activate the bi-directional mode, use the **+BD** or the **-UD** specification:

```
WKU CONFIG LPT1 +BD
WKU CONFIG LPT1 -UD
```

To activate the unidirectional mode, use the **+UD** or the **-BD** specification:

```
WKU CONFIG LPT1 +UD
WKU CONFIG LPT1 -BD
```

The **WKU CONFIG** operation works only with one WibuBox, if more than WibuBoxes are connected, the index of the WibuBox to modify must be specified, for example:

```
WKU CONFIG LPT1:2 +UD
```

Some interface circuits like the XIRCOM adapter at the parallel port are illegally addressed if a WibuKey driver searches for WibuBoxes in the unidirectional mode. If at such a port only new WibuBoxes are connected and the bi-directional mode is supported by the driver, then the unidirectional mode should be switched off via the WibuKey environment (with the option **/B**, which is stored after the port address in the contents of the WKLPT variable).

```
WKLPT = */B,*,*,330
```

Activates only the bi-directional mode at LPT1, for LPT2 up to LPT4 both modes remain active. If only the unidirectional mode should be used, then the **/U** option should be set instead.

If IPX is loaded for XIRCOM network adapter, the LPT port is „deleted“. **WKU** displays LPT1 at address `0x0001`. To access the WibuBox it is necessary to specify the LPT address in the WibuKey environment.

```
WKLPT = 378/B,*,*
```

The LPT addresses are normally `378`, `278` or `3BC`. The address of the own LPT port is displayed by **WKU** before loading IPX.

WibuBox/SP

To activate the Terminal Mode, use the **+TM** or the **-MO** specification:

```
WKU CONFIG COM1 +TM
WKU CONFIG COM1 -MO
```

To activate the Mouse Mode, use the **+MO** or the **-TM** specification:

```
WKU CONFIG COM1 +MO
WKU CONFIG COM1 -TM
```

The **WKU CONFIG** operation works only with one WibuBox, if more than WibuBoxes are connected, the index of the WibuBox to modify must be specified, for example:

```
WKU CONFIG COM1:2 +TM
```

Part VIII Distributing protected applications

Additionally to your protected software, you install on the client's side also the *WibuKey* drivers and help programs. Most of the time the question during the installation of an application is not which files to install, but how to be sure that all files are in the right place. In this chapter different methods to install *WibuKey* software components are explained. All of the various components are listed and explained, and the proper installation methods are discussed.

1 Preparing for installation

This section gives a short introduction in the different installation methods. It is important to be familiar with them before a decision is made on how to install software.

Wibu-Systems recommends using the *WibuKey* Runtime Kit for the installation of the drivers. The current version can be downloaded from www.wibu.com.

1.1 Installation via *WibuKey* Runtime Kit

The simplest way to install the *WibuKey* software components are the disks prepared by Wibu-Systems.

The *WibuKey* Runtime Kit contains all necessary driver files and a special set up program which installs all software components that are needed for the current operating system (Windows 95/98/Me/NT/2000/XP/Vista/Windows 7).

The latest version can be downloaded from the Wibu-Systems web site under www.wibu.com at any time. This is possible for software developers as well as their customers. The *WibuKey* Runtime Kit on the web does not contain the application specific settings. We recommend generating an application-specific *WibuKey* Runtime Kit (`SETUP.INI` and `WibuKey.INI`) adjusting the configuration files.

1.2 Direct copying of driver files

Wibu-Systems strongly recommends using the *WibuKey* Runtime Kit installing the *WibuKey* software. If the target platform for an application is Windows/95/98/Me/NT/2000/XP/Vista/Windows 7 it is not possible to do the complete installation by simply copying the *WibuKey* software components to their destination folders. It is strongly recommended that the install program checks the version of the driver files that are already installed.

2 *WibuKey* software components

This section lists all *WibuKey* software components.

File Name	Description
`WKWIN.DLL`	*WibuKey* driver for 16 bit Windows
`WKWIN32.DLL`	*WibuKey* driver for Win 32
`WibuKey.DLL`	*WibuKey* COM API
`WibuKey.SYS`	*WibuKey* Kernel driver for Windows NT
`WibuKey2.SYS`	*WibuKey* kernel driver for Windows 98/Me/2000/XP/Vista/Windows 7
`WibuKey.VXD`	*WibuKey* kernel driver for Windows 95/98/Me
`WKDOS.EXE`	Installable DOS driver for Windows 95/98/Me/NT/2000/XP/Vista/Windows 7, for diagnosis and extended purposes.
`WIBUKE32.CPL`	Extension to the 32 bit Windows system control
`WKU.EXE`	User management console program for DOS
`WKU32.EXE`	User management console program for Windows 95/98/Me/NT/2000/XP
`DECRYP32.EXE`	Encryption program for protected files for Windows 95/98/Me/NT/2000/XP/Vista/Windows 7
`FCRYPT32.EXE`	FCRYPT variant as 32 bit Windows application
`WKSVNW.NLM`	*WibuKey Server* process for Novell Netware
`WKSVW32.EXE`	*WibuKey Server* process for 32 bit-Windows
`WibuKey.INI`	*WibuKey* initialization file for Windows

The listed files are available for distribution. The software developer who possesses a Wibu-Systems license contract has the right to deliver *WibuKey* software components with his software.

All these software components and their updated versions can be downloaded freely from the Wibu-Systems web site. All other programs or files (e.g. **`WKCRYPT.EXE`** or **`WKFIRM.WBC`**) are strictly excluded.

3 Installation on Macintosh

The user receives the following files:

Mac OS 8 & 9

Component	Description
`WkMacList`	Application for programming and remote programming of *WibuBoxes*
`WkSvMac`	*WibuKey Server* processes
`WibuKey.INI`	`INI` file with settings
`WkMACLIB`	Library and driver to access ADB and USB ports for system extension folder
`USBWibuKeyDriver`	Class driver for *WibuBox/U* at USB for system extension folder
`USBWibuKeyDriver2`	Class driver for *WibuBox/RU* at USB for system extension folder

Mac OS X

Component	Description
`WkMacLibX.framework`	Library and driver to access USB ports for system extension folder
`WkMACLIB`	Library and driver to access USB ports for system extension folder

4 Installation on Linux

On the installation CD the file `index_en_devel.html` in subfolder Linux contains all information needed for an installation on Linux. This includes

- information on *WibuKey* driver supporting the *WibuBox /U /U+ /RU /RU+ /P /P+ /RP /RP+ /ST* and *CmCard/M* and requirements
- installation packages for SuSE, Red Hat, etc. in form of RPM packages, and for Debian derivates in form of DEB packages
- *WibuKey* Unix source driver
- Information on integrating the *WibuKey* driver into your own install package

5 Installation methods on Windows

5.1 Description of the *WibuKey* Runtime Kit

The *WibuKey* Runtime Kit is the universal tool to install and deinstall the *WibuKey* software components and to create a proper working environment for *WibuKey*. It contains a 32-bit set up program as well as all files that are required for any of the 32 bit Windows operating systems. The set up program has a user interface that informs about the set up progress.

In order to simplify the use of *WibuKey* Runtime Kit you may adjust the set up to your own requirements via a number of configuration options. The following examples are possible:

- You can chose between one disk with the minimal version for the driver installation up to various disks or a directory on CD.
- Selection of all software components, interactive as well as by an INI file.
- Selection of particular languages by files, interactive or by an INI file.
- Installation of the WibuKey server processes and the corresponding WkNet server files and the HLM files including the adjustment of a `WibuKey.INI`.
- Start in the background and repression of all outputs of dialog.
- Controlling the set up program on the client's side is realized via the `SETUP.INI` file. The following two chapters show the corresponding options and commands.

Since the files of the set up program possibly have to be delivered on disks, each file has to be specified with the number of your disk in the file `SETUP.INI`.

The set up program writes a return value in the registry (`HKLM/Software/Wibu-Systems/WibuKey/Setup/SetupStatus`).

The entry shows if the set up still runs if it has been realized successfully or if there is an error. The return value is especially important when the user interface will be disabled.

A *WibuKey* directory is created. This directory is with regard to Windows 2000/XP/Vista/Windows 7 and the Microsoft compatibility guidelines inevitably necessary. The default path for the directory is "`%Program Files%WibuKey`". The end user may change this path via a dialog. The path cannot be preset. If no user interface is shown, the default path is used. If already a *WibuKey* directory exists, this directory will be used. In this case, the user cannot modify this directory. This prevents that various *WibuKey* directories will be created.

The *WibuKey* directory contains various subdirectories, in which the *WibuKey* software is installed. The driver files are installed into the Windows root folder or in its subfolders.

The set up program generates a log file which contains all installed files. The file `SETUP.LOG` is to be found in the `WibuKey\SETUP` directory. At the beginning of the log file is the installation date and the GUID of the developer, then follows a list of the installed files and maybe some additional information.

Up to now there have been several methods to install the *WibuKey* drivers on the client's side and various programs has been necessary for the proper installation.

The previous installation programs:

- *WibuKey* control field
- `WkSet16 / WkSet32`
- `WkSet16 / WkSet32`
- and the `CPL API`

will not be developed anymore. These programs do not support the installation of various languages and the installation of *WibuKey* help programs as the server process. See page 306 for a description of the replacement of the old programs by the new ones via the *WibuKey* Runtime Kit.

5.2 Commandline parameters

The main control tool for the set up is the file `SETUP.INI`. See the next section for more details on this file. In the commandline can be specified only a path to a new – modified, for example statically generated - `SETUP.INI` or control parameters for the deinstallation. The following parameters are available:

Parameter	Description
`INI file`	specifies the initialization file with all other parameters. **Important:** Always specify the complete path.
`/R[S][:GUID]`	starts the deinstallation of the WibuKey files. The suboption S deactivates the user interface. GUID specifies the Id of the developer or the program which has installed the *WibuKey* software. If no GUID is specified and if there is only one WibuKey version installed, this one will be deinstalled. If

Parameter	Description
	various versions are installed a warning is given.
/?	indicates the online help.

5.3 Parameters of SETUP.INI

The set up program can be controlled by an INI file. The file SETUP.INI must be in the same folder as SETUP.EXE/SETUP32.EXE. The following settings can be specified:

Value	Description
[General]	Paragraph with general settings
AppId	GUID of the developer or the program that installs the *WibuKey* software.
AppName	Identifier of the developer or the program which installs the *WibuKey* software. This parameter is the readable counterpart to AppId.
Language	List of the languages which are offered by *WibuKey* for the installation. This option is important if the user interface has been disabled. You will find a list of the supported languages and their abbreviations in Appendix F.
Readme	determines if the Enduser Setup Readme help file will be displayed. The default value of 0 enables this feature. 1 will uncheck the checkbox at the dialog page and 2 will remove the whole dialog page.
ForceOverwrit	specifies if existing settings should automatically be overwritten in the registry (0=do not overwrite is default, 1=overwrite).
Gui	enables the graphical user interface (1 is default) or disables it (0). If this value is set to 0, ForceOverwrite is automatically set to 1.
ErrorMessages	enables (1, default) or disables (0) error messages.
[Files]	Paragraph which specifies the files to be installed.

Value	Description
FileN	Information about a file to be installed: The first name is the original name of the file, then follows the number of the disk on which this file is stored, at the end is the name of the zipped file. Only files mentioned here, can be installed.
[Tools]	Paragraph with tool installation options.
NoToolSelect	prevents that the tool selection dialog is displayed. All components, whose files are found on disk, will be installed.
WkSvW32Service	determines, if WkSv32 shall be installed as service (1) or not (0).
Shortcuts	determines if a program group and short cuts for the programs shall be generated (1, default) or not (0).
[Driver]	Paragraph with driver installation options.
Win32	determines if the WkWin32.dll shall be installed every time irrespective of the current operating system (0= install automatically, 1= install every time)
WkDos.exe	determines if WKDOS.EXE should be installed on Windows 95/98/Me/NT (1, default) or not (0).
WkWin.dll	determines if WKWIN.DLL should be installed on Windows 95/98/Me/NT/2000/XP/Vista/Windows 7 (1, default) or not (0).
SetupLang	determines what language is used for the installation dialog. The default value is default – the set up will try to determine by itself which language is used on the system. It also can be specified the WIBU mnemonic for a specific language.
SourcePath	determines the path to the source files. Default value is the same folder as the one for SETUP.EXE.

5.4 Transferring *WibuKey* system settings

If the set up program of the *WibuKey* Runtime Kit finds a `WibuKey.INI` file it transfers all settings from this file to the registry of Windows 95/98/Me/NT/2000/XP/Vista/Windows 7. If any settings already exist with a different value a dialog will appear to ask whether this value should be overwritten. This can be avoided by setting the `ForceOverwrite` parameter in the `SETUP.INI` to `1`. This may change important system settings and should be used very carefully. The best way is that the `WibuKey.INI` holds only settings that are absolutely necessary for your application, e.g. network settings. Then no collisions with existing settings should occur.

In Appendice A you will find a detailed description of the `WibuKey.INI`.

The following table shows the settings which are adopted from a client specific `WibuKey.INI` file:

Topic	Description
`[General]`	General settings
`[Driver]`	Driver settings
`[WkLAN]`	General **`WkLAN`** settings
`[WkLAN Client]`	**`WkLAN`** Client settings
`[WkNet Clients]`	**`WkNet`** Client settings
`[Server]`	General Server settings
`[WkLAN Server]`	**`WkLAN`** Server settings
`[Server HLM files]`	Server settings for HLM, see also section "Installation of **`WkNet`**/`HLM` files"
`[WkNet FC:UC]`	WkNet Server settings, see also section "Installation of **`WkNet`**/`HLM` files"

The set up also can install **`WkNet`** server files and adjust the path in the registry to the system. So that a **`WkNet`**/**`WkLAN`** server process is completely functional after the installation. The paragraph "Installation of **`WkNet`**/`HLM` files" describes the conditions and the operating principle.

5.5 Installation of `WkNet`/HLM files

The *WibuKey* Runtime Kit can install in addition to the network server processes also the necessary **`WkNet`** server or HLM files. The files have to be

allocated in the same folder where the set up program is allocated. At the moment, the number of files is restricted to 10 **`WkNet`** server files or 10 HLM files.

HLM information stored in the EXTENDED MEMORY of the /+.variants of *WibuBoxes* is automatically deteced by the *WibuKey Server* process at startup.

5.6 Return values

Most time it is difficult to call another application and get the proper return code. Therefore the set up program of the *WibuKey* Runtime Kit writes its status to the registry or to the `WibuKey.INI` file. The exact path on 32-bit operating systems in the registry is:

`HKEY_LOCAL_MACHINE\Software\WIBU-SYSTEMS\WibuKey\Setup\SetupStatus`.

When setup starts it initializes the `SetupStatus` value with `-1`. When all driver files are correctly installed it changes the value to 0. In case of an internal error `SetupStatus` will be less than zero, in case of an error reported by the CPL-API it will be a positive error. The following return values are possible:

Code	Description
`0`	all drivers have been installed successfully
`-1`	setup is still running
`-2`	unable to find CPL source file
`-3`	error during CPL access
`-4`	error accessing CPL copy source or target
`-5`	error copying CPL
`-6`	unable to access CPL dynamically
`-7`	unable to access CPL-API function
`-8`	error accessing registry
`-9`	unable to delete file
`-10`	error accessing CPL help file, copy source or target

Code	Description
-11	error transferring WibuKey.INI to registry
-12	error copying files from source to temp
-13	error accessing temporary directory
-14	restart necessary

5.7 Deinstallation

During the installation of the drivers files on Windows 95/98/Me/NT/2000//Vista/Windows 7 the *WibuKey* Runtime Kit will store information in the registry. In the Control Panel under the "Add/Remove Programs" icon an entry "*WibuKey* Setup (WIBUKE.REMOVE)" is created. A double click on this entry will remove all WibuKey driver files and all registry entries from the system.

6 *WibuKey* Runtime Kit setup

The setup program can be obtained in both 16 and 32-bit versions. On 16 bit systems only the drivers can be installed, while on 32bbit systems the disposal of the entire system is available. Controlling the setup program from the end user site is completed through the SETUP.INI initialization file. This section shows the appropriate options, or rather the commands. Because the data in the setup program may possibly have to be saved onto disks, all data must be labeled in the SET UP.INI data file. Please use the number of your disk as the label for this data.

WkNet data can also be installed through the setup program. To do that, go to Tools on the tool bar then click the option WkNet=OwnWkNetData. Copy this data and an appropriate WIBUKEY.INI into the first disk.

The setup program enters a return value in the registry or in the WIBUKEY.INI (16 bit). By accessing this entry, you can find determine if the setup is still running, if it was successfully ended, or if an error has been reported. The return value is especially important if the user interface is disconnected. This directory is in accordance with Windows 2000 and Microsoft compatibility guidelines and a necessary component, even when this it seems to be an unnecessary addition. A WIBUKEY-Directory is always open/installed. The standard path for this directory is "%ProgramDirPath%\WIBUKEY", the end user can also change this path using a dialogue. A customer cannot stipulate the path. In the case that a user interface is not showing up, the default path is

the standard path. If a WIBUKEY directory already exists, this directory is the default directory. Neither the end user nor the software developer can change this directory (it is much more difficult to make additional WIBUKEY directories because of this feature).

The WIBUKEY directory contains many subdirectories. At this time, the *WibuKey* software can only be installed in these subdirectories. The driver files should be copied into a subdirectory of the Windows directory. With *WibuKey* 3.0, the driver files can also be copied directly in the WIBUKEY directory.

The setup program creates a log file in which all installed files are registered. The files with the name `WIBUKEY.LOG` will be contained in the WIBUKEY directory. The installation date and the GUID of the software developer are included as supplements to the log file. Following the supplement is a list of the installed files and possibly some additional information.

6.1 Problems and solutions

It is assumed that not every basis module supports a language module for every language.

Solution: A language which can be chosen will be displayed when more than one language module exists. A list of the languages that are installed should be displayed after installation is completed. English is the default language module in the other basis modules.

How should the new `README` file look? Until now all the files have been described-how can I check them? Until now we have only referred to the help files in Tools – but how does that work if a customer did not ever plan on installing the Help files? Where do I find the `README` text on the disk: since the text was not installed, the setup program cannot find it. So it must be on the last disk. A user is not found, however, because only the first disk is searched.

Solution: At the very beginning the additional notes text is copied into the WIBUKEY directory.

Does the Plug & Play still function with the new setup? What happens when the customer distributes the files elsewhere?

Because we have equipped the new drivers with the language module, Plug & Play is no longer playable. When someone inserts a PCMCIA-Card in the slot, then the system will search for and/or request the install disk be inserted. At present, the disks alone can install the driver. The operating system does not recognize multiple languages. That cannot be written in the `WIBUKEY.INF` data files because an error message will be displayed (even when just one of the files is supposedly missing) if this is tried.

Solution: We utilize `WIBUKEY.INF`-files that contain all languages. If a customer does not want to install all languages he must specifically take the files he does not want to install out. The ability to somehow call `SETUP.EXE` up out of an `INF` file has not been technically achieved. In order to test the available hard disk space, one must know how large the information to be installed is. These numbers must also be recorded in the `SETUP.INI`. If a customer falsely enters this information, bad things will happen.

Solution: There is no real way to measure hard disk space rather just a partial test to see if they might be space for all files (Option in `SETUP.INI`: `RegSpace=xxxx`).

6.2 Options in `SETUP.INI`

The configuration files `SETUP.INI` must be stored in the same (default) directory as `SETUP.EXE`/`SETUP32.EXE` or you must specify the path to these files through the start of setup.

The following options can be set:

Option	Description
`[General]`	Section with general problems
`AppId`	GUID of the software developer or rather the program that the *WibuKey* software installed.
`DisplayName`	Setup name that is shown in all dialogs. The standard is *WibuKey Installation.*
`SetupLang`	The specified language that should be used for the Installation dialog. The default value is the default where the installations program attempts to personally choose the language. The WIBU-2 letter shorthand for a particular language can also be specified. A supplement in the form of a list of the supported languages and their abbreviations is given.
`Languages`	lists the languages that are offered by *WibuKey* for installation. This option is more important then all others if the user interface is turned off.
`SourcePath`	specifies the particular path that data that needs to be installed must follow. The default value is the file from which `SETUP[32].EXE` is started.

Option	Description
Readme	specifies, if the runtime help file should be shown (1=yes, Standard or 0=no).
ForceOverwrite	specifies if existing entries in the WIBUKEY.INI should be automatically overwritten (0= do not overwrite, default, 1 = overwrite).
Gui	turns the user interface on (1, default) or off (0). If this option is enacted ForceOverwrite automatically is programmed to the value 1.
ErrorMessages	turns the output for error messages on (1,default) or off (0).
EvalIni	specifies, if entries in the also delivered file WIBUKEY.INI should be transferred into the Registry/WIBUKEY.INI. The entries that are to be transferred must be exactly specified. The same is true for the driver and WkNet entries.
[Files]	List of the files that need to be installed.
FileN	Information concerning the files that still need to be installed. This begins with the first name that is the original name of the file, then the number of the disk (on which the files have been saved) is listed, and lastly comes the name of the zipped file. Only files that are listed here can be installed.
[Tools]	Section with the tool installation options
NoToolSelect	prohibits the tool options window from being shown. In this situation all components that have files that are found on the disk will be installed.
WkSvW32Service	states whether WkSvW32 should be installed as service (1) or not (0). This option makes sense only when GUI=0 is set. Otherwise it can be entered in the Dialog if WkSvW32 should be installed as a service.
ShortCuts	states whether a menu item *WibuKey* should be established in the menu item program. If this

Option	Description
	option is chosen, the tools can be directly called up (1). If ShortCuts=0, this menu will not be established.
[Driver]	Section with the driver installation options.
Win32	states whether the WkWin32.dll should always be installed, regardless of the actual Operating system being used. Operating Ssystem (0=should be automatically installed, default, 1=always install).
WkDos.exe	states whether the WKDOS.EXE should be installed in Windows 95/98/NT (1, default) or not (0).
WkWin.dll	states whether WKWIN.DLL should be installed in Windows 95/98/NT (1, default) or not (0).

6.3 Commandline option

The commandline option should only be used for changes at the last minute, for instance to dynamically switch to another language. The main control instrument for the setup is the folder SETUP.INI.

Option	Description
INI-Folder	states the initialization data with all proper parameters.
/R[S][GUID]	starts the de-installation of the *WibuKey* files. The sub option S deactivates the user interface. GUID is the ID of the software developer or of the program, which installed the *WibuKey* software. This will be de-installed if no GUID is given and if it is only a *WibuKey* Version. If more than one version of *WibuKey* has been installed, a warning message will be displayed.
/?	displays the online help.

6.4 Adaptation of settings from WIBUKEY.INI

Customers who would like the opportunity to initially configure *WibuKey* can create a WIBUKEY.INI file that is sorted in the actual *WibuKey* configuration after the setup program files have been installed. Setting the WIBUKEY.INI

files in the windows directory is automatically done in Windows 3.x. In the 32-bit-operating system, the `WIBUKEY.INI` files are automatically set in the directory. With the Version 2.52, the setting for the server process is not yet automatically accomplished in the registry.

6.5 Return value of the setup

The setup records a status value in the registry `WIBUKEY.INI` so that an application, which has been called up by the *WibuKey* Runtime Kit setup, can display its status (if any shutdowns or errors that have been reported).

In the file `WIBUKEY.INI` of Windows 3.x there is a section [`Setup`] in which an entry **SetupStatus**. If using the 32-bit operating system this is located in the registry using the key `HKEY_LOCAL_MACHINE\Software\WIBU-SYSTEMS\WibuKey\Setup of the value SetupStatus`.

As long as the setup functions, the setup status has a value `-1`. If the setup is successfully ended, it has a value of `0`. In the case an error was recorded, the setup status is made up of the following values:

Error Code	Value	Description
`SE_ERROR_SUCCESS`	`0`	All drivers and programs were installed successfully.
`SE_ERROR_RUNNING`	`-1`	Setup is still running.
`SE_ERROR_NO_START_OTHER`	`-2`	`Setup.exe` (16-bit) can not start `Setup32.exe` (32bit) in a 32-bit windows system.
`SE_ERROR_INVALID_OPTION`	`-3`	An invalid commandline option was entered.
`SE_ERROR_NO_CONFIG`	`-4`	Setup was not found in the folder.
`SE_ERROR_INTERNAL`	`-5`	There was an internal user failure reported.
`SE_ERROR_NO_ADMIN_RIGHTS`	`-6`	No administrator right to install services or core drivers exists.
`SE_ERROR_ABORT`	`-7`	The program was stopped by an application.
`SE_ERROR_DDE_ATTACH`	`-8`	There is no current connection

Error Code	Value	Description
		to the program manager.
SE_ERROR_FOLDERCREATE	-9	The given file cannot be created.
SE_ERROR_ACCREGISTRY	-10	The registry cannot be accessed.
SE_ERROR_CREATEGROUP	-11	The program group cannot be created.
SE_ERROR_NODISKSPACE	-12	There is not an adequate amount of free space to save on the hard disk.
SE_ERROR_NOTEMPFOLDER	-13	A temporary directory cannot be created.
SE_ERROR_REBOOTREQUIRED	-14	You must reboot the system to fully complete installation.
SE_ERROR_NOSTARTTEMPSETUP	-400	Is only used internally.

6.6 Installation tips

WibuKey software components can only be installed if no application is trying to access them at the same time. It is strongly recommended that all other programs, especially all *WibuKey* protected applications and the control panel, are closed before the set up program is started. Otherwise it may happen that some files cannot be overwritten by a newer version. Normally this problem does only occur for `DLLs` and `CPLs` which have as dynamic libraries a special status on Windows. The Kernel driver such as `WibuKey.VXD` and `WibuKey.SYS` are not restricted by this control mechanism. Nevertheless problems may occur when the operating system detects differences between the installed files and the currently running processes.

In any case it is important to check the version of the files that are already installed. All *WibuKey* drivers are downwards compatible, i.e. older software will work with newer drivers. It cannot be guaranteed though that a new program runs together with old driver files.

Wibu-Systems strongly recommends working with the latest version of *WibuKey* drivers!

6.6.1 Special hints for Windows NT/2000/XP/Vista/Windows 7

On Windows NT/2000//XP/Vista/Windows 7 a special Kernel driver is required to access a *WibuBox*. This **`WibuKey.SYS`** kernel driver has to be registered with the operating system. A simply copying of the file to the Windows NT driver folder is not sufficient.

6.6.2 Special hints for Windows 95/98/Me

On Windows 95/98/Me a special kernel driver is required to access a *WibuBox*. In contrast to Windows NT this **`WibuKey.VXD`** kernel driver does not have to be registered with the operating system. For the installation please note that the current version of the Kernel driver can't be stopped. Once the VXD has been started it resides in the memory until the computer is restarted. The *WibuKey* Runtime Kit and the *Control Panel Applet* return a special error message that says that the system needs to be restarted.

6.6.3 Compatibility with previous software

In order to secure the compatibility with our previous software this chapter describes the settings for the *WibuKey* Runtime Kit which correspond to the settings of the previous software.

The previous driver install disk will be replaced completely by the new set up. Clients, which are used to employ the Runtime Kit may work with the new set up without changing their software. Almost all options of the previous driver Install disk are available also in the new set up.

Replacement table for options/settings for the new set up.

Old option	New option
`/V1`	new option `Readme`
`/V2`	no longer apply
`/W32`	`Win32=1` in `SETUP.INI`, section `[Driver]`
`/LDE`	`SetupLang=de` in `SETUP.INI`, section `[General]`
`/LUS`	`SetupLang=us` in `SETUP.INI`, section `[General]`
`/NOGUI`	`Gui=0` in `SETUP.INI`, paragraph `[General]`
`/NOMSG`	ErrorMessages=0 in `SETUP.INI`, section [General]
`Language`	Language=us/de in `SETUP.INI`, section [General]

Old option	New option
`Readme`	new option Readme

Part IX WibuKey Classic API overview

1 Introduction

The *WibuKey* Development Kit supports a variety of different operating systems and programming languages. The different variants, however, possess a common API (*Application Programming Interface*), independent of the programming language or operating system. Only a few minor deviations from this are to be noted, which are described in detail in the following.

The API is exclusively implemented via function calls or, alternatively, interrupt calls. No communication exists with the program to be protected via common variables or memory ranges. This makes the API extremely flexible.

The subsequent sections of this chapter deal with the different operating systems and programming language variants, together with their possible applications.

Another way to implement *WibuKey* functionality in software projects is the use of the WibuKey COM Control (ActiveX) which methods and properties base on the functionality of the 'Classic' *WibuKey API*.

1.1 Windows drivers

In order to protect Microsoft Windows applications and their macro programs, a driver is available in the form of a dynamic library (DLL). The code of this driver can be found in the library **`WKWIN32.DLL`** for 32-bit Windows (Win32). In dependence of the variant of the Windows operating system (Windows 9x and Windows NT/2000) these drivers call further drivers like the `WIBUKEY.SYS` kernel driver under Windows NT. Because such lower level driver are never directly accessed from an application and their API is not specified, they are not explained here but their usage is specified in the User's Guide of the Base Kit.

Due to the driver's complete multitasking capability, a number of programs may have simultaneous access to a connected *WibuBox* and therefore hold open a number of entries. Any conflicts concerning the access to the same *WibuBox* are ruled out via a temporary locking.

For the linking of `WKWIN32.DLL` to the program to be protected, the static library **`WKWIN32.LIB`** is supplied. It must be specified as library in the parameter list of the linker (for example LINK of Microsoft), and contains the import numbers and names of the **`WKWIN32.DLL`** library. The **`WKWIN32.LIB`** does not currently contain any code for the communication except the code for the determination of the FIRM and USER CODES of the application to be protected.

Due to the fact that the `WKWIN32.DLL` allows all API functions to be called directly via their names, these can also be called via macro languages. For Visual Basic an interface module in Basic source is available, named `WKVB.BAS`. This module can also be used for other macro languages that are based on Visual Basic for Applications.

For all programming environments that don't support the linking of static libraries maybe the *WibuKey* COM Control can be used. The COM Control is automatically installed and registered with the *WibuKey* Runtime Kit.

For Win32 currently no object file format standard has been established. That's why for different Win32 compilers different library files are available which must be renamed to `WKWIN32.LIB`.

1.2 *WibuKey* on the Apple Macintosh

WibuKey is available for Apple Macintosh with the *WibuBox/A* for the ADB bus and the *WibuBox/[R]U* for the USB bus supporting the CPU platform PowerPC. Drivers, programming tools and sample applications are on the *WibuKey* CD-ROM.

1.3 *WibuKey* on Linux

WibuKey is available for Linux with the *WibuBox/ST* for the serial port and the *WibuBox/U* for the USB bus. Drivers, programming tools and sample applications are on the *WibuKey* CD-ROM.

1.4 Access from other operating systems and non-PC hardware

To implement the addressing of the *WibuKey* hardware from special operating systems or on hardware that is not compatible with the PC industrial standard architecture, the **SRCDRV** can be used, a reduced *WibuKey* driver available in commented ANSI-C including a simple test program. The **WibuKey Adaptation Kit** which that is also located on the *WibuKey* CD-ROM describes additionally the addressing of the *WibuBox* on the serial bit transfer level and the command byte communication level in greater detail and is very useful to implement an individual *WibuBox* driver by using only the low level parts of the **SRCDRV** driver.

For UNIX systems or other non-IBM-compatible PC hardware the *WibuBox/ST* for the serial port can be used. This hardware variant uses an ASCII communication and can be accessed like a terminal. A detailed description and an Implementation Kit that supports the adaptation to the different UNIX derivates can be found on the *WibuKey* CD-ROM.

1.5 Access from C

To integrate *WibuKey* into C programs, a header file with the name **WK.H** is available, which directly supports (amongst others) the following C compilers:

- Microsoft C; version 6.0 and higher
- Borland C/C++, C++Builder; version 2.0 and higher
- All other C compiler which create a Microsoft compatible object format and which support language extensions like `_pascal`, `_far` and `_near`.

The *WibuKey* driver API description for the programming language C is to specify in the file **WK.H**. This must be specified in every source module via

```
#include <WK.H>
```

and copied in the directory in which the other included files (e.g. STDLIB.H) are located.

The program **`WKDM.C`** demonstrates how a simple test program written in C can be securely protected with *WibuKey*. The program can be build by the Microsoft C compiler (with version 6.0 or higher). For this purpose, the NMAKE file **`makefile.mc`** is available. For Borland C, a variant of this file exists, named **`makefile.bc`**.

An example for the compilation of a Windows program by Microsoft C is also available. This program is named **`WKCLOCK.C`**, and can be compiled using Microsoft Visual C++ version 1.0 or higher via the NMAKE-file **`makefile.mc`** for 16 bit Windows or Win32.

Please note the comments at the beginning of the `makefile.mc` and `makefile.bc` files for setting the parameters to build the different variants.

1.6 Access from Visual Basic

For Visual Basic, a separate *WibuKey* module was created, named **WKVB.BAS** that is stored in the Visual Basic sample directory of the *WibuKey* Development Kit. It implements the complete *WibuKey* API 2 access to the *WibuKey* driver `WKWIN.DLL` and the `WKWIN32.DLL` for Visual Basic Version 4.0 and higher.

1.7 Access from Borland Delphi

For the access on *WibuKey* from Borland Delphi there are different units (DCU) for the different versions of Delphi:

Name	Description

Name	Description
`WIBUKEY1.DCU`	unit for Borland Delphi 1.0 (16-bit)
`WIBUKEY2.DCU`	unit for Borland Delphi 2.0 (32-bit)
`WIBUKEY3.DCU`	unit for Borland Delphi 3.0 (32-bit)
`WIBUKEY4.DCU`	unit for Borland Delphi 4.0 (32-bit)
`WIBUKEY5.DCU`	unit for Borland Delphi 5.0 (32-bit)

The *WibuKey* unit is just an interface for the Windows libraries **`WKWIN.DLL`**, respectively **`WKWIN32.DLL`**.

The unit must be imported with the uses statement:

```
uses WIBUKEYx;
```

The *WibuKey* unit can be placed in any order within the unit-list.

1.8 Access from other programming languages

Other programming languages are supported either by static import-libraries and declaration modules for accessing the *WibuKey* driver model or by the *WibuKey* COM Control.

A complete overview over all supported programming languages can be found in the Development Kit samples directory of the *WibuKey* CD-ROM. Most of the sample projects contain additional information, respectively information about needed components in the project- or make-files.

2 Functions overview

This section deals with the most important functions of the *WibuKey API*. As already stated, these functions are extensively independent of the operating system and programming language. As such, the programming interface of the *WibuKey* needs only to be learned once. The transferal of protected software to other programming languages or operating systems is, with respect to the copy protection, relatively straightforward.

A detailed description of the API functions, structures and error messages can be found in the corresponding Online Help file.

The *WibuKey* hardware (the *WibuBox*) contains an ASIC, which encrypts or decrypts the incoming data from the PC byte by byte, and sends it back to the computer. The standard encryption and decryption are symmetric: When a sequence is encrypted and the result again encrypted using the same

parameters, then the original, non-encrypted version emerges. Hence, the second “encryption” was in fact a decryption.

The encryption sequence itself can be selected with various different parameters. These include:

- FIRM CODE
- USER CODE
- SELECTION CODE
- direct or indirect encryption

These terms are comprehensively explained in the User’s Guide that is supplied with the *WibuKey* Base Kit. Hence, the following assumes that the reader is familiar with them. The FIRM and USER CODE are permanently stored in the *WibuBox*, as programmed by the software developer with the aid of the **WKCRYPT** program. The SELECTION CODE specifies one of 4 billion different encryption sequences (32 Bit), subsequent to the selection of a FIRM and USER CODE pair from the *WibuBox*. The choice of a direct or indirect encryption decides whether the bytes of a data sequence are to be exclusively encrypted within the *WibuBox*, or whether a small encryption sequence is to be extracted from the *WibuBox* (1 up to 20 bytes) for encryption within the PC memory without any further access to the *WibuBox*. The direct encryption is safer for anti-copy protection, since this completely rests on the confidential *WibuBox* hardware. In comparison, the indirect encryption is much faster, as the (slower) *WibuBox* hardware is not drawn on. Furthermore the indirection encryption uses different algorithms for different purposes, for example the FEAL algorithm for high security data protection or the permutation for moving existing data objects without modifying them.

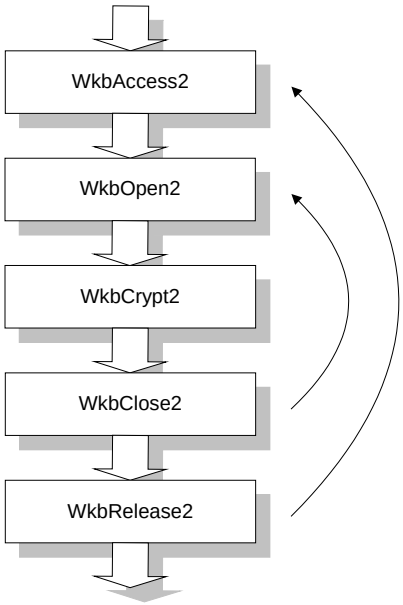


Figure 52: Calling scheme of the *WibuKey* base functions

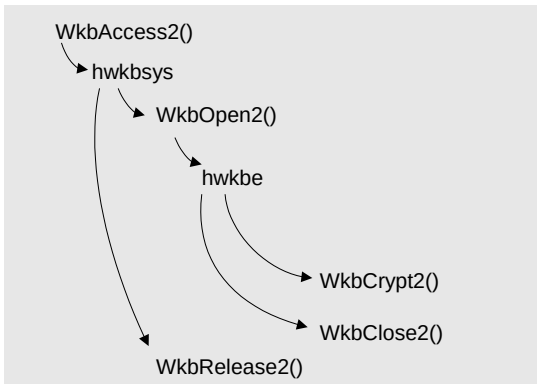


Figure 53: Addressing of operation groups via handles

The access to the *WibuBox* hardware is accomplished similar to a file access under DOS or UNIX (Figure 1):

- A specific subsystem at the local computer system or within a network is accessed via the function **WkbAccess2**. Via a network subsystem, a remote *WibuBox* within a PC network can be addressed by the **`WkNet`** (file based) or the **`WkLAN`** (protocol based) transport layer. The **WkbAccess2**

function returns a handle (see figure 2). A handle is a non-interpretable integer number that is specified in all subsequent function calls that refer to this subsystem.

- When a network subsystem is accessed via **WkNet** or **WkLAN**, **WkbAccess2** allocates also a user entry in the user limitation table. This avoids that more programs can access a network *WibuBox* than users are permitted within the network.

```
WKBACCESS wkbacc;
ULONG flAccess;
BYTE abCryptSeq[32];

// specify selected Firm and User Code
wkbacc.lFirmCode = pwkdata->ulFirmCode;
wkbacc.lUserCode = pwkdata->ulUserCode;
// version 1.32 of the subsystem is needed
wkbacc.bVersionHigh = 1;
wkbacc.bVersionLow = 32;
wkbacc.usRefreshTime = 10; // check every 10 seconds
// asynchronous mode of the API functions is default
wkbacc.flStdCtrl = WKB_ASYNCHRONOUS;
// access via the application name and WIBUKEY.INI
wkbacc.pszWkNetFile = NULL;
wkbacc.pszAppName = "DrvTest"; // application name
wkbacc.idNetUser = WKB_NETUSER_FIND; // search for user index
wkbacc.flCtrl = 0;            // set WKB_ACC_AUTOCANCEL
wkbacc.pszServerName = NULL;  // only needed for WkLAN
wkbacc.usReserved1 = 0; // reserved for future use
// encryption constants of the WkNet access
abCryptSeq[0] = 8;      abCryptSeq[16] = 114;
...
abCryptSeq[31] = 132;
wkbacc.pbEncryptSeq = abCryptSeq;

//// access on the specified subsystem
// here: access on local and WkNet subsystem
flAccess = WKB_ACC_LOCAL | WKB_ACC_WKNET | WKB_ACC_CREATE;
hwkbsys = WkbAccess2(flAccess, &wkbacc);
```

- For a simple addressing of a *WibuBox* at the local system, the calling of **WkbAccess2** can be omitted and the constant handle HWKB_LOCAL is used to address the default local subsystem.

- After having an access to a local or network subsystem, the desired FIRM and USER CODE pair must be searched into the *WibuBoxes* which can be physically addressed at this subsystem. This search is realized via the function **WkbOpen2**, which interrogates one, a number, or all connected

WibuBoxes for the specified code pair. Upon finding one, the function then "opens" the entry and returns a handle again which corresponds with the opened entry in the *WibuBox*. All subsequent function calls which uses the FIRM/USER CODE pair for encryption etc. uses this handle (hwkbe in Figure 2). An opened *WibuBox* entry is not automatically “used”. Therefore it is possible to open the same *WibuBox* entry simultaneously from different processes in a multitasking environment like Windows.

```
// scan for and open WibuBox entry in the
// specified WibuKey subsystem
// control flags: standard function control flags,
// use version 2, search entry
flCtrl = WKB_STDCTRL|WKB_OPEN_FIND|WKB_VERSION2;
hwkbe = WkbOpen2(
  hwkbsys,           // handle of the accessed subsystem
  flCtrl,            // control flags
  NULL,              // not used
  WKB_ACCESS_CODE,   // use Firm Code of subsystem
  WKB_ACCESS_CODE,   // use User Code of subsystem
  NULL               // no optional data
);
```

The encryption method is specified in the **WKBAREA** structure. An encryption sequence is selected for an open *WibuBox* entry via the specification of the desired SELECTION CODE (4 billion variants). The sequence is reset. All following **WkbCrypt2** calls with the same *WibuBox* entry handle refer to this sequence.
Using the *AREA* encryption there is no need to select and unselect the encryption algorithm by **WkbSelect2** and **WkbUnSelect2**. This encryption method is faster than the traditional way and is much efficient in network subsystems.

```
BOOL fValid;
WKBAREA wkbar;

wkbar.flCtrl = WKB_AREA_SELECT;
wkbar.flSelCtrl = WKB_STDCTRL|WKB_SEL_DIRECT;  //direct
encryptiom
wkbar.ulSelectCode = ulSC;
wkbar.pvCtrl = NULL

// encrypt of data
fValid = WkbCrypt2(
  pwkdata->hwkbe,             // handle to opened entry
  WKB_STDCTRL|WKB_CRYPT_CTRL, // flags for crypt & copy
  szCrypted,                  // destination buffer
  &wkbar,                     // source buffer
  cbData,                     // bytes to encrypt
  &cbCrypt                    // bytes encrypted by WkCrypt2
```

```
);
```

- When a *WibuBox* entry is no longer used, it must be closed. This is realized by the function **WkbClose2**, which must be called for every opened entry to ensure that within a multitasking driver (for example on Windows), no longer used open entries allocate permanently memory.

```
//// close opened entry
fValid = WkbClose2(hwkbe);
```

- The function **WkbListPort2** determines which *WibuBoxes* are connected to a specified *WibuKey* port of a used subsystem (or HWKB_LOCAL) by returning a list of the serial numbers of all *WibuBoxes* found at this port.

```
PORTLIST prtlist;

//// read serial numbers of attached WibuBoxes
fValid = WkbListPort2(
  hwkbsys,          // handle of accessed subsystem
  WKB_PORTNUMBER | WKB_STDCTRL,  // standard control flags and
                                    flag for specifying ports
name
  "LPT1",           // port is LPT1
  &prtlist,         // points to buffer for WibuBox serial
numbers
  sizeof(prtlist)); // length of buffer
```

- The contents of a single *WibuBox* with the determined serial number can be read via the function **WkbListBox2**. This function returns a list of all stored FIRM and USER CODE or USER DATA entries including bundles ADDED DATA or LIMIT COUNTER entries. This function is useful to analyze the contents of the *WibuBox*; found FIRM and USER CODES can be used in subsequent **WkbOpen2** calls.

```
WKBBOXDATA wkbbxd;

//// read contents of the specified WibuBox
fValid = WkbListBox2(
  HWKB_LOCAL,       // handle of accessed subsystem
  WKB_PORTNUMBER | WKB_STDCTRL, // standard control flags and
                                   flag for specifying ports
name
  "LPT1",           // port is LPT1
  prtlist.wkbbxl.awkbsr,  // points to buffer for WibuBox serial
numbers
  &wkbbxd,          // points to buffer for WibuBox content
  sizeof(wkbbxd)); // length of buffer
```

- When a *WibuKey* subsystem is no longer used its access must be released via the function **WkbRelease2**. This function closes in a network

the subsystem connection and frees the corresponding allocation within the user limitation table.

```
//// release accessed subsystem
fValid = WkbRelease2(pwkdata->hwkbsys);
```

- Every program encrypted via **WKCRYPT** has a defined FIRM and USER CODE. This is automatically stored at a specific location in the program code by the **WKCRYPT** option /**CX.** Both FIRM and USER CODE can be determined by the program itself after starting by calling the functions **WkbGetFC** and **WkbGetUC**.

```
//// get Firm and User Code from program code
INT idError;
idError = WkbGetFC(ulFirmCode, 0);
...
idError = WkbGetUC(ulUserCode, 0);
```

This section only yields an overview of the most important functions of the *WibuKey Classic API*. Their individual, detailed description may be found in the API online help file.

3 Specification of *WibuKey* ports

The *WibuKey* copy protection system hardware, the *WibuBox*, is available for various ports. The communication with the individual ports is the responsibility of the *WibuKey* driver. The programmer need not (and indeed, normally cannot) intervene.

The functions of all ports are, from the point of view of the software to be protected, the same. One need only specify the number of the port, for which purpose, the following values may be used as **port numbers**.

Port Number	Port Name / Description
0	LPT1
1	LPT2
2	LPT3
3	LPT4 (only available when set in the environment)
4..6	(reserved for LPT5 to LPT7)
7	mask value: scan from LPT1 to LPT7
8	COM1
9	COM2

Port Number	Port Name / Description
10	COM3
11	COM4
12..14	reserved for COM5 to COM7
15	mask value: scan from COM1 to COM7
16	SRL1
17	SRL2
18	SRL3
19	SRL4
20..31	reserved for Wibu-Systems extensions
32	PcCard1 (PCMCIA slot 1)
33	PcCard2 (PCMCIA slot 2)
34	PcCard3 (PCMCIA slot 3)
35	PcCard4 (PCMCIA slot 4)
36	PcCard5 (PCMCIA slot 5)
37	PcCard6 (PCMCIA slot 6)
38	(currently not used)
39	mask value: scan from PcCard1 to PcCard6
40	ADB bus (Apple Macintosh)
41…48	reserved for Wibu-Systems extensions
48	USB
49..63	reserved for Wibu-Systems extensions
64..127	reserved for OEM-specific extensions
128..254	reserved for future use
255	mask value: scan through all ports, beginning with LPT1.

The WibuKey API 2 supports also the name specification of ports, for example the string `"COM3"` instead of the number 10.

Mask values are generally all numbers in which the lower 3 bits are all set to 1. With such a mask value, a number of ports are scanned, beginning with the (*port number* - 7) and ending with the (*port number* - 1). The exception is the value 255, which causes all ports to be scanned beginning with channel 0 (LPT1). The name specification supports also the mask option by replacing the trailing number of a port specification by a star (*), for example `"PcCard*"` searches the same ports like the number 39.

WibuKey can also be adapted within the scope of an OEM extension to special ports. For example, the ASIC of a *WibuBox* can be procured from Wibu-Systems and be integrated into uncommon port cards. In this case, an adaptation or extension of the *WibuKey* driver may, under some circumstances, be necessary. For such instances, Wibu-Systems offers the **Adaptation Kit** with which the developer's own hardware driver can be developed and integrated in the C source of the *WibuKey* driver **SRCDRV**. For this purpose, the port numbers from 64 up to 127 may be used.

4 WkLAN network protection

WkLAN is a protocol-based communication (TCP/IP), the data is transmitted synchronously over the network between the clients and the **WkLAN** servers. The current implementation works on Windows 9x/Me, Windows NT/2000, Mac OS and Linux. It is not planned to support the NetBEUI protocol.

A description of the basic features of **WkLAN** can be found in the User's Guide. With **WkLAN** (nearly) every function, which can be used for a local access on a *WibuBox*, can be used over the network to access a *WibuBox* via the server process. **WkLAN** is completely transparent to the local use of a *WibuBox*. It is very easy to search for a local *WibuBox* first and, if none was found, continue the search in the network.

All API functions can be redirected in such a way. A *WibuBox* can be opened, read and re-programmed over the network or data can be encrypted via **WkLAN** too.

5 Huge License Management

The *Huge License Management* (HLM) is designed to avoid the limitation of the network protection of a maximum of 5 different protected program modules. Therefore the license information is stored in an encrypted file.

HLM support both, the **WkNet** and the **WkLAN** network subsystem. Detailed information can be found in the User's Guide.

6 Working with the WibuKey Classic API

WibuKey is available for different operating systems with a similar programming interface (API). But when using this interface, some features must be noted in dependence of the used operating system. Such features are explained in the following chapters.

6.1 General information

6.1.1 *Classic API* implementation

The *WibuKey Base API* is realized with a number of drivers, available as libraries or programs that are pre-translated, as part of the *WibuKey* Development Kit. The source code of these drivers is normally not needed. Moreover, Wibu-Systems reserves the right to change or extend the implementation of the drivers for the adaptation to future systems. The functions and data structures of the *WibuKey API* will remain, however, in their present form and operation and may only be expanded with new features or functions.

The functions and data structures of the *WibuKey Base API* were originally developed for the programming language C, but may be converted to nearly all other programming languages.

In detail, the following drivers and libraries are currently available, which realize the *WibuKey Classic API*:

Driver Name	Description
WKWIN.LIB	is the import library of WKWIN.DLL, the actual *WibuKey* driver for 16 bit Windows. This library is used for the development of Windows modules (applications or dynamic libraries), when the product is to be combined with a classical linker (e.g. **LINK** or Microsoft). In addition to the import descriptions for the WKWIN.DLL functions (such as WkbOpen2), WKWIN.LIB also contains the data area for the storing of the FIRM and USER CODE which can be inserted into the completed Windows module by **CRYPT** via the option **/CX**. Moreover, it contains the program code of the API functions **WkbGetFC** and **WkbGetUC**.
WKWIN32.LIB	is the import library of WKWIN32.DLL, the actual *WibuKey* driver for 32 bit Windows (Win32). Except of the different data width and the different object format this file is identical with WKWIN.LIB.

Driver Name	Description
WKWIN.DLL	is the *WibuKey* driver for 16 bit Windows from version 3.1 and higher. This contains all WibuKey Base API 1 and 2 functions except **WkbGetFC** and **WkbGetUC**, since these determine the module-specific FIRM and USER CODES which cannot be part of the dynamic library. This driver is part of the *WibuKey* Base Kit.
WKWIN32.DLL	is the *WibuKey* driver for 32 bit Windows (Win32) for Windows 3.1 (with Win32s), Windows 95 and Windows NT. It contains all API 2 functions except **WkbGetFC** and **WkbGetUC**, since these determine the module-specific FIRM and USER CODES that cannot be part of the dynamic library. This driver is part of the *WibuKey* Base Kit.
WIBUKEYx.DCU	is the Borland Delphi unit for Borland Delphi version 1.0 or higher for the access to the *WibuKey* driver `WKWIN[32].DLL` from Windows application. Similar to the `WKWIN.LIB` library, `WIBUKEYx.DCU` contains, in addition to the import functions of `WKWIN[32].DLL` (such as **WkbOpen2**), also the data range for the filing of the FIRM and USER CODE which can be entered into the completed Windows module by **WKCRYPT** via the option /CX. Moreover it contains the program code of the API functions **WkbGetFC** and **WkbGetUC**.

6.1.2 Classic API function calls

All 16-bit drivers use the *Pascal* convention for the parameter calling. Here, the parameters are put on the stack from left to right and the allocated stack area for the parameters is cleared by the called function. All functions are called via *far* calls and uses *far* pointers.

32 bit drivers use the Win32 standard system calling convention which is named with the keywords `_stdcall` or `APIENTRY32`.

6.1.3 Base API function notation

Wibu-Systems uses the so-called *Hungarian notation* for the specification of the type of variables and parameters of its functions. This has been named after its Hungarian inventor and Microsoft programmer *Charles Simonyi*, and is used, amongst others, by Microsoft, IBM and Apple in similar fashions. With this notation, every variable and parameter name begins with lower-case letters, which more or less unambiguously defines the kind and type of variable, hence

reducing the documentation effort. The actual variable name begins with a capital and only describes its meaning, not the type. Wibu-Systems uses the following indices for the notation of its API variables and functions:

Index	Description
p	pointer
b	8-bit value (BYTE)
u	unsigned integer value (UINT)
n	signed integer value (INT)
s	short integer (in general 16 bit)
l	long integer (in general 32 bit)
v	no specified type (void)
i	index within a loop or similar
c	count and quantity value
f	flag(s)
h	handle

6.2 The general function calling format

Many functions of the *Classic API* for addressing the *WibuBox* have a similar parameter format. A typical example of a such a function is **WkbListBox2**, the function to list the contents of a specific *WibuBox*:

```
INT WkbListBox2(HWKBSYSTEM hwkbsys, ULONG flCtrl, const
TCHAR FAR *pszPort, const WKBSERIAL FAR *pwkbsr, VOID FAR
*pvDest, UINT cbDest)
```

The first parameter *hwkbsys* is a handle, which specifies the used subsystem. The handle constant `HWKB_LOCAL` can be used to address the local standard subsystem. The next parameter *flCtrl* contains general and function specific options that specify the exact operation of the function. The general options specify for example, if the function is executed synchronous or asynchronously (see next chapter) and are explained in greater detail in the API online help. Function specific options are for **WkbListBox2** for example `WKB_LIST_DATA` for listing the contents of the entries of the *WibuBox* or `WKB_LIST_CONFIG` for reading the configuration data of the *WibuBox*. The list of function specific options may be increased in future.

A frequently used function specific option is WKB_PORTNUMBER: If this option is set in *flCtrl*, the parameter *pszPort* doesn't point to a real string but contains a number containing in the lower 16 bits a port number, for example 1 for LPT2 and in the higher 16 bits a 0 value. Such a number can be created simply by the **MAKEINTRESOURCE** macro:

```
WkbListBox2(HWKB_LOCAL, WKB_LIST_DATA|WKB_PORTNUMBER,
  MAKEINTRESOURCE(1), ...);
```

If the WKB_PORTNUMBER option is not specified, *pszPort* points to a real string, containing a port name, for example "LPT2":

```
WkbListBox2(HWKB_LOCAL, WKB_LIST_DATA,
  "LPT2", ...);
```

In dependence of the specific operation, specified by the options in *flCtrl*, the meaning of the other parameters may be differing. The parameter *pvDest* for example points to a WKBBOXDATA structure and the parameter *cbDest* contains the length value of sizeof(WKBBOXDATA) if the WKB_LIST_DATA option is specified. On the other hand, if the WKB_LIST_CONFIG is used, *pvDest* points to a WKBBOXCONFIG structure and *cbDest* contains the value of sizeof(WKBBOXCONFIG). Other parameters are independent of the function operation, pwkbsr for example points always to a serial number, stored in a WKBSERIAL structure.

6.3 Synchronous and asynchronous function modes

If not specified otherwise, all operations of the *WibuKey API* functions are executed in the **synchronous** mode: The function executes an operation, for example, it selects a *WibuBox* entry, and returns to the caller not before this operation is completed or is aborted if an error occurred. In both cases, the function returns the state of this operation. The older functions of API version 1 returns a value of 0 if the operation was successful and an error code if an error occurred. The functions of the *Classic API* return no direct error code; they return a Boolean value, a handle, a size value or other information. Normally a value of 0 or NULL instead of a useful value indicates an error; the error code can be returned via the **WkbGetLastError** function.

In the **asynchronous** mode, a function returns immediately after it has started its operation without waiting the completion or aborting of this operation. Instead of a return value the function returns a specific **action code** which corresponds with this operation. The caller can continue its own operations after this return; the results of the API function operations are determined automatically in the background. The caller can check the termination or the abort of an asynchronous operation by calling the **WkbIsComplete** function with the desired action code in the argument. If the operation is completed, this function returns TRUE and stores the return value of the completed asynchronous function into the buffer that is specified by the second parameter of

WkbIsComplete. If the operation returns FALSE, the specified action is undefined or still under execution which results in an error code which can be determined by a subsequent call of the **WkbGetLastError** function. Normally this is the error code `WKB_ERROR_IN_ACTION`, when the tested action is still in execution or the error code `WKB_ERROR_UNDEFINED_ACTION` when the specified action code is not valid.

If the original function returns a value that corresponds with a failed operation of this function, a subsequent call of **WkbGetLastError** after **WkbIsComplete** returns this error code.

An action code is only valid between the return from an asynchronously called function and up to the call of the **WkbIsComplete** function that returned TRUE. A subsequent or later call of **WkbIsComplete** with the same action code results in the error code `WKB_ERROR_UNDEFINED_ACTION`.

A function that is called asynchronously doesn't return always an asynchronous identification. Instead it is possible, that also an error state in the same matter like a synchronous call. Such typical error states are wrong parameters or if no new action code is available. If an asynchronously started function terminates correctly, always an action code is returned and the return value can be determined immediately or later via the calling the **WkbIsComplete** function.

To support the asynchronous starting of multiple functions at the same time, the action codes are changed after an assignment to an asynchronous function. Action codes are always negative (the highest bit is set to 1), this permits to differ such values from error codes, Boolean values or size specifications.

If too many functions are called simultaneously in the asynchronous mode, the error code `WKB_ERROR_TOO_MANY_ACTIONS` is returned.

The asynchronous mode can be activated at two different positions:

- If a new subsystem is created via the **WkbAccess2** function with the command option `WKB_ACC_CREATE`, the default behavior of the synchronous/asynchronous mode is selected in the *flStdCtrl* member of the **WKBACCESS** structure. Specifying the `WKB_ASYNCHRONOUS` flag activates the asynchronous mode, the synchronous mode is used if this flag is not specified or the default `WKB_SYCHRONOUS` flag is specified.
- This default mode is used for **all** API function calls in which the specified subsystem is used. Such functions have a *hwkbsys* or *hwkbe* handle in their parameter list which corresponds with this subsystem directly or indirectly.
- To change the default mode of a single call of a subsystem specific function, the `WKB_ASYNCHRONOUS` or `WKB_SYCHRONOUS` flags must be used in the *flCtrl* parameter of this function in combination with the `WKB_EXCLMODE` flag (see standard function control flags in the API online-

help). When the standard local subsystem with the HWKB_LOCAL handle value is used, the WKB_EXCLMODE flag is optional because this subsystem always uses the WKB_SYNCHRONOUS mode as default.

All function which are not subsystem or open-entry specific and don't have a *hwkbsys* or *hwkbe* handle parameter and no *flCtrl* control parameter, are **always** called in synchronous mode.

The asynchronous mode normally only makes sense for **WkNet**. All function calls using **WkLAN** return more or less immediately, so the standard mode for **WkLAN** should be the synchronous mode.

7 API general notes

There are a number of methods in which the *WibuKey* can be explicitly incorporated into the program to be protected. In order to attain an effective protection, the software developer should invest a few thoughts to this aspect. *WibuKey* can be flexibly employed. Codes or data can be encrypted prior to issue and at call, decrypted for the user. The program itself can encrypt and decrypt sections of itself as it runs so that it is never within the memory in a totally decrypted state. Finally, externally filed data can also be drawn on for the encryption.

8 *WibuBox* checking

Checking, as interrogation concerning whether a certain *WibuBox* with a specific Firm and User Code is connected to the computer can be accomplished via the function **WkbOpen2**. The parameters are the desired FIRM and USER CODES. When the return value is 0, the entry has not been found; otherwise the function is to be terminated by calling the function **WkbClose2**.

The interrogation of the *WibuBox* alone is not very secure and can relatively rapidly be lifted by hackers. This particularly holds for multitasking Windows variants, OS/2 and also DOS. The interrogation alone can, however, be used to issue a warning or abort the program. More complex protection mechanisms should above all then prohibit the program from being used further. These are then activated without warning, as this condition only then arises when a hacker is manipulating the program.

9 Software demo version

When the *WibuKey* protection is applied in a fashion that the program can still run without the corresponding *WibuBox*, but in a confined manner, the program may be used as a demo version. Only the user in possession of the corresponding *WibuBox* will be able to use the program properly. The program can, nevertheless, be copied and handled further. Those with a copy (without the *WibuBox*) can use the program in a restricted manner, to learn the

advantages and characteristics, without being able to exploit its relevant potential. Only after obtaining the license (*WibuBox* with documentation) can such a user put the program to use. Many producers allow their products to circulate in this manner. The necessity for the development of special demo programs is not necessary.

For many demo versions the integration of the protection is confined to sections of the software that are not able to run without the *WibuBox*. This can include, for example:

- saving of text for a text editor
- installation of fields for a compiler
- annual audit for financial accounting programs.

Such program sections should be encrypted by the *WibuKey*. It must be ensured that these sections cannot be called with the aid of a manipulation i. e. by the simple interrogation of the *WibuBox*.

Demo versions are often conceived in a more complex fashion. Examples of this are as follows:

- Installation of just small data bases in a data base system.
- Continual output of the text “Demo version” for a CAD program
- Translation of just brief procedures for a compiler.

Such demo versions may be devised by encrypting program data. For this purpose, the data that distinguishes the demo version (e.g. the output control for the text “Demo Version”) and cater for its output is normally decrypted via the *WibuBox*. The data is then compared at a concealed location to the original data, and in the case of discrepancies, the demo version is activated.

10 Encryption of data files

Due to the fact that the *WibuKey* can encrypt data with practically any sequence length means that the contents of files for data bases or text processors can be securely protected. The data is then encrypted via the corresponding *WibuBox* and filed. It can then not even be analyzed by file analyzing programs such as PC tools or Norton Utilities. Only after this data has been called or processed by the respective program may the data be decrypted by the correct *WibuBox* and then viewed. Since the *WibuBox* can be easily removed from the printer port and secured against third party access (e. g. in a safe) means that the data can be safely protected than with the use of passwords which the individual users must learn and which are often noted in accessible places.

Part X The WibuKey Interactive API Guide

The *WibuKey Interactive API Guide* (WKIAG) is a very helpful tool to get acquainted with the *WibuKey Classic API*. The most important functions can be tried out interactively by specifying the necessary parameters and evaluating the result of the function call.

Because of the various combinations of parameters and function calls the *WibuKey Interactive API Guide* is a good start to learn all about the powerful features of *WibuKey*.

1 General idea of the Interactive Guide

With the interactive guide the most important *WibuKey API* functions can be combined to a row of function calls to learn about the effects of the different control parameters.

The following *WibuKey Classic API* functions are supported by the *WKIAG*:

Base WibuBox Open and Encryption

- `WkbOpen2`
- `WkbSelect2`
- `WkbCrypt2`
- `WkbUnSelect2`
- `WkbClose2`

Subsystem (Network) Management

- `WkbAccess2`
- `WkbRelease2`
- `WkbEnumServers2`

List Port and WibuBox Contents

- `WkbListBox2`
- `WkbListPort2`
- `WkbEnumPorts2`

Detailed information about the *WibuKey API* functions and the corresponding structures can be found in the online help file.

The main window of WKIAG shows an overview over the supported API functions and offers different option fields where to function specific flags and parameters can be specified.

First an API function must be selected by choosing the tab of the function. On the function tab all necessary parameters and flags must be set. The button *Insert* at the *WibuKey Functions* field at left inserts the selected API function with its parameters in the row of function calls. The function list can be modified by the buttons *Delete*, *Modify*, *Up* and *Down*. Finally the button *Execute* starts the execution of the function list beginning with the topmost API function.

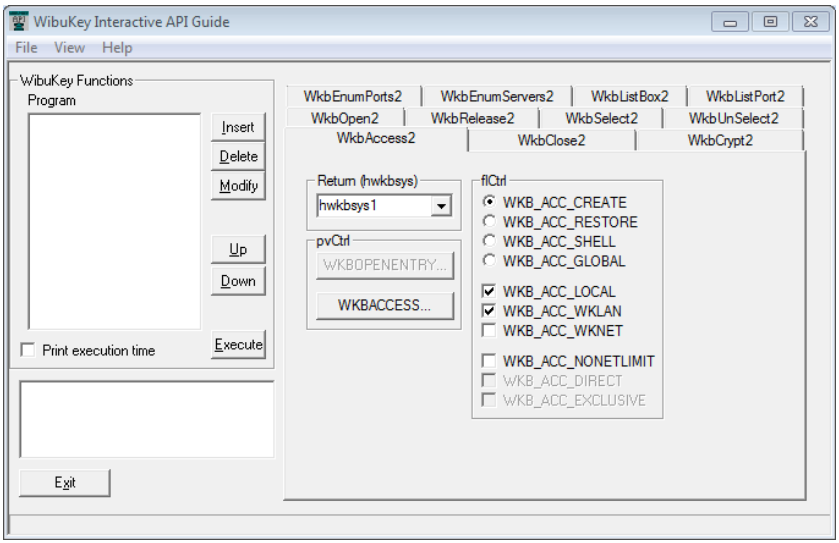


Figure 54: *WibuKey Interactive API Guide* (WKIAG) - Start

All output, e.g. the insertion or the modification of function calls, the function calls itself with all parameters and the results of the function calls are displayed in the *Output* windows.

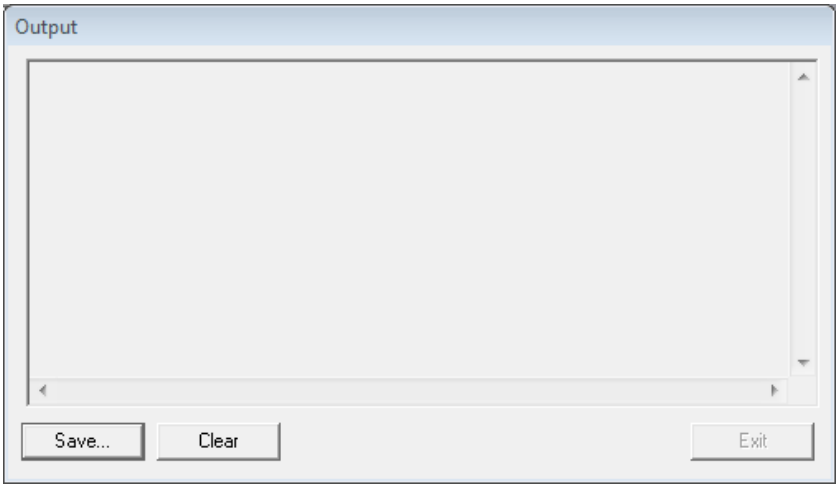


Figure 55: *WibuKey Interactive API Guide* (WKIAG) - Output

If a parameter of a function contains a structure the elements of this structure can be specified in a separate window.

2 Learning step-by-step

The following step-by-step lesson demonstrates what functions of the *Classic API* must be used for an encryption of data via the local and the **`WkLAN`** network subsystem.

Please note that this lesson demonstrates the ‘traditional’ selection and un-selection of the encryption algorithm by the functions **WkbSelect2** and **WkbUnSelect2**, new *WibuKey* implementations always should use the 'AREA' encryption that does not need these functions. Here all information is kept in the **WKBAREA** structure.

At first the (network) subsystem that contains the FIRM / USER CODE combination 10:13 must be searched. Here only the local and **`WkLAN`** subsystem are specified in the `flCtrl` flags. After a successful execution of `WkbAccess2` the subsystem handle is returned and stored in the `hwkbsys1` variable.

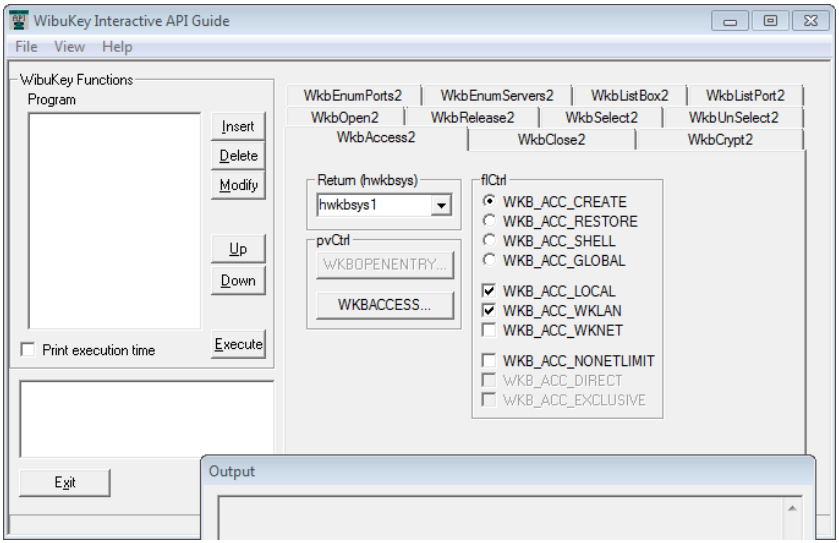


Figure 56: *WibuKey Interactive API Guide* (WKIAG) - Accessing a subsystem with WkbAccess2

In the **WKBACCESS** structure additional options, e.g. the FIRM / USER CODE combination can be specified. All options and flags a described in the online help file.

After specifying all parameters the **WkbAccess2** function call is inserted into the call list by the Insert button.

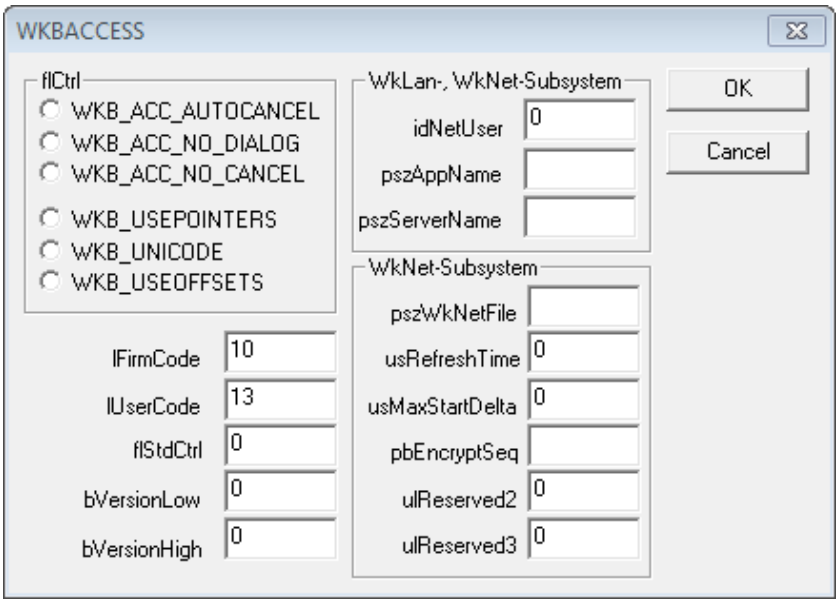


Figure 57: *WibuKey Interactive API Guide* (WKIAG) - Elements of the WKBACCESS structure

After the successful access of the subsystem the *WibuBox* entry must be opened. The FIRM / USER CODE and the version information must be passed to the **WkbOpen2** parameters. **WKB_OPEN_FIND** specifies that the accessed subsystem is to be scanned for a fitting entry.

The handle on the opened entry is returned in the `hwkbe0` variable, the information about the accessed subsystem is passed in the `hwkbsys1` variable that was returned by **WkbAccess2**.

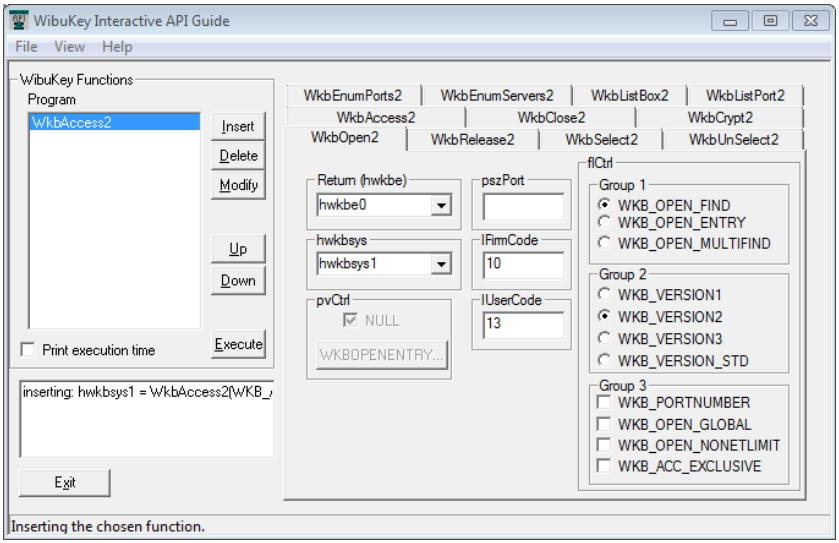


Figure 58: *WibuKey Interactive API Guide* (WKIAG) - Search for a *WibuBox* entry with WkbOpen2

A subsequent call of **WkbSelect2** sets the encryption parameters; here the direct encryption is selected in the `flCtrl` flags.

WkbSelect2 uses the handle of the opened entry returned by **WkbOpen2**. All subsequent function calls on this subsystem use this handle to identify the selected entry in the accessed subsystem.

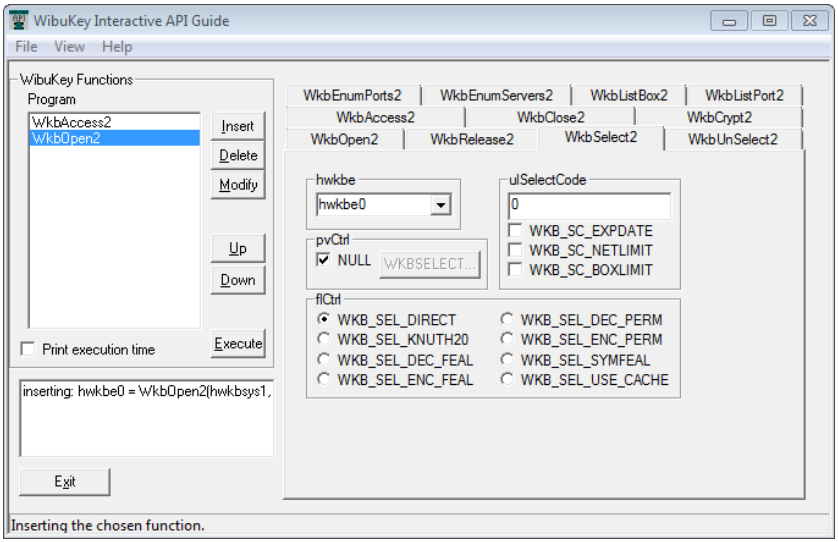


Figure 59: *WibuKey Interactive API Guide* (WKIAG) - The algorithm is selected with WkbSelect2

The encryption data must be specified in the *pvDest* paramter of the next function, the **WkbCrypt2** en-/decryption function. No destination buffer is specified in `pvCtrl` field, because the **WKB_CRYPT_CTRL** option results in overwriting the original data in the `pvDest` buffer.

If the original data should be kept the **WKB_CRYPT_COPY** option must be set and `pvCtrl` must specify a destination buffer.

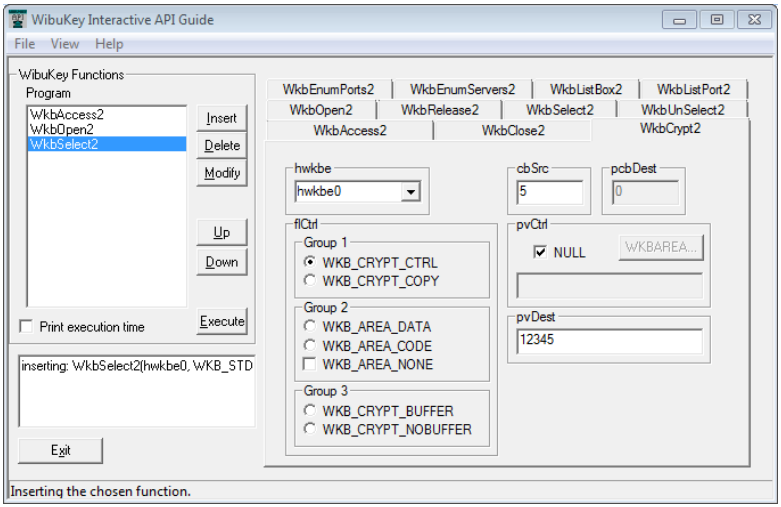


Figure 60: *WibuKey Interactive API Guide* (WKIAG) - WkbCrypt2 encrypts the specified data ‘12345’

After the encryption the algorithm must be set back to its initial state. This is done by the **WkbUnSelect** function, without that a subsequent encryption of the first encryption result won’t result in the original data.

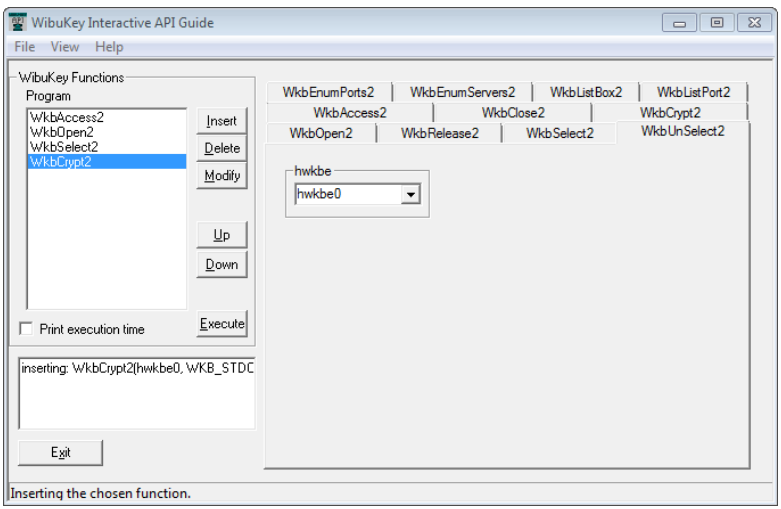


Figure 61: *WibuKey Interactive API Guide* (WKIAG) - Reset of the encryption algorithm with WkbUnSelect2

At last the calls of **WkbClose2** and **WkbRelease2** closes the opened entry and the releases the accessed subsystem.

After these two function calls the *WibuKey* driver is completely set back to its initial state and all driver internal memory is freed.

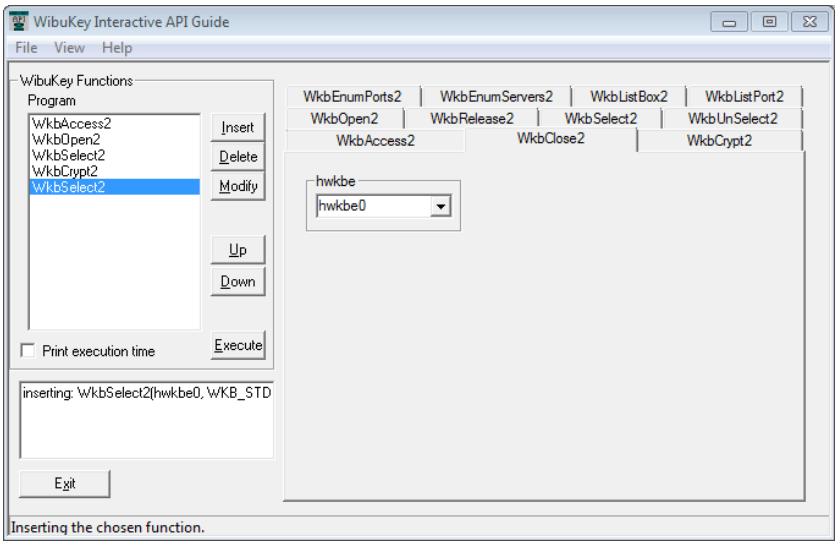


Figure 62: *WibuKey Interactive API Guide* (WKIAG) - At last WkbClose2 closes the opened entry

The following figure shows the output monitor after the insertion of all *WibuKey* API function calls. All function calls are reported with all parameters as they are used in the real execution of this function list.

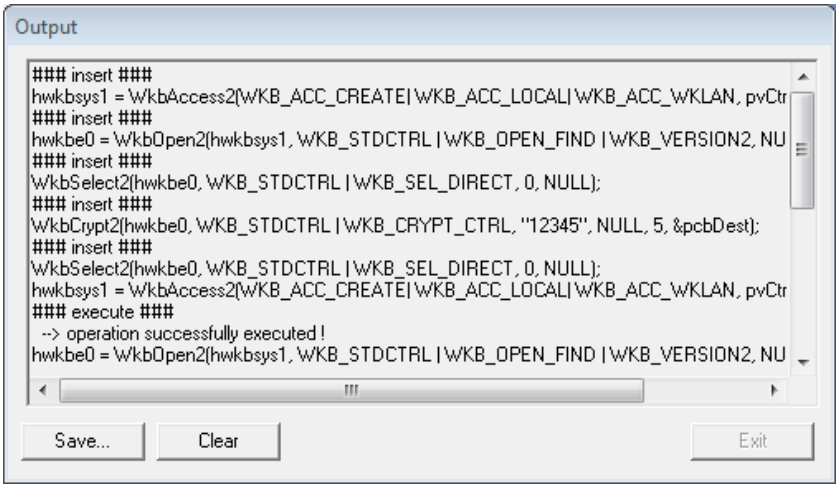


Figure 63: *WibuKey Interactive API Guide* (WKIAG) - All functions were inserted in the calling list

Every function call is reported in the output monitor, the state of the execution and the result, e.g. the encrypted date after the call of **WkbCrypt2**, is reported too.

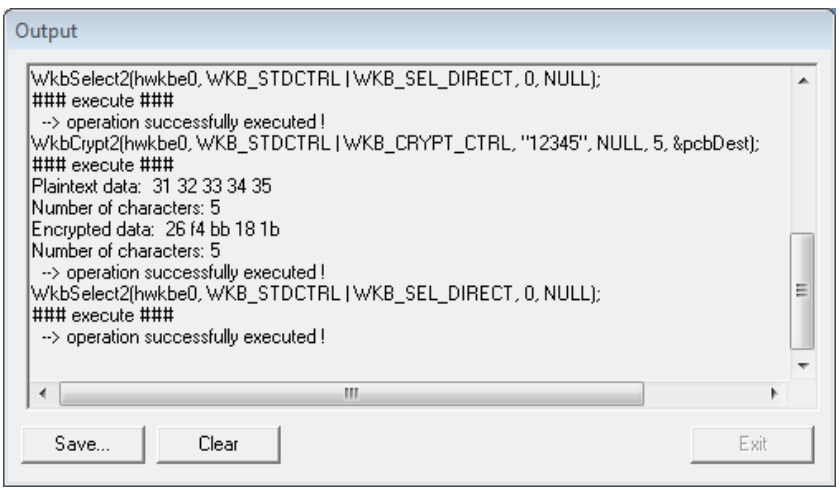


Figure 64: *WibuKey Interactive API Guide* (WKIAG) - Every function call is reported

Part XI *WibuKey COM control*

The *WibuKey* COM Component, represented by the WIBUKEY.DLL file, is the ActiveX component of *WibuKey*. This file is part of the standard WIBU-KEY User Environment, which is installed by the *WibuKey* Runtime Setup. Therefore the COM Component and its API may be used on user's site without any additional installation efforts.

The *WibuKey* COM Component can be used for any kind of *WibuBox* access, including encryption features, content listing and network access, it may be used also on developer's site to program *WibuBoxes*, or to prepare or to manage Remote Programming files.

Detailed information on the *WibuKey* COM Component can be found in the separate document *WibuKey - Guide for COM Control*.

Part XII Additional tips

1 More examples

In this section, a number of examples are discussed, which extend beyond the normal scope of description. These examples are aimed at giving the user a deeper insight into the most appropriate applications of **WKCRYPT**.

Commandline Examples

```
WKCRYPT /PAL2 /L /F10 /U13 /PN /L
```

This commandline programs a new entry in a connected *WibuBox*. The **/PAL2** option selects the printer interface LPT2 for the programming of the *WibuBox*. With the **/L** option the contents of the last *WibuBox* are displayed on the monitor. The next two options select the FIRM CODE 10 and USER CODE 13. With the **/PN** option, the entry 10:13 is programmed into the *WibuBox*. The second **/L** option displays the new contents for a visual control.

When the program DEMO.EXE is to be automatically encrypted and simultaneously a corresponding *WibuBox* programmed, together with a comprehensive visual monitor control, the following commandline is to be entered:

```
WKCRYPT /F10 /U13 /PN /CA /V DEMO.EXE
```

In the following example, all entries with the FIRM CODE 10 are erased in the *WibuBox* connected at the LPT3 port, and two new entries are programmed with the FIRM CODE 15 and respective USER CODES 28 and 35:

```
WKCRYPT /F10 /PAL3 /PC /F15 /U28 /PN /U35 /PN
```

WKCRYPT can be used to encrypt or decrypt complete discs or directory trees. For this purpose, the option **/S** is available in the file list (appendix B3). In order to automatically encrypt all TXT, DOC and WRI files in the directory \TEXT, including all corresponding sub-directories, the following line is to be entered:

```
WKCRYPT /F100 /U22 /CA \TEXT\*.TXT/S *.DOC *.WRI
```

Command File Examples

Longer instructions can be transferred to **WKCRYPT** with the aid of the commercial at (@). The *WibuKey* CD-ROM contains a series of model commands, which are presented in the following. These can be used as they are, modified, or expanded.

In the following, the command file with the name AUTO.WKC is exemplary used. It is a text file and consists of the following option settings:

```
/F10 # The Firm Code is 10.
/U13 # The User Code is 13.
/V /CA
/MS"WibuKey Automatic Decryption Test"
/ME"\aWibuBox \cf:\cu was not found!"
TEST.EXE
```

The first line specifies the FIRM CODE 10. This is followed by a comment, which begins with #, and extends to the end of the file. In a corresponding fashion, the USER CODE 13 is specified in the second line. The third line activates both the comprehensive output on the monitor, and the automatic encryption. The next two lines define character strings, which are entered in the file to be encrypted. The first character string is displayed on the monitor when the encrypted program starts. The second is displayed on the monitor when a *WibuBox* with the entry 10:13 is not found and the following message appears:

```
WibuBox 10:13 was not found!
```

It is important to note that **\a** in the character string is substituted for a bell, **\cf** for the current FIRM CODE, and **\cu** for the current USER CODE. Finally, the program file which is to be encrypted, and substituted for the encrypted file, is specified in the last line.

The command file is called as follows: **WKCRYPT @AUTO.WKC**

The start from the commandline may appear as follows:

```
WKCRYPT @AUTO.WKC TEST.TXT
```

The text file TEST.TXT will be substituted by the encrypted equivalent.

It is often necessary to program a number of *WibuBoxes* simultaneously. This is possible via the command file MULPGM.WKC:

```
/F10 /PQ%2 /V /PAL1 /U%1++ /V
/U30 /PN /L
```

The first line specifies the FIRM CODE and the port at which the *WibuBoxes* are programmed. The **/V** option activates a comprehensive process documentation. Two entries are generated per *WibuBox*: One should possess a consecutive USER CODE. The parameter is specified at call. The other possesses the USER CODE 30. Both receive the FIRM CODE 10. One single **WKCRYPT** start programs a number of *WibuBoxes* which is specified as a second parameter. Should, for instance, 5 *WibuBoxes* be programmed simultaneously with the entries 10:130, 10:131 etc. up to 10:134, in addition to the entry 10:30, one can enter the commandline:

```
WKCRYPT @MULPGM.WKC 130 5
```

A more complex command file MULPGM.WKC programs a *WibuBox*, encrypts a program automatically and copies it to a floppy in drive A: *simultaneously*, with just *one* **WKCRYPT** start. The file is called MULTI.WKC:

```
/F10 /U%2 /PA1 /PN
/W"\aEnter destination diskette into drive A:."
/CA /MS"WibuKey Automatic Decryption Test"
/OA:\ %1.EXE
```

This file possesses two dummy variables. The first specifies the name of the EXE program which is automatically encrypted. The type may not be subsequently specified! The second specifies the required USER CODE. A new entry in the *WibuBox* is programmed via the first file line, for which the FIRM CODE is 10. The second line issues a monitor message which informs that the encrypted program has been written. **WKCRYPT** then proceeds, when the operator strikes the “blank” key. The third line defines the automatic encryption of a program as monitor message. It is issued when the encrypted program is started. The last line specifies that the encrypted program is to be copied to the floppy in drive A:.

The command file MULTI.WKC may be called in the following manner:

```
WKCRYPT @MULTI.WKC TEST 295
```

The file TEST.EXE is encrypted with the USER CODE 295 and copied to the floppy in drive A.

2 How to get support?

One of the advantages working with *WibuKey* is a fast and qualified support. All customers have the possibility to get fast and qualified support on all *WibuKey* products.

Wibu-Systems offers its customers programming examples to realize an explicit encryption of their software. The examples are available in many programming languages and are distributed via the *WibuKey* CD-ROM. For additional examples ask the Wibu-Systems support team.

For international customers it is recommended to contact their Wibu-Systems distributor. They can help with any questions concerning *WibuKey*. A current list of all distributors is available on the Internet at **http://www.wibu.com**.

It is possible to contact the Wibu-Systems headquarters in Germany.

On the Internet you find a support page. It shows whom to contact for questions concerning special programming languages. The email address of the Wibu-Systems support team is **support@wibu.com**.

Part XIII Appendices

1 WibuKey.INI

```
; ---------------
; general setting
; ---------------
[General]      ; specifies the path to the WKWIN.DLL or WKWIN32.DLL if
SystemDir=     ; they are not in the usual folder

; --------------------
; global server setting
; --------------------
[Server]
WkLAN=1        ; default=1 enables the WkLAN part of the server
WkNet=1        ; default=1 enables the WkNet part of the server

; --------------------
; local driver setting
; --------------------
[Driver]
WkSubSystem=Kernel,WkLAN,WkNet       ; enables standard search of
available subsystems
WkMachine=IBMPC              ; WkMachine=NECPC disables default
                              setting
Diagnosis=0           ; no diagnosis events recorded
                      ; Diagnosis=1 enables record of diagnosis data
WkLpt=*,*,*,330/D     ; enables system default for LPT1-3
                      ; disables LPT4 (I/O-address already set for
                      ; WibuBox/CI) options after address:
                            ;/D=disabled,
                            ;/U=only unidirectional access
                            ;/B=only bidirectional access
                            ;/S=shared access
                            ; default (none): enabled, unidirectional
                            ; and bidirectional access

AcquireWait=5000      ; default value in ms for Windows 95 if shared
                      ; access is not specified for WkLpt
WkMinWait=100         ; default value in ms for DOS and Windows 3.x
WkCom=*,*,*,*         ; enables system default for COM1-4
                      ; options after address:
                            ;/D=disabled,
                            ;/R=force RTS line high
                            ;/T=force DTR line high
                            ; default (none): enabled, don't change
                            ; RTS/DTR
```

```
WkPcCard=3F                  ; enables PcCards
                             ; WkPcCard=3, INTEL, D0000 enable search
                             ; on all available ports
WkPort=Lpt,Com,PcCard, USB   ; removes comment to enable direct access
                             ; to PcCard Intel controller
; ------------------------------------------------------------------
; sample settings for WkNet server (Windows, Windows NT or Novell)
; ------------------------------------------------------------------
[WkNet 10:13]                                ; specifies Firm Code and
                                             ; User Code
ServerFile=c:\wibukey\wknet\wknet.dat        ; WkNet file specification
                                             ; for Windows server
ServerFile=dos:\test\wknet.dat               ; WkNet file specification
                                             ; for Novell Netware
RefreshTime=10                         ; WkNet file refresh time in
                                       ; seconds
TimeOut=30                             ; WkNet file time out in minutes
Selection=1234                         ; Selection Code

; ------------------------------------------------------------------
; sample settings for WkNet client (Windows 3.x, 95, 98, NT)
; ------------------------------------------------------------------
[WkNet Clients]
DrvTest=c:\wibukey\wknet\wknet.dat   ; path to WkNet file for specific
                                     ; application
WkDemo=c:\wibukey\wknet\wknet.dat    ; path for a second application

; ------------------------------------------------------------------
; sample settings for WkLAN communication (Windows 3.x, 95, 98, NT)
; ------------------------------------------------------------------
[WkLAN]               ; used UDP/TCP port: change default value
IpPort=22347          ; if it is used by another WkLAN server
; ------------------------------------------------------------------
; sample settings for WkLAN client (Windows 3.x, 95, 98, NT)
; ------------------------------------------------------------------
[WkLAN Client]                ; additional IP addresses allow the
Server1=255.255.255.255       ; search for a server on different
Server2=                      ; computers or subnetworks
                              ; default value = local network

; ------------------------------------------------------------------
; sample settings for WkLAN server (Windows 3.x, 95, 98, NT)
; ------------------------------------------------------------------
[WkLAN Server]
AccessFsb=0           ; no remote server access for Firm Security
                      ;WibuBoxes
NoBoxPgm=0            ; (hot) remote programming not invalidated
TimeOut=10            ; Timeout in minutes for WkLAN; default 24 hours

; ------------------------------------------------------------------
; sample server settings for HLM (Huge License Management)
; ------------------------------------------------------------------
[Server HLM files]
```

```
File1=c:\windows\sample.wbb   ; specifies the path to the HLM file
```

2 Notation specification

2.1 Key specifications

Despite the same function, some control and special keys possess different names, depending on the keyboard in question. The following exclusively uses the definitions for the international IBM-MF-II keyboard. The equivalent definitions of other keyboards are listed below:

Button	Other definitions
	Alt, AltGr
	PgUp, Page Up
	PgDn, Page Down
	Print Screen, PrtSc
	Insert, Ins
	Return, RET, Enter, ☐
	End, END
	Delete, DEL, Clear
	Home, HOME
	←, ↑, ↓, →
	Control, Ctrl
	→|
	Shift, ☐

  both keys must be pressed at the same time

 the keys must be pressed in succession

2.2 File specifications

A file name may not incorporate any blanks, since these terminate a file name or group. Should the necessity arise, the name must be entered in quotes ("...").

File Name Specification

`LETTER TEXT`	specifies two files, namely `Letter` and `Text`
"Letter Text"	specifies one file named `Letter Text`

With the file names, drive specifications and directories, DOS and OS/2 do not differentiate between capitals and lower case letters, provided international characters are used. The specified file name letters will, however, be substituted for the correct type. Those specifying a drive, for example, are fundamentally capitals. File names are stored with the letter type as entered. In the case of DOS and OS/2 with FAT drives, all international characters are converted to capitals. In contrast, in the case of OS/2 with HPFS, the characters remain as entered. In general, only those characters are allowed, as specified in the DOS or OS/2 guides for valid file names. The maximal length of a file name (for DOS and FAT drives on OS/2 this is 8 characters for the name and 3 characters for the type, and for HPFS drives on OS/2 a total of 254 characters). Names which are too long will be shortened accordingly by DOS, and not accepted by OS/2. File specifications within a file must be specified within a single line.

Whenever the designation *FileName* appears in the documentation, the following specifications are possible:

- A single file name with or without type extension, e.g. `TEST.DAT`,`TEST.` or `XYZ12_4.ERW`. When a specification contains a type extension or ends with a decimal point, a program will not append a default type extension. If the file name is without a type specification the program will automatically supplement a specific type name, or uses no type extension. Under Windows 95 or Windows NT a file name can contain various decimal points.

File Extension

Assembler will supplement **TEST** to **`TEST.ASM`**. In contrast, the DOS command **TYPE** assumes that the file with the name **TEST.** is concerned.

- A file name is defined as a *FileGroup* to represent a number of files. The file name then contains a question mark (**?**) or wildcard (*****). A file name may only then contain a question mark or wildcard, when explicitly stated that instead of just a single file name, a complete file group may be specified.

- Prior to the actual file name, one can specify a drive name and/or directory. When the directory begins with a backslash, the directory will be sought as from the drive main directory (root). When no backslash is specified at the start, the search commences from the current directory. Following the drive specification or directory, the file name is to be specified. The programs do not generate new directories.


For further details concerning the description of files, drives and directories, please refer to the corresponding DOS and OS/2 guides.

2.3 File lists

An extension of the file group is the *File list*. This allows files to be sought within a directory (or, if necessary, recursively in all sub-directories), with respect to all file groups specified within the file list.

The general configuration of a file list is based on the following scheme:

```
[Drive][Directory]FileName [/S] [FileName] …
```

Any number of file names may be listed. The specification of the drive or directory is, however, only allowed in conjunction with the first name. When the option **/S** is specified, the files in all sub-directories of lower levels will be drawn on in succession, beginning with the highest sub-directory.


File List Specification

`TEST1.TXT` `TEST2.TXT`	searches for `TEST1.TXT` and `TEST2.TXT` in the current directory.

`C:\TEXT\*.TXT/S`
`TEST?.ASM`
`TEST*.C`
	searches for all files which correspond to `*.TXT`, `TEST?.ASM` and `TEST*.C` in the `TEXT` directory of drive `C:`, and all sub-directories.

`C:\TEXT\*.*/S`	searches for all files within the `TEXT` directory of drive `C:`, and then in all sub-directories of a level lower than `C:\TEXT`.

2.4 Commandline specifications

When a program is called from a DOS or OS/2 commandline, or from a DOS-BAT file or OS/2-CMD file, the command argument, which designates an EXE or COM program, is to follow the command name. This is defined as the commandline, also when this is within a DOS-BAT file or OS/2-CMD file.

Each program possesses an individual commandline scheme. For Wibu-Systems products, however, the configuration of the basic elements of such a line is standardized. Basic elements are, for example, file specifications or options. The options which a program actually possesses, and the manner in which these are to be configured is program-specific. In contrast, the specification mode of an option is standardized. For a number of programs, options and file specifications may appear in command files or **SET** assignments. The term commandline remains valid.

2.5 Options

Options always begin with a hyphen (`-`) or slash(`/`). Each option is specified by one or more characters which must directly follow the introducing character. Some options have arguments e. g. a number or file specification.

No blanks may be entered between an option specification and its argument. There are, however, a few exemptions, which serve to improve the clarity of the command.

Should an option be followed by another option, or by a file specification, the specifications should be separated by at least one blank or tab for clarity. The arguments of options containing a hyphen or slash, however, may be mistaken for the next option, hence leading to unexpected results.

For most Wibu-Systems program options, it is irrelevant whether the specifications are made with capitals or lower case letters. This holds for the complete program. The characters of names etc. are not changed, except for file names, where all lower case letters are converted to capitals. Some programs differentiate between capitals and lower case letters in the options. These particular instances are explicitly defined in the program description. It then holds for all options with the exception of file specification and hexadecimal values.

As far as this documentation is concerned, the options are specified in the standard fashion, i.e. with hyphen and capitals.

2.6 Numbers

When the input notation refers to the term *number*, then a whole number may be entered as a decimal number, if necessary, with a preceeding minus character (`-`). Its valid range is explicitly stated in the text. The specified limits (from … to) are inevitably also valid “inclusive” values. Entered numbers may not include neither blanks, tabulators nor any other control characters, just the digits `0` to `9`.

Should, instead of the term *number*, the definition *address* or *SystemNumber* be mentioned, and then the value may be entered as a decimal or as hexadecimal

number. In these instances, the program language "C" is to be used, which is as follows:

- Decimal values are specified for *number* in digits from 0 to 9, without a leading 0 (e.g. 013 is an invalid entry for 13).
- Numbers which begin with a zero, followed by an **X** or **x**, are base 16 numbers (hexadecimal). These are defined by the digits 0 to 9 and A to F, or a to f for values between 10 and 15 for each order. Other letters may not be used. The letters terminate the number.

Entries in the field *Address* or *SystemNumber*

```
1      = 0x1
12     = 0x0C
100    = 0x64
512    = 0x200
65535  = 0xFFFF
```

2.7 Character strings

When the term *character string* appears in a command description, a number of characters can be specified, which define a brief text. The characters should be incorporated within quotes ("..."). Blanks and tab stops may be included. These will be returned by the program exactly as entered. Should the text include quotations marks, and then one must take care to precede these with a backslash ("\"). This backslash serves to depict special characters, as in the program language "C". When this backslash itself is required in the string, it must be entered twice. Altogether, the following special characters are presently valid:

Character	Description
\a	(*audible alert*). This is converted to a BEEP-character (value 7).
\f	(*form feed*). Is converted to the FORM FEED character (value 12).
\n	(*new line*). This is converted to the *carriage-return* (value 13) and *line feed* (value 10) characters.
\r	(*carriage return*). This is converted to the *carriage return* value (value 13).

Character	Description
`\t`	(*tabulate*). Is converted to the *horizontal tabulator* character (value 9).
`\v`	(*vertical tabulate*). Is converted to the *vertical tabulator* character (value 11).
`\"`	is converted to the " character.
`\'`	is converted to the ' character.
`\?`	is converted to the ? character.
`\\`	is converted to the \ character.
`\ooo`	is converted as three octal digits ooo (0 up to 7) into a value from 0 up to 255.
`\xhh..`	is converted the followed hexadecimal digits (0 up to 9, A up to F or a up to f) into a character value.

Further characters are possible with specific character strings, which are defined at the relevant locations.

The confining quotes may be dispensed with when the character string contains neither blanks nor tabulators, nor special characters entered in conjunction with “\”. In such cases, the backslash is of no relevance, and will be treated as a normal character.

Confining Quotes

String	results in	
`AbCdE`	results in	`AbCdE`
`"AbCdE"`	results in	`AbCdE`
`Ab CdE`	results in	`AbC and dE (two strings)`
`"Ab CdE"`	results in	`Ab CdE`
`"Ab\"CdE"`	results in	`Ab"CdE`
`"Ab\\CdE"`	results in	`Ab\CdE`
`Ab\"CdE"`	results in	`Ab\CdE`
`Ab\\CdE`	results in	`Ab\\Cde`

When a character string is specified as parameter of a DOS command, the character are interpreted compatible with the IBM character set 437, which is incompatible with the Windows character set for characters with bit 7 = 1. To correctly store umlauts, for example, into a Windows string, the \ooo or the \xhh escapes should be used.

The escape **\x** reads characters continuously until no valid hexadecimal digit follows. This avoids the specifications of digits of letters immediately after this escape. A solution of this problem is to terminate the complete string after the last digit of the escape and to start a new string with the digit or letter which should be displayed. All *WibuKey* products support the specification of subsequent character strings.

Identical Character String Specification

The following specifications are identical:

```
"ABCDEF"
"ABC""DEF"
"A""B""C""D""E""F"
```

The string `"|\x070|"` is the same as `"|p|"`

The string `"|\x07""0|"` is the same as `"|\b0|"`

2.8 Data aggregates

A data aggregate is a description of subsequent bytes which is used to generate data for encryption via the **WKCRYPT** application or to specify the initialization of an indirect encryption. An aggregate has following general form:

```
((SystemNumber|CharacterString)[:SystemNumber])[,] ...
```

If a number is specified, then this represents a single byte value in the range 0 to 255. As alternative, a character string may be specified which represents one or more characters, stored as subsequent byte values. Such a string is **not** terminated automatically by a 0 byte. If after the number of the string specification a colon (":") follows, then a repeat factor follows which specifies the number of repeated store operations of the byte sequence which is specified in front. If no colon follows, then the default value is 1. One ore more subsequent specifications with optional repeat factor may be specified, separated by comma.

Aggregate Specification

Aggregate specification	Resulted byte sequence (hexadecimal)
0,1,2,3	00 01 02 03
"ABCD"	41 42 43 44
"A":3,"B":2,"C":1,0	41 41 41 42 42 43 00

Aggregates used in commandlines are surrounded with braces, e.g.

```
{0}
{1,2,3,4,5}
{"TestString",0}
{1:5,"Test",0}
```

2.9 Delta expressions

A delta expression is used to express a relative change of a numerical value. For example to increase or decrease an existing value. The syntax of the delta expression is similar to the syntax used in the C programming language:

Expression	Description
`++`	increases a value by `1`
`--`	decreases a value by `1`
`+=number`	increases a value by the given number
`-=number`	decreases a value by the given number

Delta Expressions for User Codes

```
/U10++
/U100--
/U5+=10
/U1000-=5
```

The four examples will result in the following User Code sequences:

```
10, 11, 12, 13, ...

100, 99, 98, 97, ...

5, 15, 25, 35, ...

1000, 995, 990, 985, ...
```

2.10 Date specifications

All *WibuKey* programs use the same specification for dates: *YearMonthDay*

Year means a value between 00 and 99 for the following date range:

`70..99` for the year 1970 to 1999

`00..69` for the year 2000 to 2069

Month has to be one of the following abbreviation for a month:

`Jan`, `Feb`, `Mar`, `Apr`, `May`, `Jun`, `Jul`, `Aug`, `Sep`, `Oct`, `Nov`, `Dec`

Day means a value in the range from 00 to 31 to specify the day within the month.

Date specifications

```
80Feb29
97Dec31
05May10
```

The dates above are in the year 1980, 1997 and 2005.

Dates as `97FEB29` that are not valid are recognized and an error message is issued. Dates outside the range of `70Jan01` to `69Dec31`are not allowed.

2.11 *WibuKey* port numbers

A “port number” serves to designate interfaces or ports to which WibuBoxes can be specified. These may be the parallel interfaces *LPT1* to *LPT4*, serial interfaces *COM1* to *COM4*, PCMCIA interfaces *PcCard1* to *PcCard6* or a direct numeric port specifications (`#0` to `#127`). The place holder *BoxPort* may therefore, be allocated as follows:

([**L**|**C**|**P**|**U**][**1**|**2**|**3**|**4**]) | (*#Number*)

When the port number begins with an *L* or directly with a number between *1* and *4*, then this refers to *LPT1* to *LPT4*. The *C* refers to *Com* and the *P* to *PcCard*.

In the absence of any port number, all available interfaces of the specified type will be accessed. This is, however, only possible for the listing of the WibuBoxes, not their programming.

The port specification variations of the **`WKU/WKU32`** program is a little more relaxed: Every abbreviation of a port is possible, for example P1, PC1, PCC1, PCCa1, PCCar1 or PcCard1 specify all the PcCard1 port.

When the port number begins with a number character (#), then the direct port number in the range between 0 to 127 is to follow. The port numbers 0 to 7 are reserved for LPT, 8 to 15 for COM, 32 to 37 for PcCard1 up to PcCard6, 40 for ADB and 48 for USB. The values 16 to 31, 41 to 47 and 49 to 63 are reserved. Values from 64 upwards can designate OEM-specific *WibuBox* ports. The correlation between the number and its reference may be extracted from the following list:

***WibuBox* Port Numbers**

The following values are identical:

Port	Port Number	Default specification
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
PcCard1	#32	P1
PcCard2	#33	P2
PcCard3	#34	P3
PcCard4	#35	P4
PcCard5	#36	P5
PcCard6	#37	P6
ADB	#40	ADB
USB	#48	U

3 Encryption algorithms

3.1 Knuth-Algorithm

The Knuth-20 algorithm is a very fast and simple algorithm without a very high security which bases on the random generation principle of *Donald E. Knuth*.

3.2 Original FEAL

The Original FEAL algorithm implements the original FEAL algorithm of the Japanese NTT institute. It can be parameterized with a round factor of 8, 16 or 32. A higher round factor increases the security dramatically, but reduces on the other side the encryption speed. The FEAL algorithm is slow and not symmetrical. Furthermore it can be used only for data with multiple of 8 bytes (block chiffre).

3.3 Symmetrical FEAL

The Symmetric FEAL algorithm is a variation of the original FEAL. It is not so safe but symmetrical and can be used for data streams of any length. Strictly spoken the **Symmetric FEAL** is a Wibu-Systems own combination of the Knuth20 and the FEAL algorithm with a final Exclusive-Or encryption.

3.4 Permutation

The **Permutation Algorithm** doesn't change the value of the data to be protected but the position of the data objects in the data. For that, the length of such a data object can be specified as 1, 2, 4, or 8 bytes. The length of the complete data sequence must be a multiple of the specified object length. The permutation algorithm is not symmetrical. The big advantage is that, no new data values are introduced. It is not possible, that for example a 0 character is introduced in the middle of the string which would be interpreted at the erroneous end of the string. The Permutation Algorithm is faster than the FEAL algorithm but slower than the Knuth algorithm.

4 Program return codes

Return codes are issued by commandline programs at termination, upon reverting to the start location (operating system, BAT or CMD file, MAKE file or other program). Here, the value can be analyzed for further processing. The return value 0 always refers to a program which terminated without any errors. Other codes indicate errors or an untypical termination of the program.

4.1 Use of return codes via MAKE

MAKE is a program offered by various different producers. This serves the automation of the software development or the creation of data stock. A **MAKE** file contains one or more commandlines which, for example, serve to convert source files to object files, eventually with the aid of compiler calls. In such cases, only those source files which possess an update date earlier than that of the created object file itself will be converted. These time dependencies are specified in a **MAKE** file and allocated with commandlines and when necessary, processed. Each commandline corresponds to a commandline as normally processed by **COMMAND.COM**, i.e. they begin with a command name, which may also be a COM or EXE program, followed by the command argument.

MAKE programs for DOS are offered by various different producers, including Microsoft (NMAKE), Phoenix, Watcom (WMAKE) and Borland).

Nearly all MAKE programs possess the common feature that the return code of the program, located in the MAKE file commandline, is evaluated and called. When the return code is not zero, an error message is usually printed and the complete MAKE sequence is aborted.

As such, when MAKE programs are used, the return code may be easily established.

4.2 Use of return codes with DOS bAT files and OS/2 cmd files

BAT or CMD files serve to summarize a number of DOS or respective OS/2 commandlines in a single file, which can then be started by specifying its name (batch sequence). BAT and CMD files are an integral part of the operating system and are comprehensively described in the corresponding User's Guides. The return code of executed programs can also be used in BAT and CMD files. For this purpose, the BAT/CMD command

```
IF [NOT] ERRORLEVEL Number Command
```

should be used.

The semantics of the above syntax is as follows: Should the program executed prior to the last `IF` command a return code equal to or higher than *Number*, *Command* will be executed, provided that `IF` is not followed by `NOT`, in which case, the process branches to the next BAT or CMD file line.

When `NOT` follows `IF`, *Command* will be executed provided the return code of the program executed previous to the `IF` command is lower than *Number*. Should more than one command be executed instead of *Command*, then a jump label is to be set prior to the command group, and instead of *Command*, a `GOTO` instruction must be specified.

4.3 Return codes of programs in this documentation

Wibu-Systems programs use standard return codes. 0 always means that a program has been executed correctly and has completed the desired task. Values greater than 0 indicate errors with increasing return code values for increasing error significance:

Code	Description
`0`	No error. Program/data encrypted correctly, *WibuBoxes* correctly programmed.
`1`	The program could not be executed, due to an incorrect command argument or, for example, because the on-line help was called. Before the program can run correctly, the specified parameters must be changed. When programming with **FCRYPT**, this error code will also be returned when the specified WibuBox was not found, so that the encryption fails
`2`	The program was not completely executed, because the specified parameters lead to errors, for example: file not found, contradictory modes, or wrong data formats etc.
`3`	The program was not or only incompletely executed as a result of limitations of the program or the operating system. This includes files which are too complex or too long, functions which are not implemented or the occurrence of internal program errors.
`4`	The program was not or only incompletely executed because a serious hardware problem occurred for example a communication error between the program and a connected *WibuBox*.
`11`	The program was terminated by the user via the input of 

Code	Description
	or .

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X