CBIOS4NET/Smarx4NET: Object oriented component based syntax for .NET platform, covering CRYPTO-BOX SC and CRYPTO-BOX XS/Versa

Developer's Guide

Document: 0-9Nov011af(CBIOS4NET Developers Guide).odt
Last update: 1. September 2017 by Steffen Kaetsch
Environment: Microsoft Visual Studio 2005-2017 C#.NET

Executive Summary

*This document defines concept, implementation details and syntax of **Smarx4NET** and **CBIOS4NET** - object oriented component based SmarxOS API for .NET platform. Rapidly growing popularity of C#.NET programming environment and ASP.NET based web applications makes CBIOS4NET project of key importance for MARX customers.*

C# programming community got used to object oriented component based way of software development (main benefit of .NET platform).

Smarx4NET combines the following Smarx OS programming interfaces under one roof for .NET platform:

- *CBIOS and DO API (network mode only);*
- *RU API - RU4NET (CBIOS4NET.RFP)*

CBIOS4NET combines all Smarx OS programming interfaces under one roof for .NET platform:

- *CBIOS (local and network mode) and DO API;*
- *RU API - RU4NET (CBIOS4NET.RFP);*
- *CBIOS network administration API;*
- *DP API - DP4NET.*

Smarx4NET and CBIOS4NET cover the following types of MARX hardware:

- *CRYPTO-BOX® XS and Versa (CBU);*
- *CRYPTO-BOX® SC (CBU SC).*

The main advantage of Smarx4NET compared to CBIOS4NET: 100% managed code – no need for platform specific libraries or VC redistributables, plus support for Windows Store applications.

1. Introduction

1.1 Overview

The popularity of C#.NET is growing up rapidly and ASP.NET becomes a leading choice for web application development.
With **CBIOS4NET** and **Smarx4NET**, MARX provides an object oriented component based API for .NET platform which combines multiple Smarx OS programming interfaces:

- CBIOS API – core API, providing key features for accessing the CRYPTO-BOX on local computer and in networks (see SmarxOS Compendium, chapter 11 and 12 for an introduction to CBIOS).
- DO (DataObjects) API – provides management of Data Objects stored in CRYPTO-BOX memory (see SmarxOS Compendium, chapter 13 for an introduction to CBIOS DO).
- Remote Update functionality (**RU4NET**), providing update of the CRYPTO-BOX directly on the en-user side (see SmarxOS Compendium, chapter 14 for an introduction to Remote Update).
- Data Protection (DP) API (**DP4NET)** – access to data filter driver
- CBIOS network administration interface to manage CBIOS Server remotely.

Smarx4NET (introduced in 2014) combines CBIOS Networking, DO and RFP programming interfaces under one roof for .NET and .NET for Windows Store / Windows Phone developers.

CBIOS4NET combines all SmarxOS programming interfaces (including local CBIOS and DP API) under one roof for .NET developers.

Both interfaces support CRYPTO-BOX XS/Versa (CBU) and CRYPTO-BOX SC (CBU SC) hardware units.

This document focuses on implementation for .NET. For a general introduction to API development with the CRYPTO-BOX we strongly recommend to read our White Paper “Implementation with API” first. For an introduction on how to access the CRYPTO-BOX in networks (especially important for Smarx4NET), please refer to the Smarx Compendium, chapter 13.

1.2 Differences between Smarx4NET and CBIOS4NET

1. Both interfaces are close to each other. Some methods and type names are different.
2. Smarx4NET supports network mode only: if using on a single computer, the CBIOS Network Server must be running on localhost or on a remote computer (see Smarx Compendium, chapter 5). CBIOS4NET supports both local and network modes.
3. Smarx4NET does not need VC Redistributables. It only requires Framework.NET 4.5 or later. CBIOS4NET supports .NET framework version 2.0 and up, too, but requires VC redistributable of the corresponding MS Visual Studio environment.
4. Smarx4NET does not requires a platform specific (x64 or x32) assembly as CBIOS4NET does (where CBIOSLoader.cs/.vb is required to load platform specific assembly)
5. Smarx4NET is based on interfaces, not on classes like CBIOS4NET. Class instance creation is performed through class factory (Smarx class)
6. Smarx4NET supports quick search of servers & attached hardware for net broadcasting with asynchronous socket model (see SmarxNetworkLicensing sample in PPK). Direct connect won’t

hang up even when server is switched off (happens on CBIOS4NET network mode)

Conclusion:

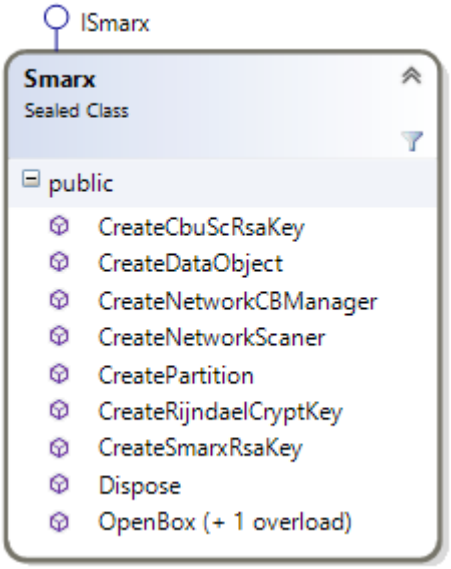
The biggest advantage of Smarx4NET compared to the older CBIOS4NET interface: it is fully managed code which makes implementation more flexible and requires no VC redistributables to be preinstalled. It supports standard C# applications as well as Windows Store applications, and is ideal for protecting multi-device applications for desktop and mobile usage.

Smarx4NET runs in network mode only, which requires an installation of the CBIOS Server (either on the same computer or in the network).

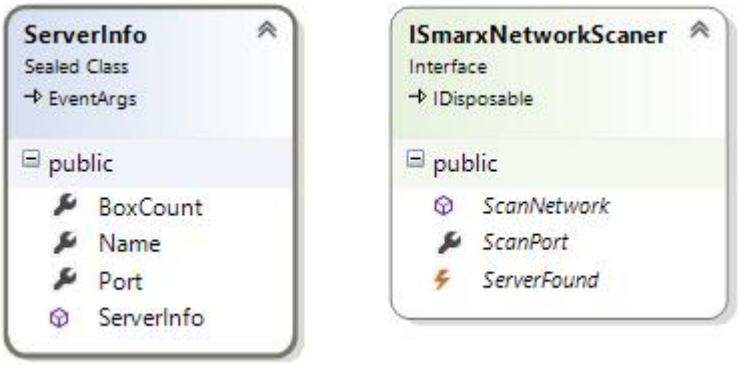
However, if you already use CBIOS4NET, support only local licensing scenario in your .NET application and you do not need support for Windows Store, it makes sense to stay with CBIOS4NET.

1.3 Comparing Smarx4Net and CBIOS4NET interfaces (technical details)

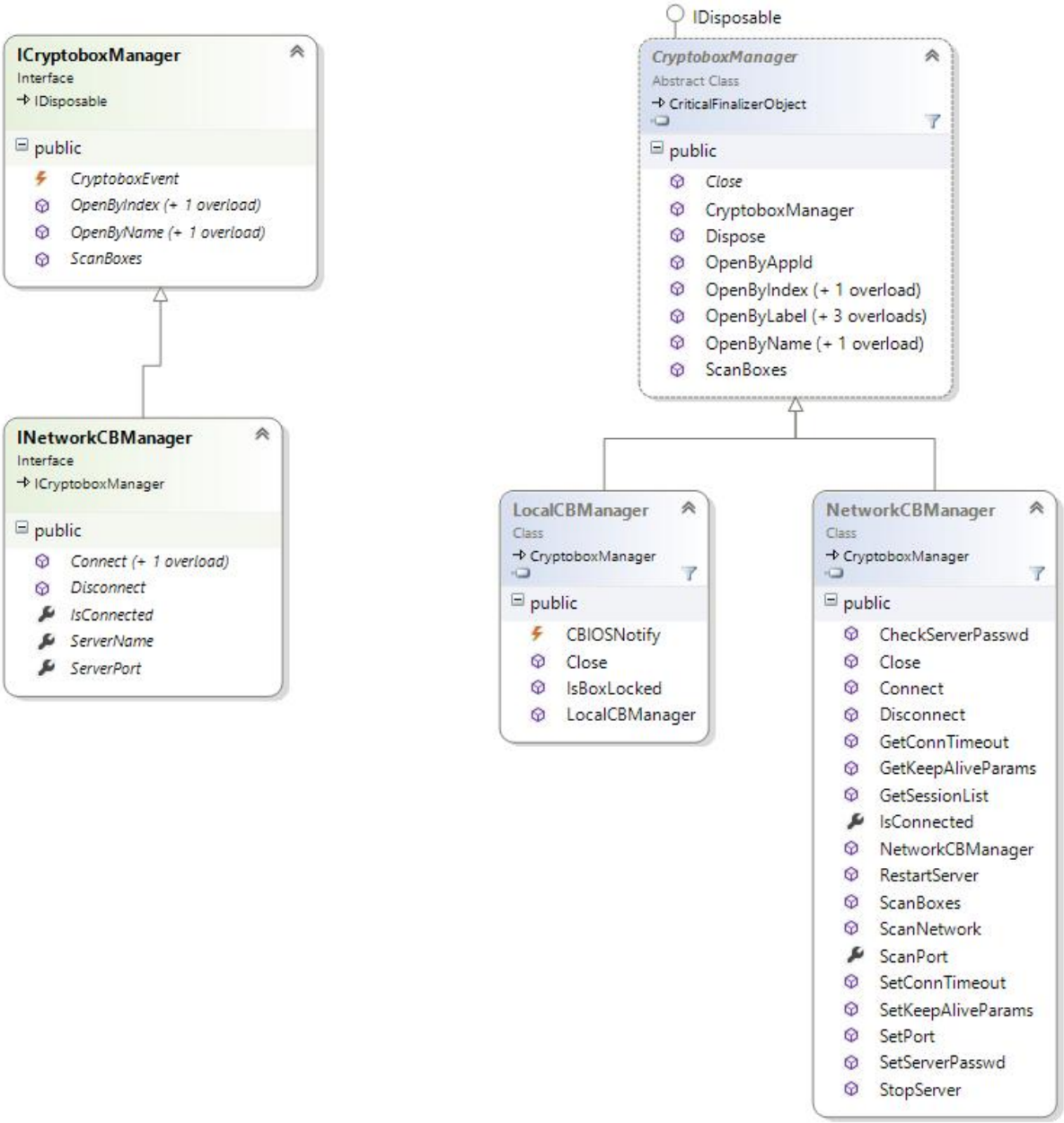
1. An instance of **NetworkCBManager, SmarxNetworkScaner, Cryptobox, CbuScRsaKey, SmarxRsaKey, RijndaelCryptKey, Partition, IDataObject** is created from new class **Smarx**:



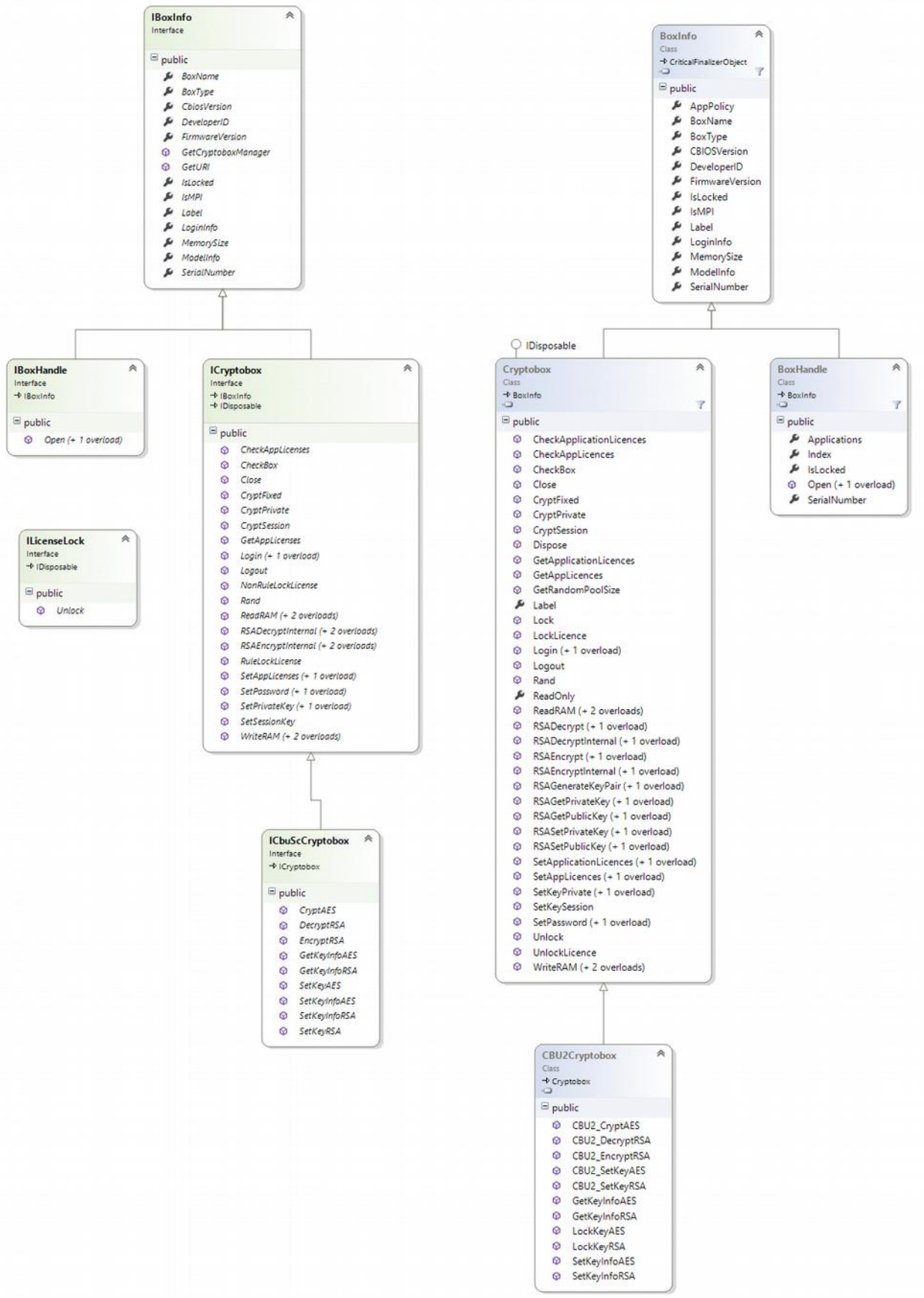
2. **New ISmarxNetworkScaner** interface is used to find MARX servers on LAN:



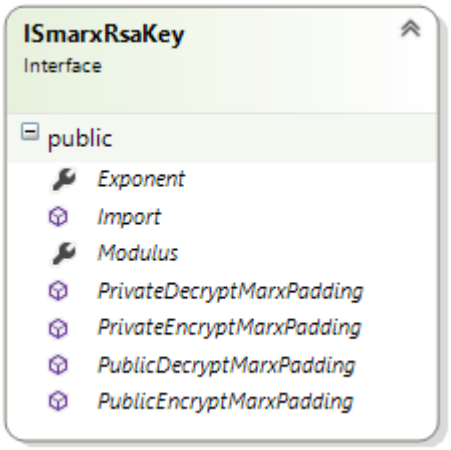
3. Local cryptobox manager and server admin management is not implemented (the right picture is CBIOS4NET):



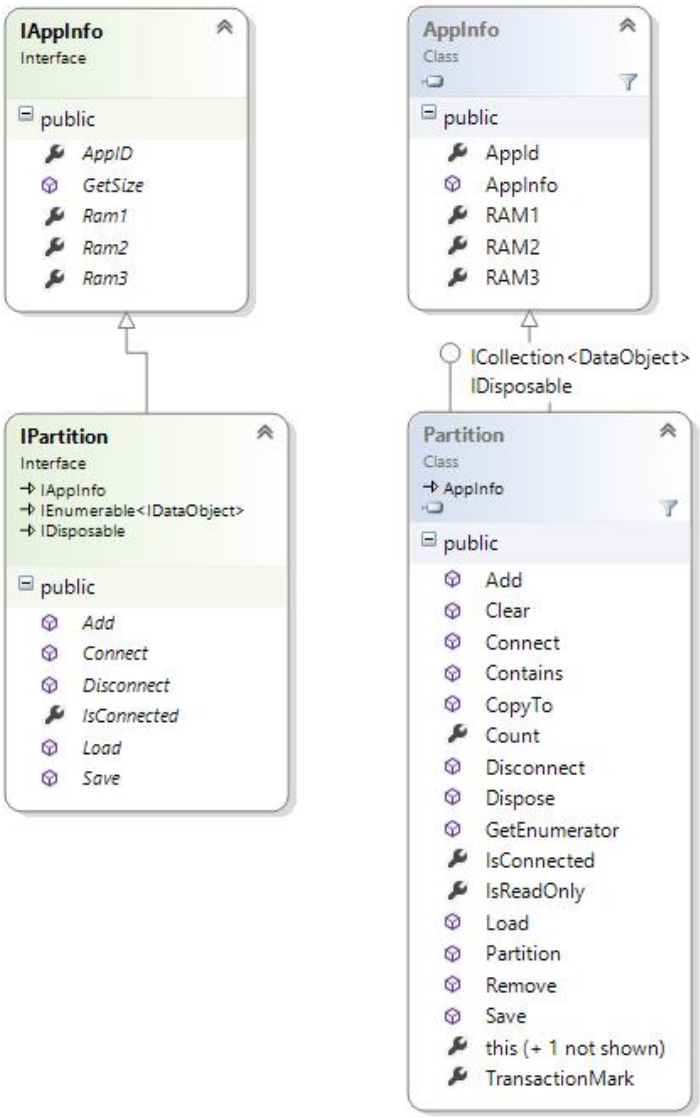
4. New **ILicenseLock** interface is used to free a license:



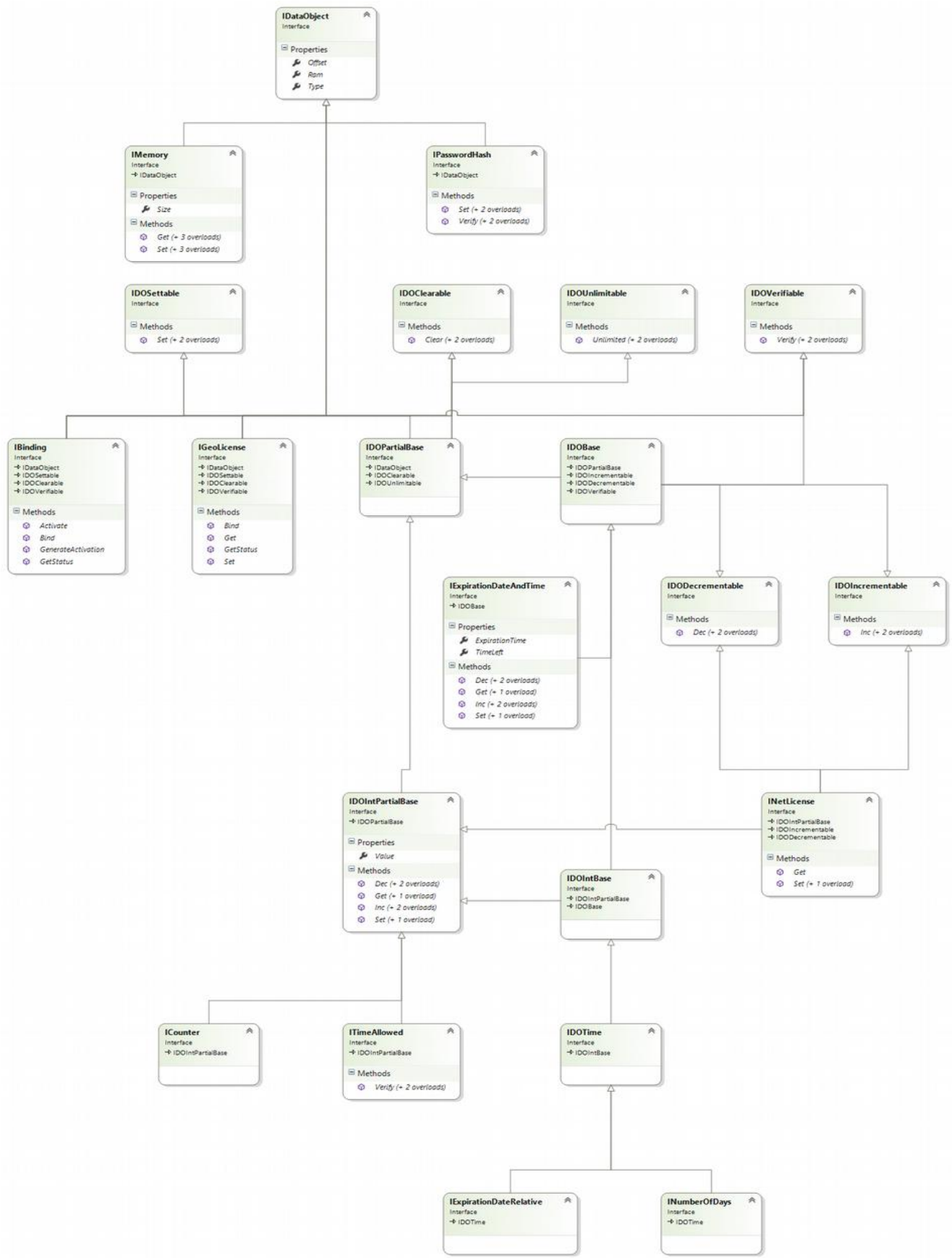
5. Software RSA encryption with MARX padding was implemented (for compatibility reasons):



6. The result of DO collection saving (serialization and deserialization) in Smarx4Net is not compatible with DO map in CBIOS4NET and in other environments:



7. DOs use multiple inheritance of interfaces:



1.4 CBIOS4NET – family of Smarx OS programming interfaces for .NET

As mentioned in chapter *1.1*, CBIOS4NET assembly combines CBIOS, DO, RU, DP, and CBIOS network administration SmarxOS interfaces. All these programming interfaces are represented with corresponding namespaces.

The **CBIOS** namespace contains implementation of CBIOS, DO and CBIOS network administration interfaces. The classes introduced in this namespace cover:

- Basic CRYPTO-BOX management:
  - obtaining information on currently attached CRYPTO-BOX units;
  - reading/writing data;
  - performing hardware based encryption;
  - handling of plug/unplug notifications;
- Manipulating Data Objects;
- Controlling CBIOS Server.

Chapter 2 of this guide illustrates component model of CBIOS namespace, displaying its key classes and methods.
Chapter 5 (5.1-5.3) provides examples of usage, while chapter 6 contains description of its classes, methods and properties.

The **CBIOS.RFP** namespace covers Remote Update API. This namespace is intended for implementation of customer specific remote license update logic. The background approach assumes three logical steps:
1) request for update creation (end-user's side)
2) request for update processing and activation code generation (distributor's side)
3) activation code processing and license data updating in the MARX hardware (end-user's side).
The request for update contains end-user's hardware specific data. Based on this data the distributor chooses corresponding update options and creates activation code for this end-user. The **RU4NET** general architecture (component diagram) can be found in chapter 3 of this document. Section 5.4 contains example of its usage. Chapter 7 provides detailed description of RU4NET classes.

The **CBIOS.DP** namespace implements classes representing DP API. This namespace is included only to Win32 build of CBIOS4NET. The DP API utilizing MARX Data Filtering technology allows customers to encrypt/decrypt their confidential files on the fly, using fast AES encryption algorithm. Only authorized process/application is allowed to access protected file. The encryption process can be suspended/resumed at any time (can be easily bound to MARX hardware plug/unplug events). The **DP4NET** component diagram is provided in chapter 4. The CBIOS.DP (DP4NET) syntax is documented in chapter 8 of this document.

This document assumes that the reader is familiar with Smarx OS general logic and its programming interfaces: CBIOS, DO, RU, DP and CBIOS network administration APIs. The Smarx4NET/CBIOS4NET assembly only structures these APIs into object oriented model. Such basic notions as: partition, data object, encryption key and more are described in the Smarx Compendium, chapter 10.

1.5 Smarx4NET/CBIOS4NET internal structure and run time requirements

Smarx4NET is fully managed code and does not requires a platform specific (x64 or x32) assembly or additional VC redistributable components (see also chapter *1.2*)

CBIOS4NET is not a 100% managed code assembly. In fact, it is a wrapper of the Smarx API for .NET environment, implemented on top of its kernel: CBIOS, RFP and DP C/C++ static libraries. The kernel contains unmanaged, CPU and platform (Win32/Win64) specific code.
This architecture implies the following run time requirements for applications using CBIOS4NET:

1) Framework.NET version 2.0 SP1 or later
2) C/C++ CRT (run time) of MSVS2005 - MS VC++ 2005 redistributable (or corresponding MSVC++ 2008-2013 redistributable in case of Visual Studio 2008-2013) for must be available for unmanaged part of CBIOS4NET code

These components can be freely downloaded through Windows Update or included to the product setup. Current version of MARX Analyzer (the latest MA version can be always downloaded through online update) includes comprehensive diagnostics of CBIOS4NET prerequisites.

More details on how to get the required redistributables can be found in the Protection Kit. Have a look at the readme file in the folder with the CBIOS4NET libraries:
<Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET 2\
or
<Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET 4\

For Smarx4NET prerequisites, see <SmarxOS PPK root>\SmarxOS\API\Win\SDK\dotNET 4.5\

1.6 CBIOS4NET: targeting Win32 and Win64 platforms

.NET 4.x and Smarx4NET users can ignore this chapter.

For .NET 4.x platform specific loader is obsolete: The CLR will load corresponding x86 or x64 CBIOS4NET assembly from the GAC. See readme file in the corresponding library folder for your .NET version (Protection Kit):
<Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET [x]

To install CBIOS4NET assembly into GAC "CBIOS4NET Setup" can be used which is available at www.marx.com → Support → Downloads → Driver and Diagnostic Tools.

As mentioned in 1.5 CBIOS4NET contains unmanaged code and is CPU and platform specific because of it. Separate builds of CBIOS4NET are available for Win32 and Win64 platforms:
CBIOS4NET.DLL is a 32 bit assembly and CBIOS4NET64.dll is a 64 bit assembly, correspondingly. You should use compatible build of CBIOS4NET for your .NET application. See section 1.6.1 for information on dynamic load of platform specific build of CBIOS4NET (x86 or x64).

Note that 64 bit assembly version does not contain Data Protection interface (DP4NET), because there is no 64 bit version of corresponding static library (SMRX_DP) at this time.

1.6.1 CBIOSLoader: dynamic loading of a platform specific build of CBIOS4NET (x86 or x64)

A special solution (CBIOSLoader) is included to SmarxOS PPK providing dynamic load of required

CBIOS4NET assembly (x86 or x64) depending on the current platform.

- <Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET 2\CBIOSLoader.cs

or:

- <Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET 4\CBIOSLoader.cs

How to use it:

1) prepare your solution;
2) place CBIOS4NET.dll & CBIOS4NET64.dll (signed build with/without strong name - see 1.4.1) assemblies in some directory relative to your final project binaries, for example: ..\bin\asm\
3) add reference to one of them (depending on you development platform)
4) set reference property "Copy local" to “false”
5) add the CBIOSLoader.cs file to your project
6) modify your entry point method with initialization of CBIOSLoaderHelper class like this:

```
static void Main ( )     {
       CBIOSLoader.CBIOSLoaderHelper.Initialize(Path.GetFullPath("..\\asm"));
       // Your code goes here
}
// the Initialize method parameter is a path to a directory
// containing CBIOS4NET*.dll files
```

7) IMPORTANT: the above code must be executed prior any call to CBIOS4NET. Moreover, it can not be placed to a method containing any CBIOS4NET calls.

1.7 CBIOS4NET and .NET platform

1.7.1 Signed build with strong name

Because of the fact that not all .NET programming environments support signed strong named assemblies, CBIOS4NET is provided in two separate builds: signed and signed strong named.

Digitally signed build of CBIOS4NET can be found in the Protection Kit (PPK):

- <Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET 2\x86\ - for Win32;
- <Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET 2\x64\ - for Win64

Digitally signed build with strong name (strong named) is here:

- <Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET 2\x86\Signed - for Win32;
- <Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET 2\x64\Signed - for Win64;

CBIOSLoader for dynamic loading of platform specific CBIOS4NET build (see 1.3.1 for details) is here:

- <Smarx PPK root>\SmarxOS\API\Win\SDK\dotNET 2\CBIOSLoader.cs

Libraries for .NET 4.0 version (Visual Studio 2012) can be found in the corresponding \dotNET 4 sub-directory.

1.7.2 Exception handling

CBIOS4NET doesn’t include its own structured system of exceptions. Instead all return codes for CBIOS, DO and RU interfaces are integrated into one CBIOSException. The ErrorCode contains detailed description of the error (more information on it can be found in CBIOSException class description). For DP API a similar

approach is used - DPException class iintegrates all return codes. Currently text messages of CBIOSException and DPException classes are not localized. If required it can be done by the customer based on the ErrorCode value.

1.7.3 Multithreaded architecture

One of CBIOS limitations – any thread can work with one box at a time. In order to work with several boxes simultaneously it is necessary to create a multithreaded application architecture having separate thread per every box. The CBIOS4NET classes are implemented in such a way to hide multithreading. Every **Cryptobox** instance already contains dedicated internal thread, so all box specific operations (including communication with its data objects) are performed in this thread.

1.7.4 Notifications and events

The CBIOS4NET assembly provides customers with one event to be used for local box plug/unplug notifications.
This event is always called in context of CBIOS thread. The following example illustrates correct event usage for GUI thread:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        LocalCBManager.CBIOSNotify += new
              CBIOSNotifyEventHandler(LocalCBManager_CBIOSNotify);
    }
    delegate void NotificationDelegate();
    void LocalCBManager_CBIOSNotify(CBIOSNotifyEventArgs e)
    {
        if (this.InvokeRequired)
        {
            NotificationDelegate del = delegate()
            {
                this.Text = e.NotificationType.ToString();
            };
            this.Invoke(del);
        }
    }
}
```

1.7.5 NET standard cryptography and its possible/recommended usage with CBIOS4NET

CBU implementation of RSA and AES algorithms is not compatible with standard .NET crypto services. Full compatibility (including encryption key conversion) is provided in CBIOS4NET version 1.5 and up for CBU SC hardware based encryption.

1.8 Smarx4NET/CBIOS4NET list of samples

The following samples illustrating CBIOS4NET usage are currently available in SmarxOS PPK:

- Standard set of samples for CBIOS (including CBU SC specific features), DO and RU API:
  - **CBIOS4NETSmp**
  - **DO**
  - **RFP**
- CBIOS4NET advanced samples: adding professional protection to your applications

    - **EncyptedAssemblySample**
    - **LargeDataEncryption**
- Using Data Protection API (DP4NET) for media protection:
    - **DataProtection4MediaFiles**
- CBIOS4NET – CBIOS network administration sample:
    - **NetAdminSample**

All samples are written for MS Visual Studio 2005. Because of Framework.NET 3.x (3.0, 3.5) backward compatibility with version 2, all the above samples can be used with MSVS2008 too "as is" – only standard conversion is required. See included readme files for further details.

CBIOS4NET sample code:

- <Smarx PPK root>\SmarxOS\API\Win\Samples\CBIOS\C#\MSVS2005 (CBIOS4NET)\
- <Smarx PPK root>\SmarxOS\API\Win\Samples\DO\C#\MSVS2005 (CBIOS4NET)\
- <Smarx PPK root>\SmarxOS\API\Win\Samples\RFP\C#\MSVS2005 (CBIOS4NET)\
- <Smarx PPK root>\SmarxOS\API\Win\Samples\AssemblyProtection\C#\MSVS2005 (CBIOS4NET)\
- <Smarx PPK root>\SmarxOS\API\Win\Samples\LargeDataEncryption\C#\MSVS2005 (CBIOS4NET)\
- <Smarx PPK root>\SmarxOS\API\Win\Samples\DataProtection4MediaFiles\C#\MSVS2005 (CBIOS4NET)\

Smarx4NET sample code:

- <SmarxOS PPK root>\SmarxOS\API\Win\Samples\CBIOS\C#\MSVS2013 (SMARX4NET)\
- <SmarxOS PPK root>\SmarxOS\API\Win\Samples\CBIOS\C#\MSVS2013 (SMARX4NET for WinStore)\

2. CBIOS and DO API calls and classes-methods of the new CBIOS4NET component model

To start working with CBIOS4NET it is necessary to create an instance of **CryptoboxManager** class: **LocalCBManager** or **NetworkCBManager**. The **CryptoboxManager** is used to search for required box in the system (local computer/network). It provides the following information on found boxes: name, memory size, firmware version, Developer ID, list of available partitions, more. This information is provided as array of **BoxHandler** objects (a result of **ScanBoxes** method of **CryptoboxManager**).
The **BoxHandler** is used to open required box (**BoxHandle.Open()**). The local box manager (**LocalCBManager**) allows to activate box plug in/plug out notification – a static event (**CBIOSNotify**) is used for this purpose.
The **NetworkCBManager** works with boxes on remote server using methods: ScanNetwork (to search for servers) and Connect (to login to a remote server). If required box is known in advance it can be opened with the OpenBy methods of the proper CryptoboxManager.

The **Cryptobox** class operates with the CRYPTO-BOX itself. It allows reading/writing memory, performing encryption, changing passwords and more. Read and write operations are supported for active partition of the open CRYPTO-BOX unit. Use OpenBy methods of the CryptoboxManager or Open method of BoxHandler to open a unit. Every instance of Cryptobox class works in its own thread. It allows transparent simultaneous work with more than one CRYPTO-BOX.

In case of CBU SC the unit is actually opened by CryptoboxManager or BoxHandle. The **CBU2Cryptobox** class is instanced and casted to Cryptobox class. You can use BoxInfo.Type property to define box type.

The CBU2Cryptobox class defines CBU SC specific methods supporting CBU SC cryptographic API. With these methods you can:

- set encryption key/keypair value to any CBU SC encryption cell for key storage (RAM4/RAM5);
- lock specific key,
- read/write encryption key info -permissions for updating and/or using the key – not the value itself;
- use keys for encryption/decryption.

The above logic corresponds to both AES and RSA keys. In case of software RSA encryption there is a static CBU2RSA class containing methods for key generation and encryption/decryption.

Use of AES and RSA encryption is similar. RSA/AES-Key classes describe the key value; RSA/AES-KeyInfo classes specify key length and permissions are defined by CBU2KeyAccess enumeration. CBU2KeyAccess enumeration is used to define key permissions level for changing the key (SET_ flags) and encryption/decryption, obtaining information on this key - key strength and current access rights (USE_ flags).
The supported flags are:

- ACCESS_NEVER – this value (if being set) cannot be reset even by MARX distributor. MARX distributor can only reprogram the whole RAM4 zone of this CBU SC unit, so all current values of AES keys (if any) will be lost;
- ACCESS_ALWAYS – this value means free access (no limitations);
- ACCESS_UPW – UPW login is required;
- ACCESS_APW – APW login is required;
- ACCESS_LOCK – the access will be locked, only MARX distributor can unlock it.

Encrypt/Decrypt methods require also “mode” information. CBU2AESKeyMode specifies operations:

- OFB - encrypt/decrypt with OFB mode,
- CBC_ENCRYPT - encrypt with CBC mode
- CBC_DECRYPT - decrypt with CBC mode.

For CBU SC RSA mode you can specify which part of a key to use for encryption/decryption. Optionally you can also specify padding: either PKCS#1 compliant padding (RSAREF_PADDING) or CBU RSA padding (MARX_PADDING).

The RSA encryption for the CRYPTO-BOX XS/Versa (CBU) uses different padding rules comparing to other popular RSA implementations (e.g. OpenSSL, WinCrypt, etc.), which use PKCS#1 padding rules. Therefore these implementations are not compatible. For the CRYPTO-BOX SC (CBU SC) is is possible to specify PKCS#1 compliant padding to ensure compatibility with popular RSA implementations. MARX padding is supported by the CBIOS API and used in various MARX solutions and technologies, such as: WEB API, OLM and Remote Update.

To use CBU SC RSA keys with .NET RSA algorithm CBU2RSAKey class introduces GetPublicKey method and public constructor with RSAParameters param. Only public part of a key can be exported from CBU2RSAKey.

DataObject classes (one class per every DO type) provide DO API functionality for CBIOS4NET. Access to these classes is provided through the **Partition** class. This class (**Partition**) assumes a collection of **DataObject** instances associated with the **Partition** instance. When working with some DO instance the corresponding object must be included to the collection associated with this Partition, which in turn must be bound to some box (using method **Connect**). After binding the map to the open box (with the **Connect** method) handles are created for every DO and the program can work with them. The map of Data Objects can be saved or loaded with Save/Load methods of **Partition** class.

The diagram below illustrates CBIOS4NET classes implementing CBIOS and DO API functionality:

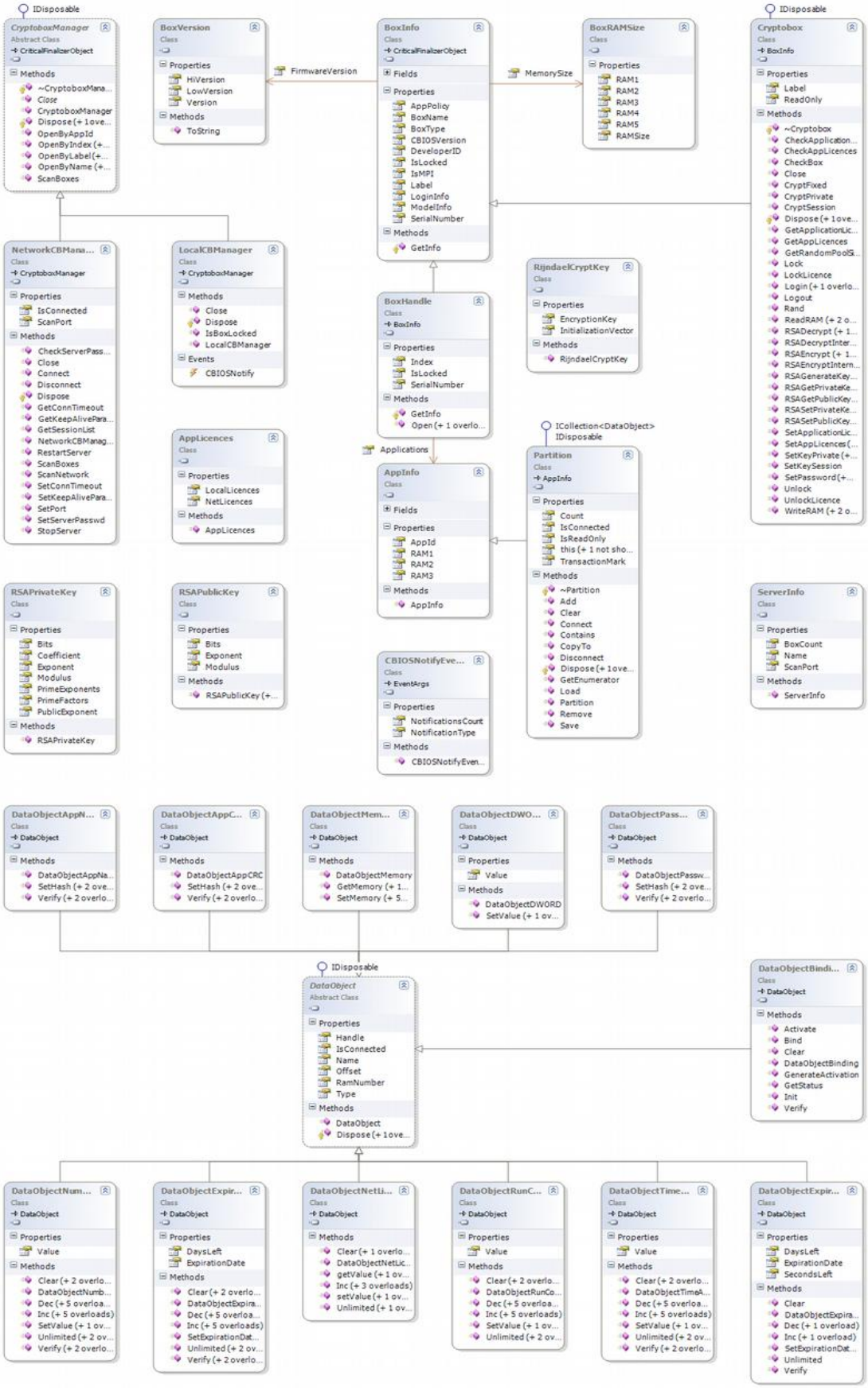


Fig. 2.1: CBIOS4NET Classes

The following diagram adds CBU SC specific classes (see description above):

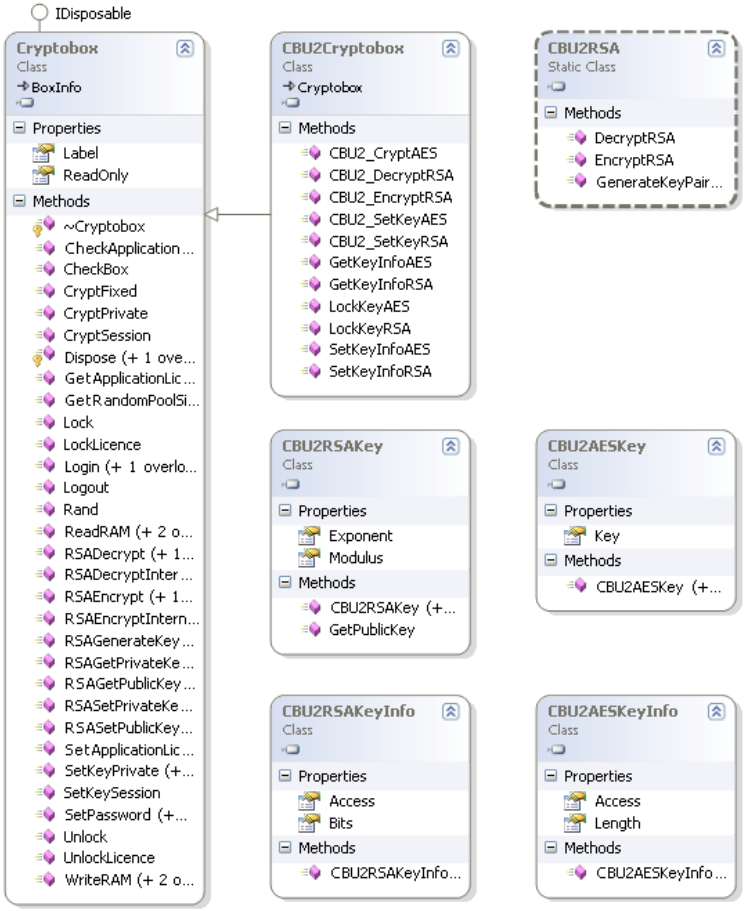


Fig. 2.2: CBIOS4NET Classes (CBU SC specific)

The following diagram adds CBU SC encryption API classes:

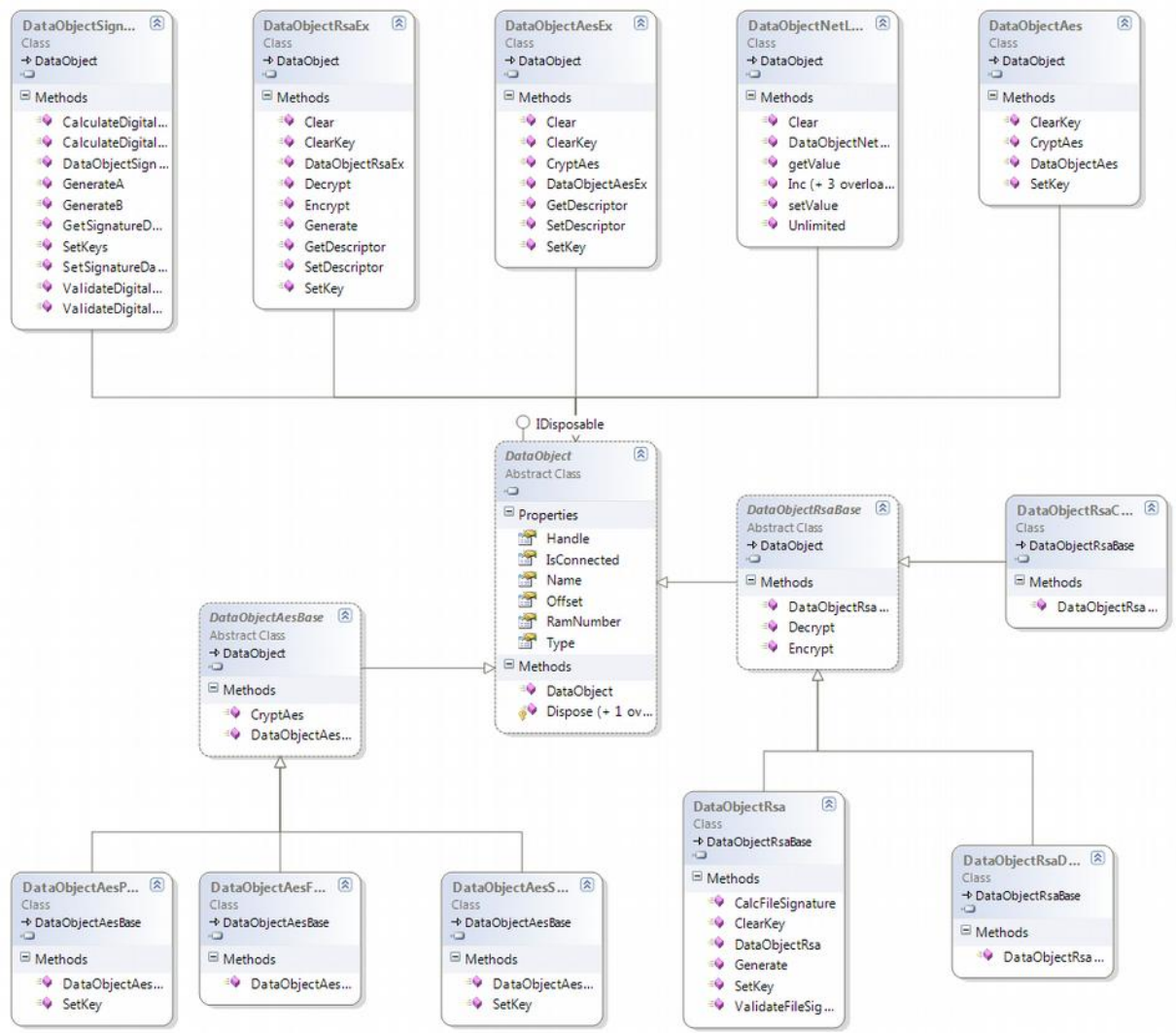


Fig. 2.3 CBIOS4NET Classes (Encryption API data objects)

3. RU API calls and classes-methods of the new RU4NET (CBIOS4NET.RFP) component model

Internal logic of RU4NET component model assumes three different logical steps - modes of remote update process:

- Request for Update creation and encryption (end-user's side);
- Request for Update decryption, processing and Activation Code preparation (distributor's side);
- Activation Code processing – updating license in end-user's CRYPTO-BOX (end-user's side).

The following classes are used for creation and encryption of requests for update:

Requests for Update are created with the **CreateRequest** method of **RequestBuilder** class. This method provides two ways of request creation:

- using values of corresponding instance properties for request parameter filling;
- explicit definition of main (RequestBaseParameters) and user-defined (RequestParameter) request parameters (static method).

The **RequestBaseParameters** class introduces predefined properties for SxAF product/project and user identification, such as: ProjectId, UserId, UserParameter. It is possible to specify which part of request's data does not require hardware based encryption. User specific parameters may contain any additional data important for further request interpretation (byte array), identified by unique ID – **Guid**. For **RequestParameter** class it is possible to specify if its data require encryption or not.

Here are main classes involved in the second step of remote update sequence - request handling at distributor's side (decryption, processing) and activation code preparation:

The **Request** class allows loading, decryption and analysis of request data at distributor's side. You can load request with the **Load** method or use a special constructor of the **Request** class for this purpose. After the request is loaded its non-encrypted data are available as values of class properties: **TransactionMark**, **UserId**, **ProjectId**, etc.
The index on the **Guid** property is used for obtaining user specific request parameters. The obtained data refer to MARX hardware originated the request; the **TranslateRequest** method allows to decrypt those data which require decryption.

The **ActivationSequenceBuilder** class is used for Activation Code creation. An instance of this class represents collection of **ActivationRecord** records. Each **ActivationRecord** record contains data on one of partitions included to the license update sequence. In turn **ActivationRecord** record includes collection of **ActivationRecordStep** instances. Each **ActivationRecordStep** specifies update of one licensing data object included to the partition related to the corresponding **ActivationRecord** instance.

The **ActivationRecordEx** class is used for partition creation, resizing during update process. It is derived from **ActivationRecord** and provides additional properties to customize partition update (resize, creation). Properties **CheckRAMXSize** are used to find a proper partition and **NewRAMXSize** properties set a new partition size. An update operation is set by **UpdateMode** property. **ActivationRecordUpdateMode** enumeration describes possible update operations:

- **Create mode** – creates a new partition in the box and initializing all its data objects with specified values

- **Delete mode** – deletes existing partition from the box. “Check” partition size should be specified.
- **Overwrite mode** – resizes and rewrites an existing partition, initializing all newly added data objects according to the new structure. If newly defined partition size has enough space to store old data it is saved, otherwise it is truncated. “Check” partition size should be specified.
- **Update Mode** (“classic” mode updating some data objects of existing partition). “Check” partition size should be specified.
- **Check_Fails** – can be applied to any update mode (default value). Rollbacks all update operations and returns error code in case of partition update error.
- **Check_Ignore** – If defined, update operation described in **ActivationRecordEx** is not applied in case of error, but the whole update process is continued with no error code returned. Can be applied to any update mode.

The **ActivationSequenceBuilder** has 2 properties for AES keys update **KeyPrivate**, **KeySession**. If specified (not null value) corresponding key will be set during update process. Note: In case of activation process fail these keys will be set. There is no rollback functionality for AES keys.

NOTE: Besides data objects stored directly in partition memory, the license may include so called “associated” data objects – which are stored in special system partitions. One of such associated data objects currently supported by CBIOS4NET is network licensing data object. It is represented with the **DataObjectNetLicence** class. The data of this object is stored separately in LCS partition. With current limitation only one connected instance of **DataObjectNetLicence** is allowed per application. In other words, the 1st net license object has to be disconnected (Partition.Remove or Partition.Disconnect method) in order to work with another one.

When required license update is ready (one or more instances of **ActivationRecord** class, each containing a set of **ActivationRecordStep** instances) the Activation Code can be created with the **MakeActivationCode** method of **ActivationSequenceBuilder** class.
The final step of Remote update sequence is covered with the **Activator.Execute()**.

The diagram below shows RU4NET classes implementing RU API functionality:

Fig. 3.1: RU4NET classes

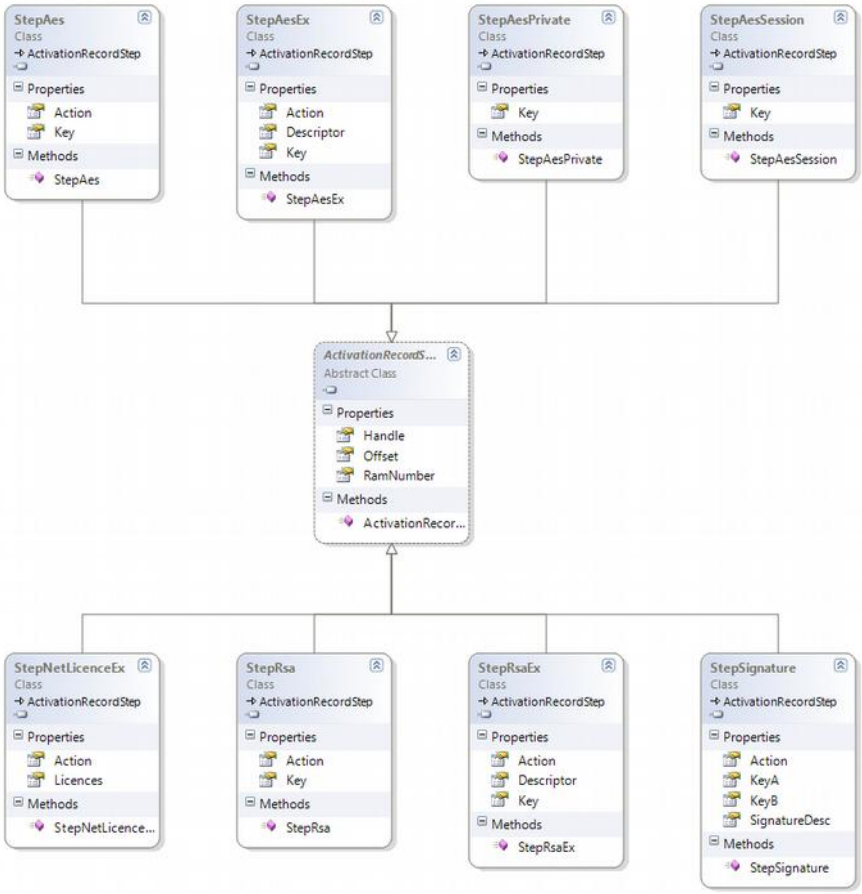


Fig. 3.2: RU4NET encryption step classes

4. Data Protection API calls and classes-methods of the new CBIOS4NET component model

The Data Protection (DP) API allows reliable protection of application critical data files. See SmarxOS Data Protection (Developer's Guide) for more information on the technology and DP API. This chapter focuses on DP4NET – DP API component based implementation included to CBIOS4NET.

The **DP** class allows creating encrypted file and signature file. For this purpose the class provides two static methods: **EncryptFile** and **GetSignature**. The encryption key is defined with an instance of the **DPEncryptionKey** class. Current Data Protection API implementation supports **AES 256** encryption algorithm.
The **DPProcess** class controls data filtering process (data decryption/encryption on the fly during read/write operations on protected files). In order to start filtering for protected file the **AddFilter** method of the **DPProcess** class is used. An instance of the **DPFilter** class defines protected file name and encryption key value to be used. After adding filter to the process the filter will be initialized (the **IsActive** property will become "true") and the process will be able to read decrypted on the fly data from the file. The filter can be used only by one process.
The **DPFilter** class allows to suspend/resume/stop for active filter - Suspend/Resume/Stop methods.
The **DPProcess** class instance allows controlling all filters attached to the process – similar actions (ResumeFiltering/SuspendFiltering/StopFiltering) but for all process related filters.

The diagram below shows DP4NET classes implementing DP API functionality:

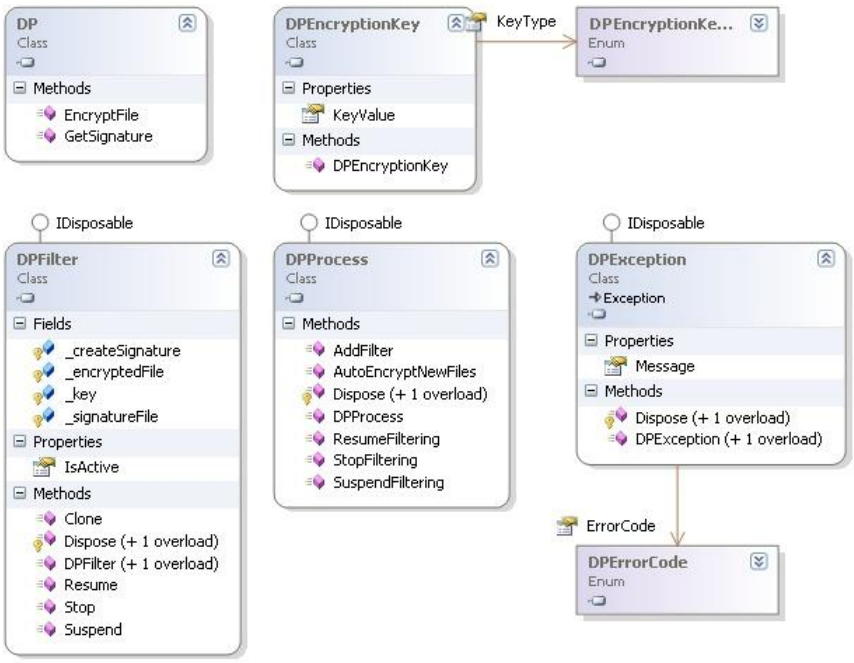


Fig. 4.1: DP4NET classes

5. CBIOS4NET and RU4NET: Examples of usage

The following series of code examples illustrate typical scenarios of CBIOS4NET and RU4NET usage; the syntax is native for C#.NET environment.

5.1 Sample 1: Communicating with local CRYPTO-BOX®, generating random sequence

This sample searches for local CRYPTO-BOX with specified DeveloperID. If found this box is used for random sequence generation:

```
static void sample1()
{
	int myBoxDeveloperId = 262704;
	ushort myAppId = 999;

	LocalCBManager manager = new LocalCBManager();
	BoxHandle[] boxes = manager.ScanBoxes();
	foreach (BoxHandle handle in boxes) {
		if (handle.DeveloperID == myBoxDeveloperId) {
			Cryptobox box = handle.Open(myAppId);
			byte[] randomPool = box.Rand(10);
			Console.WriteLine("random array:");
			foreach(byte num in randomPool)
			Console.Write(" {0},", num);
		}
	}
}
```

5.2 Sample 2: Networking, using hardware based Rijndael encryption

This sample illustrates hardware based Rijndael encryption of network CRYPTO-BOX (using Rijndael Fixed Key) with AppID 999 (containing partition 999). The sample searches the local network for CBIOS Server with port=2000. If found it tries to open CRYPTO-BOX with partition 999. The Fixed Rijndael Key of this CRYPTO-BOX is used to encrypt a sample string:

```
static void sample2()
{
	ushort myAppId = 999;

	NetworkCBManager manager = new NetworkCBManager();
	ServerInfo[] servers = manager.ScanNetwork(2000); // port = 2000
	foreach (ServerInfo server in servers) {
		if (server.BoxCount > 0) {
			manager.Connect(server.Name, server.ScanPort);

			Cryptobox box = manager.OpenByAppId(myAppId);
			if (box.Login(CryptoboxLoginType.Admin, "admin")) {
				byte[] encryptedStringBytes = box.CryptFixed(
					System.Text.Encoding.ASCII.GetBytes("Sample text."));
				String encryptedString =
					System.Text.Encoding.ASCII.GetString(
						encryptedStringBytes);
			Console.WriteLine("\nEncrypted string:");
			Console.WriteLine(encryptedString);
			box.Logout();
```



```
                        }
                }
        }
}
```

5.3 Sample 3: Manipulations with Data Objects

This sample illustrates handling of Data Objects. It creates a map of data objects, containing the DataObjectAppCRC data object, then opens the 1st CRYPTO-BOX on the local computer with demo partition (App ID 999). Then the map is bound to the open partition and the sample checks if application checksum value is equal to those one stored in the CRYPTO-BOX.

```
static void sample3()
{
        Ushort myAppId = 999;
        Uint         myAppCRCHandle = 1001;

        Partition doMap = new Partition();
        DataObjectAppCRC appCRC = new DataObjectAppCRC(myAppCRCHandle, RAM.RAM1, 0);
        doMap.Add(appCRC);

        LocalCBManager manager = new LocalCBManager();
        Cryptobox box = manager.OpenByIndex(1, myAppId);
        if (doMap.Connect(box)) {
                if(!box.Login(CryptoboxLoginType.User, "demo"))
                        return;
                if (!appCRC.Verify())
                        Console.WriteLine("Application's executable is modified!");
        }

}
```

5.4 Sample 4: RU4NET

The following large sample covers all three steps of Remote Update scenario:

- Request for Update creation (client's side);
- Request for Update processing and Activation Code generation (distributor's side);
- Activation Code processing (client's side).

Plus it initializes the CRYPTO-BOX with required data.

```
class Program
        {
          static Guid guid = new Guid("2453907A-2E98-422b-B051-631F3EF0E975");
          static String REQUEST_FILE_NAME = "rfp_request.bin";
          static String ACTCODE_FILE_NAME = "rfp_actcode.bin";
          static String KEY_USERPUBL_FILE_NAME = "cl_pub.bin";
          static String KEY_DISTRPRIV_FILE_NAME = "dist_prv.bin";
          static ushort wAppId = 999;

          static uint TEST_EXPIRATION_DATE = 1001;
          static uint TEST_COUNTER = 1002;
          static uint TEST_DOUBLE_WORD = 1003;
          static uint TEST_PASSWORD = 1005;
          static uint TEST_MEMORY = 1006;
          static uint TEST_NET_LICENCE = 1004;

          static uint TEST_EXPIRATION_DATE_OFFSET = 0;
          static uint TEST_COUNTER_OFFSET = 4;
```



```
static uint TEST_DOUBLE_WORD_OFFSET = 8;
static uint TEST_PASSWORD_OFFSET = 12;
static uint TEST_MEMORY_OFFSET = 16;

class DataObjectsVal
{
    LocalCBManager manager;
    Cryptobox box;
    Partition map;
    DataObjectExpirationDate _doDate;
    DataObjectRunCounter _doCounter;
    DataObjectDWORD _doDWORD;
    DataObjectPasswordHash _doPassword;
    DataObjectMemory _doMemory;
    DataObjectNetLicence _doLic;

    public DataObjectsVal()
    {
        manager = new LocalCBManager();
        box = manager.OpenByAppId(wAppId);
        map = new Partition();

        _doDate = new DataObjectExpirationDate(TEST_EXPIRATION_DATE,
                                RAM.RAM1, TEST_EXPIRATION_DATE_OFFSET);
        _doCounter = new DataObjectRunCounter(TEST_COUNTER, RAM.RAM1,
                                TEST_COUNTER_OFFSET);
        _doDWORD = new DataObjectDWORD(TEST_DOUBLE_WORD, RAM.RAM1,
                                TEST_DOUBLE_WORD_OFFSET);
        _doPassword = new DataObjectPasswordHash(TEST_PASSWORD, RAM.RAM1,
                                TEST_PASSWORD_OFFSET);
        _doMemory = new DataObjectMemory(TEST_MEMORY, RAM.RAM1,
                                TEST_MEMORY_OFFSET);
        _doLic = new DataObjectNetLicence(TEST_NET_LICENCE, RAM.RAM1, 0);

        map.Add(_doDate);
        map.Add(_doCounter);
        map.Add(_doDWORD);
        map.Add(_doPassword);
        map.Add(_doMemory);
        map.Add(_doLic);
    }

    public void doSaveValues()
    {
        box.Login(CryptoboxLoginType.User, "demo");
        map.Connect(box);
        _doDate.ExpirationDate = new DateTime(2009, 3, 1);
        _doCounter.Value = 99;
        _doDWORD.Value = 0xAABBCCDD;
        _doPassword.SetHash("old");
        byte[] mem = new byte[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        _doMemory.SetMemory(mem);
        _doLic.setValue(5, "admin");
        map.Disconnect();
        box.Logout();
    }

    public void doGetValues()
    {
        box.Login(CryptoboxLoginType.User, "demo");
        map.Connect(box);
        Console.WriteLine("Expiration date: {0}",
                    _doDate.ExpirationDate.ToString("dd.MM.yyyy"));
        Console.WriteLine("Run counter: {0}", _doCounter.Value);
        Console.WriteLine("DWORD: {0:x}", _doDWORD.Value);
        Console.WriteLine("NetLic: {0}", _doLic.getValue("admin"));
```



```
            Console.WriteLine("Password verify \"old\":
                      {0}",_doPassword.Verify("old")?"Y":"N");
            Console.WriteLine("Password verify \"new\": {0}",
                      _doPassword.Verify("new") ? "Y" : "N");

            byte[] mem = _doMemory.GetMemory(10);
            Console.Write("Memory: ");
            for (int i = 0; i < 10; i++)
                Console.Write("{0} ", mem[i]);
            Console.WriteLine();
            map.Disconnect();
            box.Logout();
        }

        public BoxInfo getBoxInfo()
        {
            return (BoxInfo)box;
        }
    }

    static void CreateRequest(DataObjectsVal doVals)
    {
        RequestBuilder request = new RequestBuilder();
        request.UserId = 200;
        request.NotEncryptedMask = RequestEncryptionMask.REQUEST_TRANS_MARK;

        byte[] value = System.Text.ASCIIEncoding.ASCII.GetBytes(
                                    "Please, update my box till Sunday!");
        RequestParameter param = new RequestParameter(guid, value, true);
        request.Add(param);

        byte[] data = request.MakeRequest(doVals.getBoxInfo(), "demo");

        WriteFile(REQUEST_FILE_NAME, data);
    }
    static void WriteFile(String fileName, byte[] data)
    {
         File.WriteAllBytes(fileName, data);
    }
    static byte[] ReadFile(String fileName)
    {
         return File.ReadAllBytes(fileName);
    }
    static ActivationRecord getRecord()
    {
        ActivationRecord testRecord = new ActivationRecord();
        testRecord.AppId = wAppId;

        StepExpirationDate step1 = new StepExpirationDate();
        step1.Action = ActionType.DO_INC;
        step1.Handle = TEST_EXPIRATION_DATE;
        step1.RamNumber = RAM.RAM1;
        step1.Offset = TEST_EXPIRATION_DATE_OFFSET;

        StepRunCounter step2 = new StepRunCounter();
        step2.Action = ActionType.DO_INC;
        step2.Value = 2000;
        step2.Handle = TEST_COUNTER;
        step2.RamNumber = RAM.RAM1;
        step2.Offset = TEST_COUNTER_OFFSET;

        StepDWORD step3 = new StepDWORD();
        Random r = new Random();
        step3.Value = (uint)r.Next();
        step3.Handle = TEST_DOUBLE_WORD;
        step3.RamNumber = RAM.RAM1;
```



```
        step3.Offset = TEST_DOUBLE_WORD_OFFSET;

        StepMemory step4 = new StepMemory();
        step4.SetMemory(new byte[] { 0xff, 0xfe, 0xfd, 0xfc, 0xfb, 0xfa, 0xf9,
                                     0xf9, 0xf7, 0xf6 });
        step4.Handle = TEST_MEMORY;
        step4.RamNumber = RAM.RAM1;
        step4.Offset = TEST_MEMORY_OFFSET;

        StepPasswordHash step5 = new StepPasswordHash();
        step5.Action = ActionType.DO_SET;
        step5.Value = "new";
        step5.Handle = TEST_PASSWORD;
        step5.RamNumber = RAM.RAM1;
        step5.Offset = TEST_PASSWORD_OFFSET;

        StepNetLicence step6 = new StepNetLicence();
        step6.Action = ActionType.DO_SET;
        step6.Licences = (uint)r.Next();
        step6.Handle = TEST_NET_LICENCE;
        step6.RamNumber = RAM.RAM1;

        testRecord.Add(step1);
        testRecord.Add(step2);
        testRecord.Add(step3);
        testRecord.Add(step4);
        testRecord.Add(step5);
        testRecord.Add(step6);
        return testRecord;
    }

    static void TranslateRequestandGenerateActCode()
    {
        byte[] data = ReadFile(REQUEST_FILE_NAME);
        byte[] userPublKey = ReadFile(KEY_USERPUBL_FILE_NAME);
        byte[] distrPrivKey = ReadFile(KEY_DISTRPRIV_FILE_NAME);


        DecryptKeys keys = new DecryptKeys(userPublKey, distrPrivKey);

        Request request = new Request(data);//keys);
        if ((request.NotEncryptedMask & RequestEncryptionMask.REQUEST_USER_ID)
                    == RequestEncryptionMask.REQUEST_USER_ID)
            keys.RSADistribPrivateKey.ToString();
        RequestParameter param;
        try
        {
            param = request[guid];
        }
        catch (Exception e)
        {
        }
        request.Translate(keys);

        param = request[guid];


        ActivationSequenceBuilder activation = new ActivationSequenceBuilder();
        activation.Add(getRecord());
        byte[] actCode = activation.MakeActivationCode(
                                    request.TransactionMark.Value, keys);
        WriteFile(ACTCODE_FILE_NAME, actCode);
    }
    static void ProcessActCode(DataObjectsVal doVal)
    {
        byte[] actCode = ReadFile(ACTCODE_FILE_NAME);
```



```
        CBIOS4NET.RFP.Activator.Execute(actCode, doVal.getBoxInfo(), "demo",
                                                    "admin");
    }

  static void Main ( string[] args ) // main entry
         {
        // Step#0: preparing the box
        DataObjectsVal doVals = new DataObjectsVal();
        doVals.doSaveValues();
        Console.WriteLine("Old values");
        doVals.doGetValues();

         // Step#1: creating Request for Update

        CreateRequest(doVals);

         // Step#2: process Request for Update and generate Activation Code

        TranslateRequestandfGenerateActCode();

         // Step#3: process Activation Code

        ProcessActCode(doVals);
        Console.WriteLine();
        Console.WriteLine("New values");

        doVals.doGetValues();

        Console.ReadKey();

         }
}
```


6. CBIOS4NET: Description of classes

6.1 AesExDescriptor struct

Descriptor of AesEx DO

Old syntax: struct DO_AES_EX_DATA

Public Properties:

Name	Description
uint keyIndex	Key index in Aes cell
AesExKeyLocation location	Location description (Aes key cell, private, fixed etc.)
CBU2AESKeyMode mode	Mode of Aes encryption
uint reserved	Reserved value
uint updateDescr	version/update rule

6.2 AppInfo class

This class contains application memory allocation information.
Old syntax: struct CBIOS_APP_INFO

Public Properties:

Name	Description
ushort AppId	Application ID. This property is readonly.
ushort RAM1	Application RAM1 size. This property is readonly.
ushort RAM2	Application RAM2 size. This property is readonly.
ushort RAM3	Application RAM3 size. This property is readonly.

6.3 AppLicences class

This class represents application's licenses amount value stored in LMT.

Public Properties:

Name	Description
uint LocalLicences	Local licenses amount.
uint NetLicences	Network licenses amount.

Public Methods:

Name	Description
AppLicences(uint localLicences, uint netLicences)	Object's constructor **Input Parameters:** *localLicences* – local licenses value; *netLicences* – network licenses value.

6.4 BoxHandle class

Extends: BoxInfo
This class represents non-opened CRYPTO-BOX.

Public Properties:

Name	Description
AppInfo[] Applications	Array of applications contained in CRYPTO-BOX. This property is readonly.
int Index { get; }	Physical index of CRYPTO-BOX. This property is readonly.

Public Methods:

Name	Description
Cryptobox Open()	Open CRYPTO-BOX for read/write. **Old syntax:** CBIOS_OpenByIndex
Cryptobox Open(ushort wAppID)	Open CRYPTO-BOX with selected Application ID for read/write. **Input Parameters:** *wAppID* – Application/partition identificator. **Old syntax:** CBIOS_OpenByIndex

6.5 BoxInfo class

This class contains information about the CRYPTO-BOX.
Old syntax: struct CBIOS_BOX_INFO

Public Properties:

Name	Description
bool AppPolicy	"false" means demo CRYPTO-BOX or digital signature is not valid. This property is readonly. **Old syntax:** bAppPolicy in CBIOS_BOX_INFO_ADV structure
int BoxName	Represents BoxName. This property is readonly. **Old syntax:** dwBoxName in CBIOS_BOX_INFO_ADV structure
BoxTypeInfo BoxType	Returns a BoxTypeInfo enumeration - CRYPTO-BOX type: unknown, CBU or CBU SC.
ushort CBIOSVersion	CRYPTO-BOX CBIOS version info - "0" means: not a SmarxOS formatted CRYPTO-BOX. This property is readonly. **Old syntax:** wCBIOS in CBIOS_BOX_INFO_ADV structure
int DeveloperID	DeveloperID (unique value for every MARX customer) This property is readonly. **Old syntax:** dwDeveloperID in CBIOS_BOX_INFO_ADV structure

Name	Description
CBIOS4NET.BoxVersion FirmwareVersion	Firware version. This property is readonly. **Old syntax:** bLoVersion and bHiVersion in CBIOS_BOX_INFO_ADV structure
bool IsMPI	CRYPTO-BOX is MPI formatted. This property is readonly. **Old syntax:** bMPI in CBIOS_BOX_INFO_ADV structure
byte[] Label	CRYPTO-BOX Label. This property is readonly. **Old syntax:** bBoxLabel in CBIOS_BOX_INFO_ADV structure
BoxLoginInfo LoginInfo	CRYPTO-BOX current status: Admin or user login. This property is readonly. **Old syntax:** bLogin in CBIOS_BOX_INFO_ADV structure
BoxRAMSize MemorySize	CRYPTO-BOX RAM size. This property is readonly. See BoxRAMSize class for details. **Old syntax:** dwRAMLen and dwSize_RAMx in CBIOS_BOX_INFO_ADV structure
BoxModelInfo ModelInfo	CRYPTO-BOX Model. This property is readonly. **Old syntax:** dwModel in CBIOS_BOX_INFO_ADV structure
byte[] SerialNumber	CRYPTO-BOX serial number. This property is readonly. **Old syntax:** CBIOS_GetSerialNum

6.6 BoxRAMSize class

This class represents memory size of CRYPTO-BOX.

Public Properties:

Name	Description
int RAM1	Memory size allocated for RAM1. This property is readonly.
int RAM2	Memory size allocated for RAM2. This property is readonly.
int RAM3	Memory size allocated for RAM3. This property is readonly.
int RAM4	Memory size allocated for RAM4. This property is readonly.
int RAM5	Memory size allocated for RAM5. This property is readonly.
int RAMSize	CRYPTO-BOX memory size. This property is readonly.

6.7 BoxVersion class

This class represents CRYPTO-BOX firmware version.

Public Properties:

Name	Description
int HiVersion	Firmware version High (1 for 1.6). This property is readonly.
int LowVersion	Firmware version Low (6 for 1.6). This property is readonly.
string Version	Firmware string representation (1.6). This property is readonly.

6.8 CBU2AESKey class

This class represents AES key for CBU SC.

Public Properties:

Name	Description
byte[ ] Key	Key value. (16 bytes supported)

Public Methods:

Name	Description
CBU2AESKey()	Objects constructor
CBU2AESKey(byte[ ] key)	Objects constructor **Input Parameters:** *key* – key value.

6.9 CBU2AESKeyInfo class

This class represents AES key info.

Public Properties:

Name	Description
CBU2KeyAccess Access	Key access parameters. Set and use rights. See CBU2KeyAccess enumeration
uint Length	Key length (16 bytes supported??)

Public Methods:

Name	Description
CBU2AESKeyInfo()	Objects constructor
CBU2AESKeyInfo(uint length, CBU2KeyAccess access)	Objects constructor **Input Parameters:** *length* – key length; access – key access parameters

6.10 CBU2Cryptobox class

Extends: Cryptobox
This class is used to call CBU SC methods.
This class is instanced by CryptoboxManager in case if CBU SC unit is opened and boxes it to a Cryptobox class. You should use a cast to CBU2Cryptobox in order to get access to CBU SC methods.

Public Methods:

Name	Description
byte[ ] CBU2_CryptAES(uint dwKeyIndex, CBU2AESKeyMode mode, byte[ ] IV, byte[ ] data)	Encrypts/decrypts data with specified AES key **Returns:** encrypted/decrypted data. **Input Parameters:** *dwKeyIndex* – index of a stored key to use; mode – mode of a key; IV – initialization vector for AES key; data – input data.
byte[ ] CBU2_DecryptRSA(uint dwKeyIndex, CBU2RSAKeyMode mode, byte[ ] data)	Decrypts data with specified RSA key **Returns:** decrypted data. **Input Parameters:** *dwKeyIndex* – index of a stored key to use; mode – mode of a key; data – input data.
byte[ ] CBU2_EncryptRSA(uint dwKeyIndex, CBU2RSAKeyMode mode, byte[ ] data)	Encrypts data with specified RSA key **Returns:** encrypted data. **Input Parameters:** *dwKeyIndex* – index of a stored key to use; mode – mode of a key; data – input data.
void CBU2_SetKeyAES(uint dwKeyIndex, CBU2AESKey pAESKey, CBU2AESKeyInfo pAESKeyInfo)	Used to set AES key to box storage. **Input Parameters:** *dwKeyIndex* – index of a key; pAESKey – key to set; pAESKeyInfo – corresponding key info.
void CBU2_SetKeyRSA(uint dwKeyIndex, CBU2RSAKey pRSAKeyPair, CBU2RSAKeyInfo pRSAKeyInfo)	Used to set RSA key to box storage. **Input Parameters:** *dwKeyIndex* – index of a key; pRSAKeyPair – key to set; pRSAKeyInfo – corresponding key info.
CBU2AESKeyInfo GetKeyInfoAES(uint dwKeyIndex)	Gets key info for a stored AES key **Returns:** key info. **Input Parameters:** *dwKeyIndex* – index of a stored key.
CBU2RSAKeyInfo GetKeyInfoRSA(uint dwKeyIndex)	Gets key info for a stored RSA key **Returns:** key info. **Input Parameters:** *dwKeyIndex* – index of a stored key.
void LockKeyAES(uint dwKeyIndex)	Locks specified AES key **Input Parameters:** *dwKeyIndex* – index of a stored key.
void LockKeyRSA(uint dwKeyIndex)	Locks specified RSA key **Input Parameters:** *dwKeyIndex* – index of a stored key.

Name	Description
void SetKeyInfoAES(uint dwKeyIndex, CBU2AESKeyInfo pAESKeyInfo)	Sets key info for a stored AES key **Input Parameters:** *dwKeyIndex* – index of a stored key; *pAESKeyInfo* – key info value
void SetKeyInfoRSA(uint dwKeyIndex, CBU2RSAKeyInfo pRSAKeyInfo)	Sets key info for a stored RSA key **Input Parameters:** *dwKeyIndex* – index of a stored key; *pAESKeyInfo* – key info value

6.11 CBU2RSA class

Static class for software based RSA encryption.

Public Methods:

Name	Description
static byte[ ] DecryptRSA(CBU2RSAKey pRSAKey, CBU2RSAKeyMode mode, byte[ ] data)	Decrypts data with specified RSA key **Returns:** Decrypted data. **Input Parameters:** *pRSAKey* – RSA key value; *mode* – mode of a key; *data* – input data.
static byte[ ] EncryptRSA(CBU2RSAKey pRSAKey, CBU2RSAKeyMode mode, byte[ ] data)	Encrypts data with specified RSA key **Returns:** encrypted data. **Input Parameters:** *pRSAKey* – RSA key value; *mode* – mode of a key; *data* – input data.
static void GenerateKeyPairRSA(out CBU2RSAKey pRSAKeyPair, out CBU2RSAKey pRSAPublicKey, ushort bits, byte[ ] randomPool)	Generates RSA key pair **Input Parameters:** *pRSAKeyPair* – key pair value (out parameter); *pRSAPublicKey* – public part of generated key (out parameter); *bits* – key length; *randomPool* – random data used for key generation.

6.12 CBU2RSAKey class

This class represent RSA key used with CBU SC.

Public Properties:

Name	Description
byte[ ] Exponent	Exponent value.
byte[ ] Modulus	Keys modulus value.

Public Methods:

Name	Description
CBU2RSAKey(byte[ ] modulus, byte[ ] exponent)	Objects constructor. **Input Parameters:** *modulus* – modulus value; *exponent* – exponent value.
CBU2RSAKey(RSAParameters RSAKeys)	Objects constructor. Used to initialize CBU SC RSA key from .NET RSA cryptographic services key. **Input Parameters:** *RSAKeys* – key value.
RSAParameters GetPublicKey()	Method returns public part of RSA key that can be used in .NET RSA cryptographic services. **Returns:** key value.

6.13 CBU2RSAKeyInfo class

This class represents RSA key info.

Public Properties:

Name	Description
CBU2KeyAccess Access	Key access parameters. Set and use rights. See CBU2KeyAccess enumeration
uint Bits	Key length

Public Methods:

Name	Description
CBU2RSAKeyInfo()	Objects constructor
CBU2RSAKeyInfo(uint bits, CBU2KeyAccess access)	Objects constructor **Input Parameters:** *bits* – key length; *access* – key access parameters

6.14 Cryptobox class

Extends: BoxInfo, IDisposable
This class represents Cryptobox.

Public Properties:

Name	Description
byte[] Label	CRYPTO-BOX label. Label is 8 bytes long array. CBIOSException will be thrown in case of setting longer label.
bool ReadOnly	“true” if data can be written to a box’s partition. Box should be opened with ApplicetionID specified to enable write access. This property is readonly.

Public Methods:

Name	Description
AppLicences CheckApplicationLicences()	Retrieves information about the free (not busy) local and network licenses of an application. **Returns:** Application licenses amount. **Old syntax:** CBIOS_CheckAppLicence deprecated methods use CheckAppLicences
RuleAppLicences CheckAppLicences()	Retrieves information about the free (not busy) local and network licenses of an application. **Returns:** Application licenses amount.
bool CheckBox()	Checks for the presence of currently opened CRYPTO-BOX. **Returns:** "true" if CRYPTO-BOX is attached. **Old syntax:** CBIOS_CheckBox
byte[] CryptFixed(byte[] data)	Encrypts/decrypts data using the internal hardware algorithm using with Key and fixed Initialization Vector. **Input Parameters:** *data* – binary data to be encrypted/decrypted. **Returns:** Encrypted/decrypted data **Old syntax:** CBIOS_CryptFixed
byte[] CryptPrivate(byte[] data)	Encrypts/decrypts data using internal hardware algorithm with private Key and private Initialization Vector. **Input Parameters:** *data* – binary data to be encrypted/decrypted. **Returns:** Encrypted/decrypted data **Old syntax:** CBIOS_CryptPrivate
byte[] CryptSession(byte[] data)	Encrypts/decrypts data using internal hardware algorithm with Session Key and Session Initialization Vector. **Returns:** Encrypted/decrypted data **Old syntax:** CBIOS_CryptSession
AppLicences GetApplicationLicences()	Reads information on application's local and network licenses from the License Management Table (LMT). **Returns:** Number of application licenses. **Old syntax:** CBIOS_GetAppLicences method is deprecated use GetAppLicences
RuleAppLicences GetAppLicences()	Reads information on application's local and network licenses from the LMT (taking into account extended network sharing rule). **Returns:** Application licenses amount.
static uint GetRandomPoolSize(uint keyBits)	Function helps to calculate random pool size used for generating RSA key pair. See RSAGenerateKeyPair method. **Input Parameters:** *keyBits* – RSA key's bits length. **Returns:** random pool length. **Old syntax:** CBIOS_RAND_POOL_SIZE makros

Name	Description
bool Lock()	Locks the CRYPTO-BOX. Other threads/processes will wait in the queue for 20 seconds and will throw CBIOSException, if the CRYPTO-BOX is not unlocked before timeout is reached. To unlock the CRYPTO-BOX call Unlock() method. **Returns:** "true" on success. **Old syntax:** CBIOS_LockBox
void LockLicence()	Gives exclusive access to the open partition, locking it from any other applications. **Old syntax:** CBIOS_LockLicence
bool Login(CBIOS4NET.CryptoboxLoginType Type, string password)	Login to the opened CRYPTO-BOX in User/Admin modes **Input Parameters:** *Type* – login type(user/admin) defined in CryptoboxLoginType enumeration; *Password* – login password. **Returns:** "true" on success. **Old syntax:** CBIOS_LoginUPW(), CBIOS_LoginAPW()
bool Login(CBIOS4NET.CryptoboxLoginType Type, byte[ ] password)	Login to the CRYPTO-BOX opened in User/Admin modes **Input Parameters:** *Type* – login type(user/admin) defined in CryptoboxLoginType enumeration; *Password* – login password. **Returns:** "true" on success. **Old syntax:** CBIOS_LoginUPW(), CBIOS_LoginAPW()
void Logout()	Performs logout **Old syntax:** CBIOS_Logout
byte[] Rand(uint length)	Retrieves hardware random bit stream. **Input Parameters:** *length* – desired length of random data pool array. **Returns**: Random data pool **Old syntax:** CBIOS_GetHWRand
byte[] ReadRAM(CBIOS4NET.RAM ram, int offset, int size)	Reads specified number of bytes from partition's ram. Null password assumed. **Input Parameters:** *ram* – partition's ram to be read defined in RAM enumeration; *offset* – requested data offset; *size* – number of bytes to be read. **Returns**: Data pool read. **Old syntax:** CBIOS_ReadRAMx
byte[] ReadRAM(CBIOS4NET.RAM ram, int offset, int size, string password)	Reads specified number of bytes from partition's ram. **Input Parameters:** *ram* – partition's ram to be read defined in RAM enumeration; *offset* – requested data offset; *size* – number of bytes to be read; *password* – user/admin password. **Returns**: Data pool read. **Old syntax:** CBIOS_ReadRAMx

Name	Description
byte[] RSADecrypt(CBIOS4NET.RAM keyMemory, uint keyOffset, byte[] data, string password)	Decrypts data using the RSA algorithm. **Input Parameters:** *keyMemory* – partition's ram where RSA key is stored; *offset* – RSA key offset; *data* – data to be decrypted; *password* – user/admin password. **Returns**: Decrypted data. **Old syntax:** CBIOS_DecryptRSA
byte[] RSADecryptInternal(CBIOS4NET.RSAKeyType type, byte[] data, string password)	Decrypts data using the Smarx OS internal RSA (InternalRSA_DISTR_PUBL or InternalRSA_CLIENT_PRIV) key (for secure communication). **Input Parameters:** *type* – key type. Either distributor's public or client private key; *data* – data to be decrypted; *password* – user/admin password. **Returns**: Decrypted data. **Old syntax:** CBIOS_ DecryptInternalRSA
byte[] RSAEncrypt(CBIOS4NET.RAM keyMemory, uint keyOffset, byte[] data, string password)	Encrypts data using the RSA algorithm. **Input Parameters:** *keyMemory* – partition's ram where RSA key is stored; *offset* – RSA key offset; *data* – data to be encrypted; *password* – user/admin password. **Returns**: Encrypted data. **Old syntax:** CBIOS_EncryptRSA
byte[] RSAEncryptInternal(CBIOS4NET.RSAKeyType type, byte[] data, string password)	Decrypts data using the Smarx OS internal RSA (InternalRSA_DISTR_PUBL or InternalRSA_CLIENT_PRIV) key (for secure communication). **Input Parameters:** *type* – key type. Either distributor's public or client private key; *data* – data to be encrypted; *password* – user/admin password. **Returns**: Encrypted data. **Old syntax:** CBIOS_ EncryptInternalRSA
void RSAGenerateKeyPair(CBIOS4NET.RAM publicKeyMemory, uint publicKeyOffset, CBIOS4NET.RAM privateKeyMemory, uint privateKeyOffset, ushort bits, byte[] randomPool, string password)	Generates the private and the public RSA keys and stores them in corresponding RAM. **Input Parameters:** *publicKeyMemory* – partition's RAM to store public key; *publicKeyOffset* – public key offset; *privateKeyMemory* – partition's RAM to store private key; *privateKeyOffset* – private key offset; *bits* – desired key length; *randomPool* – some random data; *password* – user/admin password. **Old syntax:** CBIOS_GenerateKeyPairRSA

Name	Description
RSAPrivateKey RSAGetPrivateKey(CBIOS4NET.RAM keyMemory, uint keyOffset, string password)	Retrieves the private RSA Key stored in RAM for software RSA encryption. See RSAPrivateKey class for details. **Input Parameters:** *keyMemory* – partition's RAM where private key is stored; *keyOffset* – private key offset; *password* – user/admin password. **Returns**: Private RSA key. **Old syntax:** CBIOS_GetKeyPrivateRSA
RSAPublicKey RSAGetPublicKey(CBIOS4NET.RAM keyMemory, uint keyOffset, string password)	Retrieves the public RSA Key stored in RAM for software RSA encryption. See RSAPublicKey class for details. **Input Parameters:** *keyMemory* – partition's RAM where public key is stored; *keyOffset* – public key offset; *password* – user/admin password. **Returns**: Public RSA key. **Old syntax:** CBIOS_ GetKeyPublicRSA
void RSASetPrivateKey(CBIOS4NET.RAM keyMemory, uint keyOffset, CBIOS4NET.RSAPrivateKey key, string password)	Stores the private RSA key in RAM for software RSA encryption. **Input Parameters:** *keyMemory* – partition's RAM to store private key; *keyOffset* – private key offset; *key* – private key. See RSAPrivateKey class; *password* – user/admin password. **Old syntax:** CBIOS_ SetKeyPrivateRSA
void RSASetPublicKey(CBIOS4NET.RAM keyMemory, uint keyOffset, CBIOS4NET.RSAPublicKey key, string password)	Stores the public RSA Key in RAM for software RSA encryption. **Input Parameters:** *keyMemory* – partition's RAM to store public key; *keyOffset* – public key offset; *key* – public key. See RSAPublicKey class; *password* – user/admin password. **Old syntax:** CBIOS_ SetKeyPublicRSA
void SetApplicationLicences(CBIOS4NET.AppLicences licenses, string password)	Writes information on application's local and network licenses to the LMT. **Input Parameters:** *licenses* – number of licenses; *password* – admin password. **Old syntax:** CBIOS_SetAppLicences Deprecated method use **SetAppLicences**
void SetAppLicences(byte *netLic*, byte[ ] *password*)	Writes information on application's local and network licenses to the LMT. **Input Parameters:** *netLic* – number of licenses; *password* – admin password.

Name	Description
void SetAppLicences(RuleAppLicences *ruleAppLic*, byte[ ] *password*)	Writes information about the application's local and network licenses to the LMT (taking into account extended network sharing rule). **Input Parameters:** *ruleAppLic* – number of licenses; *password* – admin password.
void SetKeyPrivate(CBIOS4NET.RijndaelCryptKey key, string password)	Sets private Key and initialization vector for hardware encryption/decryption (Rijndael algorithm) **Input Parameters:** *key* – key value. See RijndaelCryptKey class; *password* – user/admin password. **Old syntax:** CBIOS_SetKeyPrivate,CBIOS_SetIVPrivate
void SetKeySession(CBIOS4NET.RijndaelCryptKey key)	Sets session Key and initialization vector for hardware encryption/decryption (Rijndael algorithm) **Input Parameters:** *key* – key value. See RijndaelCryptKey class; **Old syntax:** CBIOS_SetKeySession,CBIOS_SetIVSession
bool SetPassword(CBIOS4NET.CryptoboxLoginType Type, string oldPassword, string newPassword)	Changes password for specified login. **Input Parameters:** *Type* – login type (admin/user); *oldPassword* – old password; *newPassword* – new password. **Old syntax:** CBIOS_SetUPW, CBIOS_SetAPW
void Unlock()	Unlocks previously locked box **Old syntax:** CBIOS_UnlockBox
void UnlockLicence()	Unlocks the currently opened partition so that other applications (processes/threads) can access it. **Old syntax:** CBIOS_ReleaseLicense
void WriteRAM(CBIOS4NET.RAM ram, byte[] data, int offset)	Writes data to a specified box's partition. Null password assumed. **Input Parameters:** *ram* – partition's ram to write data defined in RAM enumeration; *data* – data to be written; *offset* – data offset. **Old syntax:** CBIOS_WriteRAMx
void WriteRAM(CBIOS4NET.RAM ram, byte[] data, int offset, string password)	Writes data to a specified box's partition. **Input Parameters:** *ram* – partition's ram to write data defined in RAM enumeration; *data* – data to be written; *offset* – data offset; *password* – user/admin password. **Old syntax:** CBIOS_WriteRAMx

6.15 CryptoboxManager class

Abstract class which helps manage attached CRYPTO-BOX units.

Public Methods:

Name	Description
Cryptobox OpenByAppId (ushort wAppID);	Open box with provided Application ID. If more than one CRYPTO-BOX with this Application ID is attached the first one which matches the criteria will be opened. **Input Parameters:** *wAppID* – application id. **Returns:** Opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByApp
Cryptobox OpenByIndex (int iBoxIndex)	Open box by index. **Input Parameters:** *iBoxIndex* – box index. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByIndex
Cryptobox OpenByIndex (int iBoxIndex, ushort wAppID)	Open CRYPTO-BOX by index with selected Application ID. **Input Parameters:** *iBoxIndex* – box index; *wAppID* – application id. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByIndex
Cryptobox OpenByLabel (byte[] label)	Open box by binary label. If more than one CRYPTO-BOX with this label is attached the first one which matches the criteria will be opened. **Input Parameters:** *label* – label assigned to box. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByLabel
Cryptobox OpenByLabel (string label)	Open box by text label. If more than one CRYPTO-BOX with this label is attached the first one which matches the criteria will be opened. **Input Parameters:** *label* – string representation of box's label. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByLabel
Cryptobox OpenByLabel (byte[] label, ushort wAppID)	Open box by binary label selected Application ID. If more than one CRYPTO-BOX is attached the first one which matches the criteria will be opened. **Input Parameters:** *label* – label assigned to box; *wAppID* – application id. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByLabel

Name	Description
Cryptobox OpenByLabel (string label, ushort wAppID)	Open box by text label selected Application ID. If more than one CRYPTO-BOX is attached the first one which matches the criteria will be opened. **Input Parameters:** *label* – string representation of box's label; *wAppID* – application id. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByLabel
BoxHandle[] ScanBoxes ( )	Scan for attached boxes. **Returns**: an array of found CRYPTO-BOX handles **Old syntax:** CBIOS_ScanBoxes

6.16 DataObject class

Abstract class for DataObject API.

Public Properties:

Name	Description
bool IsConnected	Shows whether DataObject is connected to a Cryprobox's partition. Use Partition method Connect() to attach all DataObject from its collection to a CRYPTO-BOX partition. This property is read only.
string Name	Name parameter of DataObject.
uint Offset	DataObject's offset in partition where it is stored.
CBIOS4NET.RAM RamNumber	RAM of partition where DataObject is stored.
CBIOS4NET.DataObjectType Type	Type of DataObject (expiration date for example). This property is read only.

6.17 DataObjectAes class

Extends: DataObject
Represents Aes DataObject.

Public Methods:

Name	Description
void ClearKey(CBU2AESKeyInfo *keyInfo*, byte[ ] *password*)	Clears the key in Aes key cell and sets new keyInfo **Input Parameters:** *keyInfo* – info for key usage; *password* – password.

Name	Description
byte[ ] CryptAes(CBU2AESKeyMode *dwMode*, byte[ ] *IV*, byte[ ] *data*, byte[ ] *password*)	Encrypts data **Input Parameters:** *dwMode* – encryption mode; *IV* – initialization vector; *data* – data to encrypt*;* *password* – password. **Returns**: encrypted data
DataObjectAes(uint *handle*, uint *offset*)	Constructor **Input Parameters:** *handle* – DataObjects handle; *offset* – number of Aes key cell.
void SetKey(CBU2AESKey *key*, CBU2AESKeyInfo *keyInfo*, byte[ ] *password*)	Set the key in Aes key cell and new keyInfo **Input Parameters:** *key* – key data; *keyInfo* – info for key usage; *password* – password.

6.18 DataObjectAesBase class

Extends: DataObject
Base DataObject for old (CBU1) Aes keys.

Public Methods:

Name	Description
virtual byte[ ] CryptAes(byte[ ] *data*, byte[ ] *password*)	Encrypts data **Input Parameters:** *data* – data to encrypt*;* *password* – password. **Returns**: encrypted data
DataObjectAesBase(uint *handle*, CBIOS4NET.DataObjectType *type*)	Constructor **Input Parameters:** *handle* – DataObjects handle; *type* – type of instanced DataObject.

6.19 DataObjectAesEx class

Extends: DataObject
Represents AesEx DataObject.

Public Methods:

Name	Description
void Clear(byte[ ] *password*)	Clears descritpor in ram zone and key specified by descriptor. **Input Parameters:** *password – password.*

Name	Description
void ClearKey(CBU2AESKeyInfo *keyInfo*, byte[ ] *password*)	Clears the key specified by descriptor and sets new key info **Input Parameters:** *keyInfo* – info for key usage; *password* – *password.*
byte[ ] CryptAes(byte[ ] *IV*, byte[ ] *data*, byte[ ] *password*)	Encrypts data **Input Parameters:** *IV* – initialization vector; *data* – data to encrypt*;* *password* – password. **Returns**: encrypted data
DataObjectAesEx(uint *handle*, CBIOS4NET.RAM *ram*, uint *offset*)	Constructor **Input Parameters:** *handle* – DataObjects handle; *ram* – ram zone of descriptor; *offset* – offset in ram zone where descriptor is stored .
AesExDescriptor GetDescriptor(byte[ ] *password*)	Gets descriptor **Input Parameters:** *password* – *password.* ***Returns****:* descriptor
void SetDescriptor(AesExDescriptor *descriptor*, byte[ ] *password*)	Sets new descriptor **Input Parameters:** *descriptor* – info for key usage; *password* – *password.*
void SetKey(CBU2AESKey *key*, CBU2AESKeyInfo *keyInfo*, byte[ ] *password*)	Set the new key value **Input Parameters:** *key* – key data; *keyInfo* – info for key usage; *password* – password.

6.20 DataObjectAesFixed class

Extends: DataObjectAesBase
Represents Aes fixed DataObject.

Public Methods:

Name	Description
DataObjectAesFixed(uint *handle*)	Constructor **Input Parameters:** *handle* – DataObjects handle.

6.21 DataObjectAesPrivate class

Extends: DataObjectAesBase
Represents Aes private DataObject.

Public Methods:

Name	Description

Name	Description
void SetKey(RijndaelCryptKey *key*, byte[ ] *password*)	Set the new key value **Input Parameters:** *key* – key data; *password* – password.

6.22 DataObjectAesSession class

Extends: DataObjectAesBase
Represents Aes session DataObject.

Public Methods:

Name	Description
DataObjectAesSession(uint *handle*)	Constructor **Input Parameters:** *handle* – DataObjects handle.
void SetKey(RijndaelCryptKey *key*, byte[ ] *password*)	Set the new key value **Input Parameters:** *key* – key data; *password* – password.

6.23 DataObjectAppCRC class

Extends: DataObject
Represents Application file CRC DataObject.

Public Methods:

Name	Description
DataObjectAppCRC(uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
void SetHash(string value)	Calculates specified file's CRC and stores it to partition. Path to a file should be specified as input parameter. **Input Parameters:** *value* – Path to application's executable file.
bool Verify()	Calculates CRC of application's file and compares it to a stored hash.

6.24 DataObjectAppNameHash class

Extends: DataObject
Represents Application file name CRC DataObject.

Public Methods:

Name	Description
DataObjectAppNameHash (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
void SetHash(string value)	Calculates specified file's name CRC and stores it to partition. Path to a file should be specified as input parameter. **Input Parameters:** *value* – Path to application's executable file.
bool Verify()	Calculates CRC of application's file name and compares it to a stored hash.

6.25 DataObjectBinding class

Extends: DataObject
Represents Binding DataObject.

Public Methods:

Name	Description
DataObjectBinding(uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
DataObjectBinding(uint handle, CBIOS4NET.RAM ram, uint offset, uint flag)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset. *flag* - additional flag used for network binding (to be used with NetworkCBManager). Default value is 0 – the data object will bind the box to server's computer; DataObjectBinding.BIND_LOCAL_FLAG – specifies that the box (attached to the server) will be bound to the local computer (calling client). In this case one data object per client allocation should be considered.
void Activate(byte[ ] activation, byte[ ] password)	Activates dataobject. Status – Activated. **Input Parameters:** *activation* – activation sequence.(activation request signed with private distributor rsa key); *password* – unique identifier for data object.
void Bind(byte[ ] password)	Binds dataobject to PC hardware. Status – Bound. **Input Parameters:** *password* – unique identifier for data object.

Name	Description
void Clear(byte[ ] password)	Clears dataobject's binding. Status – Unbound **Input Parameters:** *password* – unique identifier for data object.
byte[] GenerateActivation(byte[] password)	Generates activation code that can be used to activate dataobject. **Input Parameters:** *password* – unique identifier for data object. **Returns**: activation code
DOBindingStatus GetStatus(byte[] password)	Returns binding status. For all statuses except for Unbound the Verify method should be used. That will define if binding status is correct. **Input Parameters:** *password* – unique identifier for data object. **Returns**: bind status
void Init(byte[] password)	Inits data object internal structure with default values. Status – Unbound **Input Parameters:** *password* – unique identifier for data object.
bool Verify(byte[] password)	Verifies that returned DOBindingStatus is correct. **Input Parameters:** *password* – unique identifier for data object. **Returns**: whether got status is correct

6.26 DataObjectDWORD class

Extends: DataObject
Represents DWORD DataObject.

Public Properties:

Name	Description
uint Value	Gets ot sets DataObject value. **Old syntax:** CBIOS_DOSet, CBIOS_DOGet

Public Methods:

Name	Description
DataObjectDWORD (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.

6.27 DataObjectExpirationDate class

Extends: DataObject
Represents expiration date DataObject.

Public Properties:

Name	Description
uint DaysLeft	Days left. **Old syntax:** CBIOS_DOSet, CBIOS_DOGet
System.DateTime ExpirationDate	Expiration date. **Old syntax:** CBIOS_DOSet, CBIOS_DOGet

Public Methods:

Name	Description
void Clear()	Clears expiration date. **Old syntax:** CBIOS_DOClear
DataObjectDWORD (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
void Dec()	Decreases expiration date by 1 day. **Old syntax:** CBIOS_DODec
void Dec(uint days)	Decreases expiration date by specified number of days. **Input Parameters:** *days* – number of days to decrement expiration date. **Old syntax:** CBIOS_DODec
void Inc()	Increases expiration date by 1 day. **Old syntax:** CBIOS_DOInc
void Inc(uint days)	Increases expiration date by specified number of days. **Input Parameters:** *days* – number of days to decrement expiration date. **Old syntax:** CBIOS_DOInc
void Unlimited()	Sets expiration date to an unlimited value. **Old syntax:** CBIOS_DOUnlimited
bool Verify()	Verifies a DataObject value in the CRYPTO-BOX partition. **Old syntax:** CBIOS_DOVerify

6.28 DataObjectExpirationDateEx class

Extends: DataObject
Represents expiration date DataObject; uses 8 bytes in box partition memory.

Public Properties:

Name	Description
uint DaysLeft	Days left. **Old syntax:** CBIOS_DOGet
System.DateTime ExpirationDate	Expiration date. **Old syntax:** CBIOS_DOGet

Name	Description
UInt32 SecondsLeft	Seconds left. **Old syntax:** CBIOS_DOGet

Public Methods:

Name	Description
void Clear(byte[] password)	Clears expiration date. **Input Parameters:** *password* – password. **Old syntax:** CBIOS_DOClear
DataObjectExpirationDateEx(uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
void Dec(byte[] password)	Decreases expiration date by 1 day. **Input Parameters:** *password* – password. **Old syntax:** CBIOS_DODec
void Dec(uint days, byte[] password)	Decreases expiration date by specified number of days. **Input Parameters:** *days* – number of days to decrement expiration date; *password* – password. **Old syntax:** CBIOS_DODec
void Inc(byte[] password)	Increases expiration date by 1 day. **Input Parameters:** *password* – password. **Old syntax:** CBIOS_DOInc
void Inc(uint days, byte[] password)	Increases expiration date by specified number of days. **Input Parameters:** *days* – number of days to decrement expiration date; *password* – password. **Old syntax:** CBIOS_DOInc
void SetExpirationDate(DateTime value, byte[] password)	Sets expiration date. **Input Parameters:** *value* – expiration date value; *password* – password. **Old syntax:** CBIOS_DOSet
void SetExpirationDate(DateTime value, DateTime timeStamp, byte[] password)	Sets expiration date and timestamp values. **Input Parameters:** *value* – expiration date value; *timeStamp* – timestamp value; *password* – password. **Old syntax:** CBIOS_DOSet
void Unlimited(byte[] password)	Sets expiration date to an unlimited value. **Input Parameters:** *password* – password. **Old syntax:** CBIOS_DOUnlimited

Name	Description
bool Verify(byte[] password)	Verifies a DataObject value in the CRYPTO-BOX partition. **Input Parameters:** *password* – password. **Old syntax:** CBIOS_DOVerify

6.29 DataObjectMemory class

Extends: DataObject
Represents memory data object, enables read/write operations with CRYPTO-BOX partition memory.

Public Methods:

Name	Description
DataObjectMemory (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
byte[] GetMemory(uint memSize)	Reads data from box. null offset assumed + DataObject's offset. **Input Parameters:** *memSize* – size of memory block to read.
byte[] GetMemory(uint memSize, uint offset)	Reads data with specified relative offset. **Input Parameters:** *memSize* – size of memory block to read; *offset* – memory offset.
void SetMemory(byte[] value)	Writes data to box's partition. Null relative offset assumed. **Input Parameters:** *value* – data to store on box.
void SetMemory(byte[] value, uint offset)	Writes data to box's partition with specified relative offset. **Input Parameters:** *value* – data to store in box; *offset* – relative offset of stored data.

6.30 DataObjectNetLicence class

Extends: DataObject
It provides Data Object functionality for network license. With current limitation only one connected instance of **DataObjectNetLicence** is allowed for application. You have to disconnect the first network licensing data object in order to work with another one.
Example: if application works with network license for partition "999" of box A, then connecting a **DataObjectNetLicence** object for box B without disconnecting the object or box A first will throw exception. To disconnect data object from the box use **Partition.Remove** or **Partition.Disconnect** method.

Public Methods:

Name	Description
void Clear(byte[ ] apw)	Clears net license. **Input Parameters:** *apw* – admin password value.
DataObjectNetLicence (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
uint getValue(byte[ ] apw)	Reads net license value. **Input Parameters:** *apw* – admin password value.
void Inc(byte[ ] *apw*)	Increments net license amount by 1 **Input Parameters:** *apw* – admin password value.
void Inc(uint *counter*, byte[ ] *apw*)	Increments net license amount by counter **Input Parameters:** *counter* – amount of licenses to add; *apw* – admin password value.
void setValue(uint value, byte[ ] apw)	Writes net license value. **Input Parameters:** *value* – network license amount; *apw* – admin password value.
void Unlimited(byte[ ] apw)	Sets net license to unlimited value. **Input Parameters:** *apw* – admin password value.

6.31 DataObjectNetLicenceEx

Extends: DataObject
It provides Data Object functionality for network license. With current limitation only one connected instance of **DataObjectNetLicence** is allowed for application. You should disconnect first network licensing data object in order to work with another one.
Example: if application works with network license for partition "999" of box A, then connecting a **DataObjectNetLicence** object for box B without disconnecting the object or box A first will throw exception. To disconnect data object from the box use **Partition.Remove** or **Partition.Disconnect** method.

Public Methods:

Name	Description
void Clear(byte[ ] apw)	Clears net license. **Input Parameters:** *apw* – admin password value.

Name	Description
DataObjectNetLicence (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
uint getValue(byte[ ] apw)	Reads net license value. **Input Parameters:** *apw* – admin password value.
void Inc(byte[ ] *apw*)	Increments net license amount by 1 **Input Parameters:** *apw* – admin password value.
void Inc(uint *counter*, byte[ ] *apw*)	Increments net license amount by counter **Input Parameters:** *counter* – amount of licenses to add; *apw* – admin password value.
void setValue(DONetLicenceExData *value*, byte[ ] *apw*)	Writes net license value. **Input Parameters:** *value* – network license amount; *apw* – admin password value.
void Unlimited(byte[ ] apw)	Sets net license to unlimited value. **Input Parameters:** *apw* – admin password value.

6.32 DataObjectNumberOfDays class

Extends: DataObject
Represents number of days DataObject.

Public Properties:

Name	Description
uint Value	Gets or sets number of days value. **Old syntax:** CBIOS_DOSet, CBIOS_DOGet

Public Methods:

Name	Description
void Clear()	Clears number of days counter.
DataObjectNumberOfDays (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
void Dec()	Decreases number of days value by 1 day.

Name	Description
void Dec(uint days)	Decreases number of days value by specified number of days. **Input Parameters:** *days* – number of days to decrement data object's value.
void Inc()	Increases number of days value by 1 day.
void Inc(uint days)	Increases number of days value by specified number of days. **Input Parameters:** *days* – number of days to increment data object's value.
void Unlimited()	Sets number of days to unlimited value.
bool Verify()	Verifies whether number of days value is valid and decreases it.

6.33 DataObjectPasswordHash class

Extends: DataObject
Represents Application file name CRC DataObject.

Public Methods:

Name	Description
DataObjectPasswordHash (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
void SetHash(string value)	Calculates password's CRC and stores it to partition. **Input Parameters:** *value* – password's value.
bool Verify(string hash)	Compares calculated password's CRC to stored hash value. Returns "true" on success. **Input Parameters:** *hash* – password's value.

6.34 DataObjectRsa

Extends: DataObjectRsaBase
Represents Rsa DataObject.

Public Methods:

Name	Description
void CalcFileSignature(CBU2RSAKeyMode *dwMode*, string *fileName*, string *signatureName*, byte[ ] *password*)	Calculates file signature **Input Parameters:** *dwMode* – mode of Rsa usage; *fileName* – path to a file to calculate signature; *signatureName* – path to signature file; *password* – password.
void ClearKey(CBU2RSAKeyInfo *keyInfo*, byte[ ] *password*)	Clears the key in Rsa key cell and sets new keyInfo **Input Parameters:** *keyInfo* – info for key usage; *password* – password.
DataObjectRsa(uint *handle*, uint *offset*)	Constructor **Input Parameters:** *handle* – DataObjects handle; *offset* – number of Rsa key cell.
CBU2RSAKey Generate(CBU2RSAKeyInfo *keyInfo*, byte[ ] *password*)	Generates new Rsa key and stores it in Rsa key cell **Input Parameters:** *keyInfo* – info for key usage; *password* – password. **Returns**: public part of generated key
void SetKey(CBU2RSAKey *key*, CBU2RSAKeyInfo *keyInfo*, byte[ ] *password*)	Set the new key value **Input Parameters:** *key* – key data; *keyInfo* – info for key usage; *password* – password.
bool ValidateFileSignature(CBU2RSAKeyMode *dwMode*, string *fileName*, string *signatureName*, byte[ ] *password*)	Validates file signature **Input Parameters:** *dwMode* – mode of Rsa usage; *fileName* – path to a file; *signatureName* – path to signature file; *password* – password. **Returns**: whether signature is valid

6.35 DataObjectRsaBase

Extends: DataObject
Base abstract class for RSA DO. Provides Encrypt/Decrypt methods

Public Methods:

Name	Description
DataObjectRsaBase(uint *handle*, DataObjectType *type*, uint *offset*)	Constructor **Input Parameters:** *handle* – handle; *type* – type of DataObject; *offset* – offset;

Name	Description
byte[ ] Decrypt(CBU2RSAKeyMode *dwMode*, byte[ ] *data*, byte[ ] *password*)	Decrypts data **Input Parameters:** *dwMode* – mode of Rsa usage;; *data* – data; *password* – password. **Returns**: decrypted data
byte[ ] Encrypt(CBU2RSAKeyMode *dwMode*, byte[ ] *data*, byte[ ] *password*)	Encrypts data **Input Parameters:** *dwMode* – mode of Rsa usage;; *data* – data; *password* – password. **Returns**: encrypted data

6.36 DataObjectRsaClient class

Extends: DataObjectRsaBase
Represents RsaClient DataObject.

Public Methods:

Name	Description
DataObjectRsaClient(uint *handle*)	Constructor **Input Parameters:** *handle* – handle;

6.37 DataObjectRsaDistributor class

Extends: DataObjectRsaBase
Represents RsaDistributor DataObject.

Public Methods:

Name	Description
DataObjectRsaDistributor(uint *handle*)	Constructor **Input Parameters:** *handle* – handle;

6.38 DataObjectRsaEx class

Extends: DataObject
Represents RsaEx DataObject.

Public Methods:

Name	Description
void Clear(byte[ ] *password*)	Clears descritpor in ram zone and key specified by descriptor. **Input Parameters:** *password* – *password.*
void ClearKey(CBU2RSAKeyInfo *keyInfo*, byte[ ] *password*)	Clears the key specified by descriptor and sets new key info **Input Parameters:** *keyInfo* – info for key usage; *password* – password.
DataObjectRsaEx(uint *handle*, RAM *ram*, uint *offset*)	Constructor **Input Parameters:** *handle* – DataObjects handle; *ram* – ram zone of descriptor; *offset* – offset in ram zone where descriptor is stored .
byte[ ] Decrypt(byte[ ] *data*, byte[ ] *password*)	Decrypts data **Input Parameters:** *data* – data to decrypt; *password* – password. **Returns**: decrypted data
byte[ ] Encrypt(byte[ ] *data*, byte[ ] *password*)	Encrypts data **Input Parameters:** *data* – data to encrypt; *password* – password. **Returns**: encrypted data
CBU2RSAKey Generate(CBU2RSAKeyInfo *keyInfo*, byte[ ] *password*)	Generates new Rsa key and stores it **Input Parameters:** *keyInfo* – info for key usage; *password* – password. **Returns**: public part of generated key
RsaExDescriptor GetDescriptor(byte[ ] *password*)	Gets descriptor **Input Parameters:** *password* – password. **Returns**: descriptor
void SetDescriptor(RsaExDescriptor *descriptor*, byte[ ] *password*)	Sets descriptor **Input Parameters:** *descriptor* – desctiptor value; *password* – password.
void SetKey(CBU2RSAKey *key*, CBU2RSAKeyInfo *keyInfo*, byte[ ] *password*)	Set the new key value **Input Parameters:** *key* – key data; *keyInfo* – info for key usage; *password* – password.

6.39 DataObjectRunCounter class

Extends: DataObject
Represents run counter DataObject.

Public Properties:

Name	Description
uint Value	Gets or sets run counter value. **Old syntax:** CBIOS_DOSet, CBIOS_DOGet

Public Methods:

Name	Description
void Clear()	Clears run counter.
DataObjectRunCounter (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
void Dec()	Decreases run counter value by 1.
void Dec(uint counter)	Decreases run counter value by specified number. **Input Parameters:** *counter* – decrement value.
void Inc()	Increases run counter value by 1.
void Inc(uint counter)	Increases run counter value by specified number. **Input Parameters:** *counter* – increment value.
void Unlimited()	Sets run counter to unlimited value.

6.40 DataObjectSignature class

Extends: DataObject
Represents signature DataObject.

Public Methods:

Name	Description
byte[ ] CalculateDigitalSignature(byte[ ] *data*, byte[ ] *password*)	Calculates digital signature **Input Parameters:** *data* – data to sign; *password* – password. **Returns**: signature
void CalculateDigitalSignatureF(string *dataFilePath*, string *signatureFilePath*, byte[ ] *password*)	Calculates digital signature **Input Parameters:** *dataFilePath* – path to data file; *signatureFilePath* – path to signature file; *password* – password.
DataObjectSignature(uint *handle*, RAM *ram*, uint *offset*)	Constructor **Input Parameters:** *handle* – DataObjects handle; *ram* – ram zone of descriptor; *offset* – offset in ram zone where descriptor is stored .

Name	Description
CBU2RSAKey GenerateA(CBU2RSAKeyInfo *keyInfo*, byte[ ] *password*)	Generates new Rsa key and stores it as A key of signature **Input Parameters:** *keyInfo* – info for key usage; *password* – password. **Returns**: public part of generated key
CBU2RSAKey GenerateB(CBU2RSAKeyInfo *keyInfo*, byte[ ] *password*)	Generates new Rsa key and stores it as B key of signature **Input Parameters:** *keyInfo* – info for key usage; *password* – password. **Returns**: public part of generated key
SignatureData GetSignatureData(byte[ ] *password*)	Gets signature description **Input Parameters:** *password* – password. **Returns**: signature description
void SetKeys(CBIOS4NET.CBU2RSAKey *keyA*, CBU2RSAKeyInfo *keyInfoA*, CBU2RSAKey *keyB*, CBU2RSAKeyInfo *keyInfoB*, byte[ ] *password*)	Set the new signature keys value **Input Parameters:** *keyA* – A key data; *keyInfoA* – info for A key usage; *keyB* – B key data; *keyInfoB* – info for B key usage; *password* – password.
void SetSignatureData(SignatureData *signatureData*, byte[ ] *password*)	Sets signature description **Input Parameters:** *signatureData* – signature data value; *password* – password.
DateTime? ValidateDigitalSignature(byte[ ] *data*, byte[ ] *signature*, byte[ ] *password*)	Validates signature **Input Parameters:** *data* – data; *signature* – signature value to validate; *password* – password. **Returns**: Timestamp when signature was created
DateTime? ValidateDigitalSignatureF(string *dataFilePath*, string *signatureFilePath*, byte[ ] *password*)	Validates file signature **Input Parameters:** *dataFilePath* – path to data file; *signatureFilePath* – path to signature file; *password* – password. **Returns**: Timestamp when signature was created

6.41 DataObjectTimeAllowed class

Extends: DataObject
Represents time allowed DataObject.

Public Properties:

Name	Description
uint Value	Gets or sets time allowed value. **Old syntax:** CBIOS_DOSet, CBIOS_DOGet

Public Methods:

Name	Description
void Clear()	Clears time allowed value.
DataObjectTimeAllowed (uint handle, CBIOS4NET.RAM ram, uint offset)	DataObject constructor. Initializes its base parameters. **Input Parameters:** *handle* – unique identifier for data object; *ram* – partition's ram where data object is stored; *offset* – data object's offset.
void Dec()	Decreases time allowed value by 1 second.
void Dec(uint seconds)	Decreases time allowed value by specified number of seconds. **Input Parameters:** *seconds* – decrement value.
void Inc()	Increases time allowed value by 1 second.
void Inc(uint seconds)	Increases time allowed value by specified number of seconds. **Input Parameters:** *seconds* – increment value.
void Unlimited()	Sets time allowed to unlimited value.
bool Verify(uint seconds)	Verifies whether data object is valid for specified number of seconds and decreases data object's value by that number. **Input Parameters:** *seconds* – decrement value.

6.42 DONetLicenceExData struct

Represents data of netLic.

Public Properties:

Name	Description
byte bNetLic	NetLicence amount
byte bRuleId	ruleId
ushort wReserved	Some data

6.43 LocalCBManager class

Extends: CryptoboxManager, IDisposable
Cryptobox manager which helps manage local CRYPTO-BOX units.

Public Events:

Name	Description
event CBIOSNotifyEventHandler CBIOSNotify;	This event is raised when CRYPTO-BOX is plugged or unplugged from local computer. Event is fired in internal CBIOS thread context. **Old syntax:** CBIOS_RegisterNotificationCallback, CBIOS_UnregisterNotificationCallback

6.44 MD5 class

MD5 class allows calculation of md5 checksum.

Public Methods:

Name	Description
static byte[] GetHash(byte[] data)	Returns data's md5 hash. **Input Parameters:** *data* – data to calculate hash. **Old syntax:** CBIOS_MD5Hash

6.45 NETCBIOSConnectionInfo struct

Specifies connection to a CBIOS Server instance.

Public Members:

Name	Description
uint dwBoxName	Opened box name by connection
uint dwHandle	Connection handle
bool isLicenceLocked	Shows whether network license is locked by connection
uint Timeout	Timeout parameter for the connection
string UserAppFileName	Application file name that opened the connection
string UserComputerName	Smb host name that opened the connection
ushort wAppID	Opened application id of the box

6.46 NETCBIOSKeepAliveParams struct

Keep alive parameters.

Public Members:

Name	Description
uint ScanRateMSec	Scan rate value
int TimeoutSec	Timeout value

6.47 NetworkCBManager class

Extends: CryptoboxManager, IDisposable
Cryptobox manager which helps manage CRYPTO-BOX units available in the network.

Public Properties:

Name	Description
bool IsConnected	Is this manager connected to remote server. This property is read only.
ushort ScanPort	Default port for network scan **Old syntax:** CBIOS_GetScanPort, CBIOS_SetScanPort

Public Methods:

Name	Description
ServerInfo[] ScanNetwork (int dwScanTimeMs);	Scan network for servers. **Input Parameters:** *dwScanTimeMs* – time in milliseconds allowed for operation. **Returns:** array of founded servers. **Old syntax:** CBIOS_ScanNetwork
void Connect(string server, ushort port);	Connects a manager to a network server. **Input Parameters:** *server* – server's ip address/name; *port* – port listened by server. **Old syntax:** CBIOS_Connect
void Disconnect();	Disconnect from currently connected network server. **Old syntax:** CBIOS_Disconnect
bool CheckServerPasswd(byte[ ] *srvPass*)	Checks password for connected CBIOS server **Input Parameters:** *srvPass* – server password. **Returns:** true if password check succeds. **Old syntax:** CBIOS_CheckServerPasswd
int GetConnTimeout(byte[ ] *srvPass*)	Gets server connection timeout parameter **Input Parameters:** *srvPass* – server password. **Returns:** timeout value. **Old syntax:** CBIOS_GetConnTimeout
NETCBIOSKeepAliveParams GetKeepAliveParams(byte[ ] *srvPass*)	Get keep alive parameters for connected server **Input Parameters:** *srvPass* – server password. **Returns:** keep alive parameters value. **Old syntax:** CBIOS_GetConnTimeout
NETCBIOSConnectionInfo[ ] GetSessionList()	Gets server's current connections list **Returns:** array of connections. **Old syntax:** CBIOS_GetSessionList
void RestartServer(byte[ ] *srvPass*)	Restart server **Input Parameters:** *srvPass* – server password. **Old syntax:** CBIOS_RestartServer

Name	Description
void SetConnTimeout(int *lTimeoutSec*, byte[ ] *srvPass*)	Set connection timeout **Input Parameters:** *lTimeoutSec* – timeout value; *srvPass* – server password. **Old syntax:** CBIOS_SetConnTimeout
void SetKeepAliveParams(NETCBIOSKeepAliveParams *params*, byte[ ] *srvPass*)	Set keep alive parameters for connected server **Input Parameters:** *params* – keep alive parameters struct; *srvPass* – server password. **Old syntax:** CBIOS_SetKeepAliveParams
void SetPort(ushort *port*, byte[ ] *srvPass*)	Sets server's listen port **Input Parameters:** *port* – port value; *srvPass* – server password. **Old syntax:** CBIOS_SetPort
void SetServerPasswd(byte[ ] *oldPass*, byte[ ] *newPass*)	Set new server password **Input Parameters:** *oldPass* – server password; *newPass* – server password. **Old syntax:** CBIOS_SetServerPasswd
void StopServer(byte[ ] *srvPass*)	Stop connected server **Input Parameters:** *srvPass* – server password. **Old syntax:** CBIOS_StopServer

6.48 Partition class

Extends: AppInfo, ICollection<DataObject>
Partition class represents CRYPTO-BOX partition and helps to manage its DataObjects.

NOTE: There is a restriction. Only one Connected instance of DataObjectNetLicence is allowed. See DataObjectNetLicence description for more info.

Public Properties:

Name	Description
bool IsConnected	Is this partition connected to a CRYPTO-BOX with opened partition. See Connect method for datails. This property is read only.
DataObject this[int index]	Numeric indexed iterator for data objects collection.
DataObject this[string name]	String indexed iterator for data objects collection. DataObject's Name property used.
uint TransactionMark { set; get; }	Partition's transaction mark. **Old syntax:** CBIOS_GetTransactionMark, CBIOS_SetTransactionMark

Public Methods:

Name	Description
bool Connect(CBIOS4NET.Cryptobox box)	Connects partition object to CRYPTO-BOX partition. On success all data objects added to partition's collection can be used. **Input Parameters:** *box* – Cryptobox class instance to connect to. **Returns:** success result of operation. **Old syntax:** TEOS_DoCreateReference
void Disconnect()	Disconnects from currently connected CRYPTO-BOX. **Old syntax:** TEOS_DoClearReferences
void Load(byte[] source)	Loads previously saved data objects map of partition from binary data. Partition should be connected to use this method. **Input Parameters:** *source* – previously stored data object's map data. **Old syntax:** TEOS_DoLoadMap
public Partition()	Objects constructor.
byte[] Save()	Stores currently specified data objects map to a binary array. **Old syntax:** TEOS_DoSaveMap

6.49 RijndaelCryptKey class

This class represents a key for symmetric encryption/decryption (Rijndael algorithm).

Public Properties:

Name	Description
byte[] EncryptionKey	Gets or sets encryption key.
byte[] InitializationVector	Gets or sets initialization vector.

6.50 RsaExDescriptor struct

Descriptor of RsaEx DO

Old syntax: struct DO_RSA_EX_DATA

Public Properties:

Name	Description
uint keyIndex	Key index in Rsa cell
RsaExKeyLocation location	Location description (Rsa key cell, client, distributor etc.)
CBU2RSAKeyMode mode	Mode of Rsa encryption

Name	Description
uint reserved	Reserved value
uint updateDescr	version/update rule

6.51 RSAPrivateKey class

This class represents private key for RSA encryption.
Old syntax: struct CBIOS_RSA_PRIVATE_KEY

Public Properties:

Name	Description
uint Bits	Number of key bits.
byte[] Coefficient	Keys coefficient.
byte[] Exponent	Keys exponent.
byte[] Modulus	Keys modulus.
byte[,] PrimeExponents	Keys prime exponent.
byte[,] PrimeFactors	Keys prime factors.
byte[] PublicExponent	Keys public exponent.

6.52 RSAPublicKey class

Class represents public key for RSA encryption.
Old syntax: struct CBIOS_RSA_PUBLIC_KEY

Public Properties:

Name	Description
uint Bits	Number of key bits.
byte[] Exponent	Keys exponent.
byte[] Modulus	Keys modulus.

6.53 RuleAppLicences struct

RuleId+ NetLicence.

Public Properties:

Name	Description
byte NetLicences	Number of Net licenses
byte RuleId	ruleId

Public Methods:

Name	Description
RuleAppLicences(byte *ruleId*, byte *netLicences*)	Constructor **Input Parameters:** *ruleId* – ruleId; *netLicences* – netLic amount

6.54 ServerInfo class

Class represents information about network server.

Public Properties:

Name	Description
int BoxCount	Number of boxes connected to network server. This property is readonly.
string Name	Server's name. This property is readonly.
ushort ScanPort	Network port listened by server. This property is readonly.

6.55 SignatureData struct

Description of Signature DO
Old syntax: struct DO_SIGNATURE_DATA

Public Properties:

Name	Description
SignatureHashType hashType	Hash calculation algorithm
RsaExDescriptor keyA	Descritpion of A rsa key
RsaExDescriptor keyB	Descritpion of B rsa key
SignatureTimestampType timeStampSize	Type of timestamp

7. RU4NET: Description of classes

7.1 ActivationRecord class

Extends: ICollection<ActivationRecordStep>
This class represents a *record* of activation sequence. It corresponds to an application or partition in CRYPTO-BOX memory. It describes a set of operations to update partition's data objects.

Public Properties:

Name	Description
ushort AppId	Application ID.
int Count	ActivationRecordStep count. This property is readonly.
ActivationRecordStep this[int index]	Numeric indexed iterator for ActivationRecordSteps collection.

Public Methods:

Name	Description
ActivationRecord()	Object's constructor
ActivationRecord(ushort wAppId, IEnumerable<ActivationRecordStep> steps)	Object's constructor **Input Parameters:** *wAppId* – Application ID; *steps* – enumeration of record steps to add to object's collection.

7.2 ActivationRecordEx class

Extends: ActivationRecord
This class represents a *record* of activation sequence. It corresponds to an application or partition in CRYPTO-BOX memory. It describes a set of operations to update partition's data objects and its size.

Public Properties:

Name	Description
uint CheckRAM1Size	Defines a partition's RAM1 size used to find an partition
uint CheckRAM2Size	Defines a partition's RAM2 size used to find an partition
uint CheckRAM3Size	Defines a partition's RAM3 size used to find an partition
uint NewRAM1Size	A new size of parition's RAM1 zone.
uint NewRAM2Size	A new size of parition's RAM2 zone.
uint NewRAM3Size	A new size of parition's RAM3 zone.
ActivationRecordUpdateMode UpdateMode	An update operation applied to partition. (resize, delete)

Public Methods:

Name	Description
ActivationRecordEx()	Object's constructor

7.3 ActivationRecordStep class

Basic class representing single update operation on target data object.

Public Properties:

Name	Description
uint Handle	Handle of data object to update.
uint Offset	Offset of target data object.
RAM RamNumber	RAM number where data object is stored.

Public Methods:

Name	Description
ActivationRecordStep()	Object's constructor

7.4 ActivationSequenceBuilder class

Extends: ICollection<ActivationRecord>
A builder class for activation sequence.

Public Properties:

Name	Description
int Count	ActivationRecord count.
int Index { get; }	Numeric indexator of ActivationRecord collection.
RijndaelCryptKey KeyPrivate	Specifies a private AES key to be stored in the box during update. Default value is null (not applied).
RijndaelCryptKey KeySession	Specifies a session AES key to be stored in the box during update. Default value is null (not applied).

Public Methods:

Name	Description
ActivationSequenceBuilder()	Object's constructor
ActivationSequenceBuilder(IEnumerable<ActivationRecord> records)	Object's constructor **Input Parameters:** *records* – enumeration of ActivationRecord to add in activation sequence.
ActivationRecord GetByAppId(ushort wAppId)	Gets an ActivationRecord by its partition number **Returns:** ActivationRecord. **Input Parameters:** *wAppId* – number of partition (name).
byte[] MakeActivationCode(uint transactionMark, DecryptKeys keys)	Generates activation sequence **Returns:** activation code **Input Parameters:** *transactionMark* – transaction mark of a request; *keys* – user's DecryptKeys.
static byte[] MakeActivationCode(uint transactionMark, DecryptKeys keys, params ActivationRecord[] records)	Static method generating activation sequence. **Returns:** activation code **Input Parameters:** *transactionMark* – transaction mark of a request; *keys* – user's DecryptKeys; *records* – array of ActivationRecord(s) used in activation sequence.
static byte[ ] MakeActivationCode(uint transactionMark, DecryptKeys keys, RijndaelCryptKey keySession, RijndaelCryptKey keyPrivate, params ActivationRecord[ ] records)	Static method generating activation sequence. **Returns:** activation code **Input Parameters:** *transactionMark* – transaction mark of a request; *keys* – user's DecryptKeys; *keySession* – specifies a session key to be stored in box. Can be null; *keyPrivate* – specifies a private key to be stored in box. Can be null; *records* – array of ActivationRecord(s) used in activation sequence.

7.5 Activator class

This class is used to execute activation sequence for a CRYPTO-BOX.

Public Methods:

Name	Description
static void Execute(byte[] activationCode, BoxInfo box, string UPW, string APW)	Static method that executes activation sequence. **Input Parameters:** *activationCode* – activation code; *box* – BoxInfo that describes target box; *UPW* – user password; *APW* – admin password.

7.6 DecryptKeys class

A storage class for RSA keys used for authorization.

Public Properties:

Name	Description
byte[] RSADistribPrivateKey	Distributor's private key.
byte[] RSAUserPublicKey	Public user's key.

Public Methods:

Name	Description
DecryptKeys(byte[] keyRSAUserPublic, byte[] keyRSADistribPrivate)	Object's constructor. **Input Parameters:** *keyRSAUserPublic* – public key; *keyRSADistribPrivate* – private key.

7.7 Request class

Extends: IRequestBaseParameters
This class is used to read and translate request for update.

Public Properties:

Name	Description
RequestEncryptionMask NotEncryptedMask	RequestEncryptionMask enum shows request fields that not encrypted. This property is readonly.
uint ProjectId	ProjectId parameter. This property is readonly.
RequestParameter this[Guid paramGuid]	Indexator of user defined request parameters by guid index.
uint TransactionMark	TransactionMark generated for request. This property is readonly.
uint UserId	UserId parameter. This property is readonly.
uint UserParameter	UserParameter. This property is readonly.

Public Methods:

Name	Description
void Load(byte[] data)	Method used to load request data and fill object's fields. **Input Parameters:** *data* – request data.
void Load(byte[] data, DecryptKeys keys)	Method used to load request data and fill object's fields. **Input Parameters:** *data* – request data; *keys* – keys used to decrypt encrypted parameters.
Request()	Object's constructor.
Request(byte[] data)	Object's constructor. Loads request data. **Input Parameters:** *data* – request data.
Request(byte[] data, DecryptKeys keys)	Object's constructor. Loads and translates request data with specified keys. **Input Parameters:** *data* – request data; *keys* – keys used to decrypt encrypted parameters.
void Translate(DecryptKeys keys)	Translates request (decrypts data). **Input Parameters:** *keys* – keys used to decrypt encrypted parameters.

7.8 RequestBaseParameters class

Extends: IRequestBaseParameters
Class represents request base parameters.

Public Properties:

Name	Description
RequestEncryptionMask NotEncryptedMask	Property defines request fields that should not be encrypted.
uint ProjectId	ProjectId parameter.
uint UserId	UserId parameter.
uint UserParameter	UserParameter.

Public Methods:

Name	Description
RequestBaseParameters()	Object's constructor.

7.9 RequestBuilder class

Extends: RequestBaseParameters, ICollection<RequestParameter>
The class is used to create a request for update.

Public Properties:

Name	Description
int Count	User defined parameter's count.

Public Methods:

Name	Description
byte[] MakeRequest(BoxInfo box, string UPW)	Creates request sequence. **Returns:** request data **Input Parameters:** *box* – BoxInfo that describes target box; *UPW* – user password.
static byte[] MakeRequest(BoxInfo box, IRequestBaseParameters baseParams, string UPW, params RequestParameter[] params)	Creates request sequence **Returns:** request data **Input Parameters:** *box* –BoxInfo that describes target box; *baseParams* – request base parameters; *UPW* – user password; *params* – array of user defined parameters.
RequestBuilder()	Object's constructor.

7.10 RequestParameter class

The class represents user defined request parameter.

Public Properties:

Name	Description
bool Encrypted	Boolean flag whether to encrypt parameter.
System.Guid Guid	Guid used to retrieve parameter
byte[] Value	Data

Public Methods:

Name	Description
RequestParameter(System.Guid guid, byte[] value, bool encrypted)	Object's constructor. **Input Parameters:** *guid* – Guid value; *value* – data; *encrypted* – Encrypted flag.

7.11 StepAes

Extends: ActivationRecordStep
Specifies operation to update DataObjectAes.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
CBU2AESKey Key	Aes key to set

Public Methods:

Name	Description
StepAes()	Object's constructor.

7.12 StepAesEx

Extends: ActivationRecordStep
Specifies operation to update DataObjectAesEx.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
AesExDescriptor Descriptor	Descriptor value to set
CBU2AESKey Key	Aes key to set. Can be null

Public Methods:

Name	Description
StepAesEx()	Object's constructor.

7.13 StepAesPrivate

Extends: ActivationRecordStep
Specifies operation to update DataObjectAesPrivate.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
RijndaelCryptKey Key	Aes key to set

Public Methods:

Name	Description
StepAesPrivate()	Object's constructor.

7.14 StepAesSession

Extends: ActivationRecordStep
Specifies operation to update DataObjectAesSession.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
RijndaelCryptKey Key	Aes key to set

Public Methods:

Name	Description
StepAesSession()	Object's constructor.

7.15 StepAppCRC class

Extends: ActivationRecordStep
Specifies operation to update DataObjectAppCRC.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
string Value	Full path for application's executable file.

Public Methods:

Name	Description
StepAppCRC()	Object's constructor.

7.16 StepAppNameHash class

Extends: ActivationRecordStep
Specifies operation to update DataObjectAppNameHash.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
string Value	Full path for application's executable file.

Public Methods:

Name	Description
StepAppNameHash()	Object's constructor.

7.17 StepBinding class

Extends: ActivationRecordStep
Specifies operation to update DataObjectBinding.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
byte[] Activation	Activation code. Activation request signed with private distributor's RSA key.

Public Methods:

Name	Description
StepBinding()	Object's constructor.

7.18 StepDWORD class

Extends: ActivationRecordStep
Specifies operation to update DataObjectDWORD.

Public Properties:

Name	Description
ActionType Action	A type of update operation. Readonly property
uint Value	DWORD value

Public Methods:

Name	Description
StepDWORD()	Object's constructor.

7.19 StepExpirationDate class

Extends: ActivationRecordStep
Specifies operation to update DataObjectExpirationDate.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
uint DaysCount	Days count. Used for all operations except ActionType.Set.
DateTime ExpirationDate	A date to be set as data object's value. Used with ActionType.Set operation only.

Public Methods:

Name	Description
StepExpirationDate()	Object's constructor.

7.20 StepExpirationDateEx class

Extends: StepExpirationDate
Specifies operation to update DataObjectExpirationDateEx.

Public Properties:

Name	Description
DateTime? TimeStamp	Optional timestamp value to set while update. Used with ActionType.Set operation only. DateTime.Now value would be used by default.

Public Methods:

Name	Description
StepExpirationDateEx()	Object's constructor.

7.21 StepMemory class

Extends: ActivationRecordStep
Specifies operation to update DataObjectMemory.

Public Methods:

Name	Description
void SetMemory(byte[] value)	Sets a memory block.
void SetMemory(byte[] value, uint offset)	Sets a memory block with corresponding offset.

Name	Description
StepMemory()	Object's constructor.

7.22 StepNetLicence class

Extends: ActivationRecordStep
Specifies operation to update DataObjectNetLicence.

MakeActivationCode method of **ActivationSequenceBuilder** class uses CBIOS4NET data objects functionality (**DataObjectNetLicence** is instanced there). So with current restrictions there should be no connected instance of **DataObjectNetLicence** for the application while generating activation code including network license step. See **DataObjectNetLicence** description for more details.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
uint Licences	Network licenses count value

Public Methods:

Name	Description
StepNetLicence()	Object's constructor.

7.23 StepNetLicenceEx

Extends: ActivationRecordStep
Specifies operation to update DataObjectNetLicence.

MakeActivationCode method of **ActivationSequenceBuilder** class uses CBIOS4NET data objects functionality (**DataObjectNetLicence** is instanced there). So with current restrictions there should be no connected instance of **DataObjectNetLicence** for the application while generating activation code including network license step. See **DataObjectNetLicence** description for more details.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
DONetLicenceExData Licences	Network licenses count value. Only DONetLicenceExData.bNetLic value is used with action DO_INC

Public Methods:

Name	Description
StepNetLicenceEx()	Object's constructor.

7.24 StepNumberOfDays class

Extends: ActivationRecordStep
Specifies operation to update DataObjectNumberOfDays.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
uint Value	Number of days value

Public Methods:

Name	Description
StepNumberOfDays()	Object's constructor.

7.25 StepPasswordHash class

Extends: ActivationRecordStep
Specifies operation to update DataObjectPasswordHash.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
string Value	A password value to calculate hash

Public Methods:

Name	Description
StepPasswordHash()	Object's constructor.

7.26 StepRsa

Extends: ActivationRecordStep
Specifies operation to update DataObjectRsa.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
CBU2RSAKey Key	Key value to set

Public Methods:

Name	Description
StepRsa()	Object's constructor.

7.27 StepRsaEx

Extends: ActivationRecordStep
Specifies operation to update DataObjectRsaEx.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
RsaExDescriptor Descriptor	Descriptor value to set

Name	Description
CBU2RSAKey Key	Key value to set

Public Methods:

Name	Description
StepRsaEx()	Object's constructor.

7.28 StepRunCounter class

Extends: ActivationRecordStep
Specifies operation to update DataObjectRunCounter.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
uint Value	Run counter value.

Public Methods:

Name	Description
StepRunCounter()	Object's constructor.

7.29 StepSignature

Extends: ActivationRecordStep
Specifies operation to update DataObjectSignature.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
CBU2RSAKey KeyA	A key value to set
CBU2RSAKey KeyB	B key value to set
SignatureData SignatureDesc	Signature description

Public Methods:

Name	Description
StepSignature()	Object's constructor.

7.30 StepTimeAllowed class

Extends: ActivationRecordStep
Specifies operation to update DataObjectTimeAllowed.

Public Properties:

Name	Description
ActionType Action	A type of update operation.
uint Value	Time allowed value

Public Methods:

Name	Description
StepTimeAllowed()	Object's constructor.

8. DP4NET: Description of classes

See **DataProtection4MediaFiles** sample to play with DP4NET. The sample can be found here: *<SmarxOS PPK root>\SmarxOS\API\Win\Samples\DataProtection4MediaFiles\C#\MSVS2005 (CBIOS4NET)*

8.1 DP class

This class provides some static functionality, allowing to encrypt (prepare) data files and obtain signatures for encryption key value used.

Public Methods:

Name	Description
static void **EncryptFile** (string SrcFilePath, string DestFilePath, string SignFilePath, DPEncryptionKey EncrKey)	Encrypts specified file with the encryption key. Optionally allows creating a signature file. **Input Parameters:** *SrcFilePath* – Path to a source file to encrypt; *DestFilePath* – result file path; *SignFilePath* – signature file path. Can be null or empty when signature file is not needed; *EncrKey* – encryption key. **Old syntax:** SMRX_DP_EncryptFile.
static byte[ ] **GetSignature**(DPEncryptionKey EncrKey)	Calculates a signature data for specified encryption key. **Returns:** signature data **Input Parameters:** *EncrKey* – encryption key. **Old syntax:** SMRX_DP_GetSignature.

8.2 DPEncryptionKey class

Specifies encryption key value.

Public Properties:

Name	Description
DPEncryptionKeyType KeyType	Encryption key type. AES 256 is the only supported key type. This property is read only.
byte[ ] KeyValue	Key value.

Public Methods:

Name	Description
DPEncryptionKey(byte[ ] keyValue)	Object's constructor **Input Parameters:** *keyValue* – value of the encryption key.

8.3 DPException class

Extends: Exception
Exception class used for data protection exception handling.

Public Properties:

Name	Description
DPErrorCode ErrorCode	Error type. Use DPErrorCode enumeration for comparison. This property is read only.
string Message	Text message representing error info. This property is read only.

Public Methods:

Name	Description
DPException(int dpErrorCode)	Object's constructor **Input Parameters:** *dpErrorCode* – error type. DPErrorCode enumeration contains valid error codes
DPException(int dpErrorCode, Exception innerException)	Object's constructor **Input Parameters:** *dpErrorCode* – error type. DPErrorCode enumeration contains valid error codes; *innerException* – inner exception.

8.4 DPFilter class

This class is used to apply filtering API for encrypted file.

Public Properties:

Name	Description
bool IsActive	This property shows whether filter is started. This property is read only.

Public Methods:

Name	Description
DPFilter Clone()	Method is used to copy DPFilter data.
DPFilter(string encryptedFile, DPEncryptionKey encrKey)	Object's constructor **Input Parameters:** *encryptedFile* – Path to the encrypted file the filter should be applied to; *encrKey* – encryption key.

Name	Description
DPFilter(string encryptedFile, DPEncryptionKey encrKey, string signatureFile)	Object's constructor **Input Parameters:** *encryptedFile* – Path to the encrypted file the filter should be applied to; *encrKey* – encryption key; *signatureFile* – signature file path used for verification purpose.
void Resume()	Resumes filtering. **Old syntax:** SMRX_DP_ResumeFiltering.
void Stop(uint timeout)	Stops filtering for active filter **Input Parameters:** *timeout* –timeout value. **Old syntax:** SMRX_DP_StopFiltering.
void Suspend(uint timeout)	Suspends filtering for active filter **Input Parameters:** *timeout* –timeout value. **Old syntax:** SMRX_DP_SuspendFiltering.

8.5 DPProcess class

Represents the process data protection API should be applied to.

Public Methods:

Name	Description
void AddFilter(DPFilter filter, uint timeout)	Object's constructor **Input Parameters:** *filter* – filter instance that should be applied to the process. Makes filter instance active; *timeout* – timeout value. **Old syntax:** SMRX_DP_StartFiltering.
void AutoEncryptNewFiles(DPEncryptionKey encrKey)	Applies specified encryption key for all newly created files by the process. **Input Parameters:** *encrKey* – encryption key. **Old syntax:** SMRX_DP_AutoEncryptAllNewFiles.
DPProcess(Process process)	Object's constructor **Input Parameters:** *process* – process instance.
void ResumeFiltering()	Resumes filtering for all process data filters. **Old syntax:** SMRX_DP_ResumeAllFilters.
void StopFiltering()	Stops all data filters applied for process. **Old syntax:** SMRX_DP_StopAllFilters.
void SuspendFiltering()	Suspend all applied data filters. **Old syntax:** SMRX_DP_SuspendAllFilters.

9. Support

USA
MARX CryptoTech LP
489 South Hill Street
Buford, GA 30518
USA
www.marx.com

Sales: sales@marx.com
Support: support@marx.com
Phone: (+1) 770-904-0369
Fax: (+1) 678-730-1804
E-Mail: contact@marx.com

Germany
MARX Software Security GmbH
Vohburger Str. 68
85104 Wackerstein
Germany
www.marx.com

Sales: sales-de@marx.com
Support: support-de@marx.com
Phone: +49 (0) 8403 9295-0
Fax: +49 (0) 8403 9295-40
E-Mail: contact-de@marx.com

