SmarxCPP: Object Oriented Smarx®OS API for C++

Developer's Guide

Document: 0-27Oct020(SmarxCPP Developers Guide).odt
Last update: 16 November 2020 by AM
Environment: Microsoft Visual Studio C++ 2013-2019

Executive Summary

*This document defines concept, implementation details and syntax of **SmarxCPP** - object oriented Smarx®OS API C++ library for the CRYPTO-BOX® software protection system.*

SmarxCPP combines/supports the following implementations:

- ***CBIOS and DO API**: Object oriented Smarx OS API implementation giving developers full control over every implementation aspect;*
- ***SmarxAC**: Quick and easy implementation, similar to AutoCrypt. It allows developers with just one call to:*
  - Start periodic license validation (including both local and network scenarios)
  - Add exit event notification (AppExitEvent) with error SmarxException argument
- ***SmarxLicense**: Introduces a higher abstract layer allowing developers with just one call to:*
  - validate license (licensing data objects), including both local and network scenarios
  - add event notifications and customer specific CRYPTO-BOX attach/detach event processing

MARX Software Security GmbH
Vohburger Strasse 68
85104 Wackerstein, Germany
Phone: +49 (0) 8403 / 9295-0
contact-de@marx.com

MARX CryptoTech LP
489 South Hill Street
Buford, GA 30518 U.S.A.
Phone: (+1) 770 904 0369
contact@marx.com

www.marx.com

Table of Contents

1. Introduction

SmarxCPP provides an object oriented API for C++ 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).

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

SmarxCPP also introduces the following higher abstract layers:
- SmarxAC – allows to start periodic license validation (including both local and network scenarios) and add exit event notification
- SmarxLicense - allows developer with one call to validate license and/or this or that licensing data object (including both local and network scenarios), plus adding event notifications and customer specific processing related to CRYPTO-BOX attach/detach events to the program

Sample code for all layers describes above are part of the Professional Protection Kit (PPK) for Windows and the Smarx OS 4 Linux / Smarx OS 4 Mac packages for the CRYPTO-BOX.

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

To start working with **SmarxCPP** 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 – **CryptoboxEvent** 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 the CRYPTO-BOX SC (CBU SC) the unit is actually opened by CryptoboxManager or BoxHandle. The **CbuScCryptobox** 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:
- UseAccessNever – 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;
- UseAccessAlways – this value means free access (no limitations);
- UseAccessUpw – UPW login is required;
- UseAccessApw – APW login is required;
- UseAccessLock – 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,
- CbcEncrypt - encrypt with CBC mode
- CbcDecrypt - 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.

3. SmarxCPP: Description

3.1 IAppInfo

Represents application memory allocation information.
Old syntax: struct CBIOS_APP_INFO

Public Properties:

Name	Description
int AppID	Application ID.
int Ram1	Application RAM1 size.
int Ram2	Application RAM2 size.
int Ram3	Application RAM3 size.

3.2 IBoxHandle

Extends: IBoxInfo
Represents non-opened CRYPTO-BOX.

Public Methods:

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

3.3 IBoxInfo

Represents information about the CRYPTO-BOX.
Old syntax: struct CBIOS_BOX_INFO

Public Methods:

Name	Description
int BoxName	Represents BoxName. **Old syntax:** dwBoxName in CBIOS_BOX_INFO_ADV structure
CBIOS::BoxType BoxType()	Returns a BoxType enumeration - CRYPTO-BOX type: unknown, CBU or CBU SC.
int CbiosVersion()	CRYPTO-BOX CBIOS version info - “0” means: not a SmarxOS formatted CRYPTO-BOX **Old syntax:** wCBIOS in CBIOS_BOX_INFO_ADV structure
int DeveloperID()	DeveloperID (unique value for every MARX customer). **Old syntax:** dwDeveloperID in CBIOS_BOX_INFO_ADV structure

Name	Description
BoxVersion FirmwareVersion()	Firware version. **Old syntax:** bLoVersion and bHiVersion in CBIOS_BOX_INFO_ADV structure
bool IsMPI()	CRYPTO-BOX is MPI formatted. **Old syntax:** bMPI in CBIOS_BOX_INFO_ADV structure
std::array<unsigned char, 8> Label()	CRYPTO-BOX Label. T **Old syntax:** bBoxLabel in CBIOS_BOX_INFO_ADV structure
BoxLoginInfo LoginInfo()	Returns CRYPTO-BOX current status: Admin or user login. **Old syntax:** bLogin in CBIOS_BOX_INFO_ADV structure
BoxRAMSize MemorySize()	CRYPTO-BOX RAM size. See BoxRAMSize struct for details. **Old syntax:** dwRAMLen and dwSize_RAMx in CBIOS_BOX_INFO_ADV structure
BoxModelInfo ModelInfo()	Returns BoxModelInfo enumeration - CRYPTO-BOX Model: Unknown, CBU, CBUSC. **Old syntax:** dwModel in CBIOS_BOX_INFO_ADV structure
std::array<unsigned char, 16> SerialNumber()	CRYPTO-BOX serial number. **Old syntax:** CBIOS_GetSerialNum

3.4 BoxRAMSize struct

Represents memory size of CRYPTO-BOX.

Public Properties:

Name	Description
Int Ram1	Memory size allocated for RAM1.
int Ram2	Memory size allocated for RAM2.
int Ram3	Memory size allocated for RAM3.
int Ram4	Memory size allocated for RAM4.
int Ram5	Memory size allocated for RAM5.
int Ram	CRYPTO-BOX memory size.

3.5 BoxVersion struct

Represents CRYPTO-BOX firmware version.

Public Properties:

Name	Description
int HiFwVersion	Firmware version High (1 for 1.6).
int LowFwVersion	Firmware version Low (6 for 1.6).
string Version	Firmware string representation (1.6).

3.6 CbuScAESKeyInfo struct

Represents AES key info.

Public Properties:

Name	Description
CbuScKeyAccess access	Key access parameters. Set and use rights. See CbuScKeyAccess enumeration
int iength	Key length

3.7 CbuScRsaKeyInfo struct

Represents RSA key info.

Public Properties:

Name	Description
CbuScKeyAccess access	Key access parameters. Set and use rights. See CBU2KeyAccess enumeration
int bits	Key length

3.8 CryptoboxEventInfo struct

Represents Cryptobox Event.

Public Properties:

Name	Description
CryptoboxEventType type	Event type
int id	Event ID

3.9 ICryptobox interface

Extends: IBoxInfo
Represents Cryptobox.

Public Methods:

Name	Description
AppLicensesRule CheckAppLicenses()	Retrieves information about the free (not busy) local and network licenses of an application. **Returns:** Application licenses amount.
void CheckBox()	Checks for the presence of currently opened CRYPTO-BOX. **Old syntax:** CBIOS_CheckBox
std::vector<unsigned char> CryptFixed(const std::vector<unsigned char>& 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

Name	Description
std::vector<unsigned char> CryptPrivate(const std::vector<unsigned char>& 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
std::vector<unsigned char> CryptSession(const std::vector<unsigned char>& data)	Encrypts/decrypts data using internal hardware algorithm with Session Key and Session Initialization Vector. **Returns:** Encrypted/decrypted data **Old syntax:** CBIOS_CryptSession
AppLicensesRule GetAppLicences()	Reads information on application's local and network licenses from the LMT (taking into account extended network sharing rule). **Returns:** Application licenses amount.
void Login(CryptoboxLoginType type, const std::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. **Old syntax:** CBIOS_LoginUPW(), CBIOS_LoginAPW()
void Logout()	Performs logout **Old syntax:** CBIOS_Logout
std::vector<unsigned char> ReadRAM(RamType 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
std::vector<unsigned char> RsaDecryptInternal(RsaKeyType type, const std::vector<unsigned char>& data)	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; **Returns**: Decrypted data. **Old syntax:** CBIOS_ DecryptInternalRSA
std::vector<unsigned char> RsaEncryptInternal(RsaKeyType type, const std::vector<unsigned char>& data)	Encrypts 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; **Returns**: Encrypted data. **Old syntax:** CBIOS_ EncryptInternalRSA

Name	Description
void SetAppLicenses(const AppLicensesRule &ruleAppLic, const std::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 SetAppLicenses(const AppLicensesRule &ruleAppLic, const std::array<unsigned char, 16> &password)	**Input Parameters:** *licenses* – number of licenses; *password* – admin password. **Old syntax:** CBIOS_SetAppLicences Deprecated method use **SetAppLicences**
void SetPrivateKey(const IRijndaelCryptKey &key, const std::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
SetPassword(CryptoboxLoginType type, const std::string &oldPassword, const std::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 WriteRAM(RamType ram, const std::vector<unsigned char>& data, int offset, const std::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

3.10 ICbuScCryptobox interface

Extends: ICryptobox
This interface is used to call CRYPTO-BOX SC (CBU SC) methods.
This interface 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
std::vector<unsigned char> CryptAes(int keyIndex, CbuScAesKeyMode mode, const std::vector<unsigned char>& IV, const std::vector<unsigned char>& data)	Encrypts/decrypts data with specified AES key **Returns:** encrypted/decrypted data. **Input Parameters:** *keyIndex* – index of a stored key to use; mode – mode of a key; IV – initialization vector for AES key; data – input data.

Name	Description
std::vector<unsigned char> DecryptRsa(int keyIndex, CbuScRsaKeyMode mode, const std::vector<unsigned char>& data)	Decrypts data with specified RSA key **Returns:** decrypted data. **Input Parameters:** *keyIndex* – index of a stored key to use; mode – mode of a key; data – input data.
std::vector<unsigned char> EncryptRsa(int keyIndex, CbuScRsaKeyMode mode, const std::vector<unsigned char>& data)	Encrypts data with specified RSA key **Returns:** encrypted data. **Input Parameters:** *keyIndex* – index of a stored key to use; mode – mode of a key; data – input data.
void SetKeyAes(int keyIndex, const std::vector<unsigned char>& aesKey, const CbuScAesKeyInfo &aesKeyInfo)	Used to set AES key to box storage. **Input Parameters:** *keyIndex* – index of a key; aesKey – key to set; aesKeyInfo – corresponding key info.
void SetKeyRsa(int keyIndex, const ICbuScRsaKey &rsaKeyPair, const CbuScRsaKeyInfo &rsaKeyInfo)	Used to set RSA key to box storage. **Input Parameters:** *keyIndex* – index of a key; rsaKeyPair – key to set; rsaKeyInfo – corresponding key info.
CbuScAesKeyInfo GetKeyInfoAes(int keyIndex)	Gets key info for a stored AES key **Returns:** key info. **Input Parameters:** *keyIndex* – index of a stored key.
CbuScRsaKeyInfo GetKeyInfoRsa(int keyIndex)	Gets key info for a stored RSA key **Returns:** key info. **Input Parameters:** *keyIndex* – index of a stored key.
void LockKeyAES(int keyIndex)	Locks specified AES key **Input Parameters:** *keyIndex* – index of a stored key.
void LockKeyRSA(int keyIndex)	Locks specified RSA key **Input Parameters:** *keyIndex* – index of a stored key.
void SetKeyInfoAES(uint keyIndex, CBU2AESKeyInfo pAESKeyInfo)	Sets key info for a stored AES key **Input Parameters:** *keyIndex* – index of a stored key; *aesKeyInfo* – key info value
void SetKeyInfoAes(int keyIndex, const CbuScAesKeyInfo &aesKeyInfo)	Sets key info for a stored RSA key **Input Parameters:** *keyIndex* – index of a stored key; *aesKeyInfo* – key info value

3.11 IDataObject interface

Interface for DataObject API.

Public Methods:

Name	Description
int GetOffset()	Get DataObject's offset in partition where it is stored.
void SetOffset(int offset)	Set DataObject's offset in partition where it is stored.
CBIOS::RamType GetRam()	Get RAM of partition where DataObject is stored.
void SetRam(CBIOS::RamType ramType)	Set RAM of partition where DataObject is stored.
DOType GetType()	Get DataObject type

3.12 ICDOSettable interface

Interface for DataObject API.

Methods:

Name	Description
void Set(const std::vector<unsigned char>& signature)	Set CDO value **Input Parameters:** *signature* –DataObject signature **Old syntax:** TEOS_DOSet

3.13 ICDOIncrementable interface

Interface for DataObject API.

Methods:

Name	Description
void Inc(const std::vector<unsigned char>& signature)	Increment CDO value **Input Parameters:** *signature* –DataObject signature **Old syntax:** TEOS_DOInc

3.14 ICDOUnlimitable interface

Interface for DataObject API.

Methods:

Name	Description
void Unlimited(const std::vector<unsigned char>& signature)	Set unlimited value for CDO **Input Parameters:** *signature* –DataObject signature **Old syntax:**TEOS_DOSet

3.15 IDOClearable interface

Interface for DataObject API.

Methods:

Name	Description
void Clear()	Clears a DO value from the CRYPTO-BOX partition **Old syntax:**TEOS_DOClear

3.16 IDOUnlimitable interface

Interface for DataObject API.

Methods:

Name	Description
void Unlimited()	Set unlimited value for DO **Old syntax:**TEOS_DOSet

3.17 IDOIncrementable interface

Interface for DataObject API.

Methods:

Name	Description
void Inc()	Increment DO value. **Old syntax:**TEOS_DOInc

3.18 IDODecrementable interface

Interface for DataObject API.

Methods:

Name	Description
void Dec()	Decrement DO value **Old syntax:**TEOS_DODec

3.19 IDOVerifiable interface

Interface for DataObject API.

Methods:

Name	Description
void Verify()	Verify DO value. Throws exception if verification failed **Old syntax:**TEOS_DOVerify

3.20 IDOGettable interface

Interface for DataObject API.

Methods:

Name	Description
Svalue Value()	Get DO value **Old syntax:**TEOS_DOGet

3.21 IDOPartialBase interface

Extends: IDataObject, IDOClearable, IDOUnlimitable
Interface for DataObject API.

3.22 IDOBase interface

Extends: IDOPartialBase, IDOIncrementable, IDODecrementable, IDOVerifiable
Interface for DataObject API.

3.23 ICDOBase interface

Extends: IDataObject, IDOClearable, ICDOUnlimitable, ICDOIncrementable, IDODecrementable, IDOVerifiable, IDOGettable
Interface for DataObject API.

3.24 ICDOIntBase interface

Extends: IDataObject, IDOClearable, ICDOUnlimitable, IDODecrementable, IDOGettable
Interface for DataObject API.

Methods:

Name	Description
void Set(const std::vector<unsigned char>& signature)	Set CDO value **Input Parameters:** *signature* –DataObject signature **Old syntax:** TEOS_DOClear
void Get(const std::vector<unsigned char>& signature, const std::array<unsigned char, 16> &password)	Get CDO value **Input Parameters:** *signature* –DataObject signature *password* –APW byte array **Old syntax:** TEOS_DOSet
void Inc(const std::vector<unsigned char>& signature)	Increment CDO value **Input Parameters:** *signature* –DataObject signature **Old syntax:** TEOS_DOInc
void Dec()	Decrement CDO value **Old syntax:** TEOS_DODec

3.25 ICryptoBinding interface

Extends: IDataObject
Represents Binding DataObject.

Public Methods:

Name	Description
int GetStatus()	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 Activate(const std::vector<unsigned char> &activation)	Activates dataobject. Status – Activated. **Input Parameters:** *activation* – activation sequence.(activation request signed with private distributor rsa key).
void Bind()	Binds dataobject to PC hardware. Status – Bound.
std::vector<unsigned char> GenerateActivation()	Generates activation code that can be used to activate dataobject. **Returns**: activation code

3.26 ICryptoExpirationDateTime interface

Extends: IDataObject
Represents expiration date DataObject.

Public Methods:

Name	Description
int SecondsLeft()	Seconds left. **Old syntax:** CBIOS_DOSet, CBIOS_DOGet
time_t ExpirationTime()	Expiration time. **Old syntax:** CBIOS_DOSet, CBIOS_DOGet
void Dec(uint time)	Decreases expiration datetameby specified number of seconds. **Input Parameters:** *time* – number of seconds to decrement expiration date. **Old syntax:** CBIOS_DODec
virtual void Inc(int time, const std::vector<unsigned char>& signature) = 0;	Increases expiration date by specified number of seconds. **Input Parameters:** *time* – number of days to decrement expiration date. *signature* –DataObject signature **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 Set(const time_t &expirationTime, const std::vector<unsigned char>& signature)	Set espiration datatim **Input Parameters:** *expirationTime* – Unix timestamp, seconds. *signature* –DataObject signature **Old syntax:** CBIOS_DOSet
void Set(const std::string &expirationTime, const std::vector<unsigned char>& signature)	Set espiration datatim **Input Parameters:** *expirationTime* – datetime string (dd MMM yyyy HH:mm:ss) *signature* –DataObject signature **Old syntax:** CBIOS_DOSet

3.27 IMemory interface

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

Methods:

Name	Description
int GetSize()	Get memory size allocated for memory object
void SetSize(int size)	Set memory size **Input Parameters:** *size* – size of memory block
std::vector<unsigned char> Get(int size, int offset)	Reads data from box. null offset assumed + DataObject's offset. **Input Parameters:** *size* – size of memory block to read. *offset* – memory offset.
void Set(const std::vector<unsigned char> &value, int 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.

3.28 ICDODays interface

Extends: ICDOIntBase, ICDOBase
Represents number of days DataObject.

3.29 ICryptoPasswordHash interface

Extends: IDataObject
Represents password hash DataObject.

Public Methods:

Name	Description
void Set(const std::string &value, const std::vector<unsigned char>& signature)	Calculates password's CRC and stores it to partition. **Input Parameters:** *value* – password's value; *signature* –DataObject signature
void Verify(const std::string &value)	Compares calculated password's CRC to stored hash value. Returns "true" on success. **Input Parameters:** *hash* – password's value.

3.30 ICryptoCounter interface

Extends: ICDOIntBase
Represents run counter DataObject.

3.31 CDOSignature class

Represents signature DataObject.

Public Methods:

Name	Description
static std::vector<unsigned char> Create(int boxName, const CBIOS::ISmarxRsaKey &rsaKeyPair)	Calculates cryptobox digital signature **Input Parameters:** *boxName* – cryptobox name; *rsaKeyPair* – RSA key that is used as encryption key . **Returns**: signature
static std::vector<unsigned char> Create(int value, int boxName, const CBIOS::ISmarxRsaKey &rsaKeyPair)	Calculates digital signature for int value **Input Parameters:** *value* – int value; *boxName* – cryptobox name; *rsaKeyPair* – RSA key that is used as encryption key . **Returns**: signature
static std::vector<unsigned char> Create(const std::vector<unsigned char> &value, int boxName, const CBIOS::ISmarxRsaKey &rsaKeyPair)	Calculates digital signature for array **Input Parameters:** *value* –data to be signed; *boxName* – cryptobox name; *rsaKeyPair* – RSA key that is used as encryption key . **Returns**: signature
static std::vector<unsigned char> Create(const std::string &value, int boxName, const CBIOS::ISmarxRsaKey &rsaKeyPair)	Calculates digital signature for string **Input Parameters:** *value* –string to be signed; *boxName* – cryptobox name; *rsaKeyPair* – RSA key that is used as encryption key . **Returns**: signature
static std::vector<unsigned char> Create(const STime &value, int boxName, const CBIOS::ISmarxRsaKey &rsaKeyPair)	Calculates digital signature for Stime structure **Input Parameters:** *value* –StTme object; *boxName* – cryptobox name; *rsaKeyPair* – RSA key that is used as encryption key . **Returns**: signature

3.32 ICryptoTimeAllowed interface

Extends: ICDOIntBase
Represents time allowed DataObject.

Public Methods:

Name	Description
bool Verify(int value)	Verifies whether data object is valid for specified number of seconds and decreases data object's value by that number. **Input Parameters:** *value* – decrement value(seconds).

3.33 ICryptoboxManager interface

Interface which helps manage attached CRYPTO-BOX units.

Public Methods:

Name	Description
std::shared_ptr<ICryptobox> OpenByIndex(int boxIndex)	Open box by index. **Input Parameters:** *boxIndex* – box index. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByIndex
std::shared_ptr<ICryptobox> OpenByIndex(int boxIndex, int appID)	Open CRYPTO-BOX by index with selected Application ID. **Input Parameters:** *boxIndex* – box index; *appID* – application id. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByIndex
std::shared_ptr<ICryptobox> OpenByName(int boxName)	Open box by box name. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByLabel
std::vector<std::unique_ptr<IBoxHandle >> ScanBoxes()	Scan for attached boxes. **Returns**: an array of found CRYPTO-BOX handles **Old syntax:** CBIOS_ScanBoxes

3.34 IlocalCBManager interface

Extends: ICryptoboxManager
Interface which helps manage local CRYPTO-BOX units.

3.35 INetworkCBManager interface

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

Public Methods:

Name	Description
void Connect(const std::string &server, int 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
std::shared_ptr<IAsyncAction> ConnectAsync(const std::string &server, int port)	Connects a manager to a network server asynchronously. **Input Parameters:** *server* – server's ip address/name; *port* – port listened by server. **Old syntax:** CBIOS_Connect
bool Disconnect();	Disconnect from currently connected network server. **Old syntax:** CBIOS_Disconnect
bool IsConnected()	Is this manager connected to remote server. This property is read only.
std::string ServerName()	Server name
std::string ServerPort()	Server port

3.36 IPartition interface

Extends: IAppInfo

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 Methods:

Name	Description
void Connect(std::shared_ptr<CBIOS::ICryptobox> 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 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(const std::istream &inputStream)	Loads previously saved data objects map of partition from binary data. Partition should be connected to use this method. **Input Parameters:** *inputStream* – previously stored data object's map data input stream. **Old syntax:** TEOS_DoLoadMap
void Save(std::ostream &outputStream)	Stores currently specified data objects map to a binary array. **Input Parameters:** *outputStream* –data object's map data output stream. **Old syntax:** TEOS_DoSaveMap
bool IsConnected()	Is this partition connected to a CRYPTO-BOX with opened partition. See Connect method for datails. This property is read only.
void Add(std::shared_ptr<IDataObject> dataObject)	Add data object to partition **Input Parameters:** *dataObject* –data object to be added

3.37 IRijndaelCryptKey interface

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

Public Methods:

Name	Description
void SetKey(const std::vector<unsigned char> & key)	Set key **Input Parameters:** *key* –AES key.
void SetIV(const std::vector<unsigned char> & iv)	Set initialization vector **Input Parameters:** *iv* –initialization vector.

3.38 ISmarxRsaKey interface

Base RSA interface

Public Methods:

Name	Description
void Set(const std::vector<unsigned char>& modulus, const std::vector<unsigned char>& exponent)	Set RSA key **Input Parameters:** *modulus* – RSA key modulus *exponent* – RSA key exponent
std::vector<unsigned char> PublicEncryptMarxPadding(const std::vector<unsigned char>& data)	Encrypt data using this RSA key **Input Parameters:** *data* – data to be encrypted;; **Returns:** encrypted data **Old syntax:** CBIOS_EncryptRSAEx
std::shared_ptr<ICryptobox> OpenByName(int boxName)	Open box by box name. **Returns:** opened CRYPTO-BOX **Old syntax:** CBIOS_OpenByLabel
std::vector<std::unique_ptr<IBoxHandle >> ScanBoxes()	Scan for attached boxes. **Returns**: an array of found CRYPTO-BOX handles **Old syntax:** CBIOS_ScanBoxes

3.39 AppLicensesRule struct

Rule+ LicensesCount.

Public Properties:

Name	Description
int LicensesCount	Number of Net licenses
LicenseRule Rule	RuleId from LicenseRule enumeration

3.40 ServerInfo struct

Class represents information about network server.

Public Properties:

Name	Description
int BoxCount	Number of boxes connected to network server.
string Name	Server's name.
int Port	Network port listened by server.

3.41 ISmarxLicense interface

Interface to license file generated by SxAF for AutoCrypt project. Allows to validate license and/or this or that licensing data object (including both local and network scenarios), add event notifications and customer specific processing related to MARX hardware events to the program

Public Methods:

Name	Description
void LoadFile(const std::string &licensePath)	Load the license file (Generated with SxAF) **Input Parameters:** *licensePath* –path to license file.
void LoadString(const std::string &license)	Load the license content from string **Input Parameters:** *license* –license content. **IMPORTANT:** When you try to parse a string which is not UTF-8 encoded, our signature will be broken and will lead to “Corrupted Data” exception. To mark string as UTF-8 encoded, you need to place u8R before raw string instead of R. See also here: https://docs.microsoft.com/en-us/cpp/cpp/string-and-character-literals-cpp?view=msvc-160
void Validate(const std::string &licenseID)	Validate the license ID(get LicenseIDs from corresponding license map file that is generated by SxAF along with license file) **Input Parameters:** *licenseID* –license ID.
void Validate()	Validate all license IDs
SValue GetValue(const std::string &licenseID)	Gets a value of the license ID **Input Parameters:** *licenseID* –license ID.
const std::type_info& GetValueType(const std::string &licenseID)	Gets a value type of the license ID **Input Parameters:** *licenseID* –license ID.
DialogParams GetDialogParams(const std::string& dialogType)	Gets a value of the dialog strings **Input Parameters:** *dialogType* –dialogType.
CBIOS::ICryptoboxManager::CryptoboxEventRemover operator +=(const CBIOS::ICryptoboxManager::CryptoboxEvent &)	Subscribe to CryptoboxEvent event
void operator -=(CBIOS::ICryptoboxManager::CryptoboxEventRemover)	Unsubscribe from CryptoboxEvent event

3.42 ISmarxAc interface

Interface to license file generated by SxAF for AutoCrypt project. Quick and easy implementation, similar to AutoCrypt.

Public Methods:

Name	Description
void StartFromFile(const std::string &licensePath, bool isWait = false)	Load the license file (Generated with SxAF) **Input Parameters:** *licensePath* –path to license file, *isWait* –wait for the first license validation.

Name	Description
void StartFromString(const std::string &license, bool isWait = false)	Load the license content from string **Input Parameters:** *license* –license content, *isWait* –wait for the first license validation.
AppExitRemover operator +=(const AppExitEvent &)	Subscribe to ApplicationExit event **Input Parameters:** *AppExitEvent* –typedef std::function<void(const SmarxException&)> AppExitEvent
void operator -=(AppExitRemover)	Unsubscribe from ApplicationExit event **Input Parameters:** *AppExitRemover* –typedef std::shared_ptr<AppExitEvent> AppExitRemover

3.43 Smarx class

Provides static functions for SmarxAPI objects creation

Public Methods:

Name	Description
static int libraryVersion()	Returns SmarxCPP library version
static std::string libraryVersionStr()	Returns SmarxCPP library version
static std::shared_ptr<CBIOS::INetworkCBManager> CreateNetworkCBManager()	Creates Network Crypto-box manager **Returns**: object that implements INetworkCBManager interface
static std::shared_ptr<CBIOS::ILocalCBManager> CreateLocalCBManager()	Creates Local Crypto-box manager **Returns**: object that implements ILocalCBManager interface
static std::unique_ptr<CBIOS::ISmarxNetworkScanner> CreateNetworkScanner()	Creates network scanner **Returns**: object that implements ISmarxNetworkScanner interface
static std::unique_ptr<CBIOS::ICbuScRsaKey> CreateCbuScRsaKey()	Create RSA key object for CBU SC **Returns**: object that implements ICbuScRsaKey interface
static std::unique_ptr<CBIOS::ISmarxRsaKey> CreateSmarxRsaKey()	Create RSA key object **Returns**: object that implements ISmarxRsaKey interface
static std::unique_ptr<CBIOS::IRijndaelCryptKey> CreateRijndaelCryptKey()	Create AES key object **Returns**: object that implements IRijndaelCryptKey interface
static std::shared_ptr<CBIOS::ICryptobox> OpenBox(const std::string& uriBox)	Open cryptobox by URI(cryptobox name) **Returns**: object that implements ICryptobox interface
static std::shared_ptr<CBIOS::ICryptobox> OpenBox(const std::string& uriBox, int appID)	Open cryptobox by URI(cryptobox name) and application on opened cryptobox **Returns**: object that implements ICryptobox interface
static std::unique_ptr<CBIOS::ISearchOptions> CreateSearchOptions()	Create SearchOptions object **Returns**: object that implements ISearchOptions interface
static std::unique_ptr<CBIOS::ISearch> CreateSearch()	Create Search object **Returns**: object that implements ISearch interface
static std::shared_ptr<DO::IPartition> CreatePartition()	Create partition for DataObjects **Returns**: object that implements IPartition interface

Name	Description
static std::shared_ptr<DO::IDataObject> CreateDataObject(DO::DOType type)	Create Data Object **Returns**: object that implements IDataObject interface
static std::unique_ptr<Protection::ISmarxLicense> CreateSmarxLicense()	Create SmarxLicense object **Returns**: object that implements ISmarxLicense interface
static std::unique_ptr<Protection::ISmarxAc> CreateSmarxAc()	Create SmarxAC object **Returns**: object that implements ISmarxAc interface

3.44 List of Enumerations

Name	Values
BoxType	Unknown Cbu CbuSc
BoxModelInfo	Versa XS XL
BoxLoginInfo	UserLogin AdminLogin
CryptoboxLoginType	Admin User
RamType	Ram1 Ram2 Ram3
CryptoboxEventType	BoxAttached BoxRemoved BoxesChanged
RsaKeyType	InternalDistrPubl InternalClientPriv MarxPadding RsaRefPadding
LicenseRule	Default - licenses are locked without taking client's IP/Terminal Session into account (default behavior as implemented now), IP1license - "one seat per one IP", only one license will be locked for all calls coming from one IP address without notice on TS (unlimited number of instances can be launched concurrently from one IP) MacAddrTS1license - "one seat per one <MAC+User+Add>" for dynamic IP, only one license will be locked for all calls coming from: one MAC address (unlimited number of instances can be launched concurrently from one MAC), one Terminal Session (unlimited number of instances can be launched concurrently in the same TS) and additional checking (soft&hard components) IPAddrTS1license - "one seat per one <IP+MAC+User+Add>" for static IP, the same as MacAddrTS1license + IP
CbuScRsaKeyMode	PublKey PrivKey MarxPadding RsaRefPadding

Name	Values
CbuScAesKeyMode	Ofb, CbcEncrypt CbcDecrypt
CbuScKeyAccess	UseAccessNever UseAccessAlways UseAccessUpw UseAccessApw UseAccessLock SetAccessNever SetAccessAlways SetAccessUpw SetAccessApw SetAccessLock
BindingStatus	Unbound Bound Activated
DOType	CryptoExpirationDateTime CryptoExpirationDateRelative CryptoNumberOfDays CryptoTimeAllowed CryptoCounter CryptoMemory Memory CryptoPasswordHash CryptoClientBinding CryptoServerBinding CryptoClientGeoLicense CryptoServerGeoLicense,

4. Technical 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
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
E-Mail: contact-de@marx.com

