SmartCardAPI SDK

rev 18JAN2021

Generated by Doxygen 1.8.10

Mon Jan 18 2021 10:22:02

Contents

Chapter 1

Overview

This is the API reference for SmartCard API (Professional). For a general framework overview please refer to the developer guide.

1.1 Version History

This section contains important information about SmartCardAPI version history. The current version of SmartCard API (Professional) SDK is: 5.0.21.117

1.1.1 Version 5.0.21.117 (rev.17JAN2021)

- faster reader detection
- binaries signed with SHA1 and SHA2
- change: improve Windows error code processing
- change: remove wait time after terminal detection
- new: PIV FASC-N decoder class
- new: DESFire: Value processing
- new: Picopass read/write
- fix: Startup(false) overwrites card terminal registry
- fix: DESFire: writing over 256 bytes fails
- fix: DESFire: invalid MAC when reading multiple frames
- fix: DESFire: Read() plain file after AES authentication fails
- fix: de/reconnecting reader may trigger unhandled exception

1.1.2 Version 5.0.20.304 (rev.04MAR2020)

- new: PIVcrypto implementation
- new: PIV SHA256 signature

1.1.3 Version 5.0.20.131 (rev.31JAN2020)

- added SDK documentation CardModule.PIV and CardModule.CAC

1.1.4 Version 5.0.20.120 (rev.20JAN2020)

- replaces public release 5.0.19.1204
- removed installer; all distributions ZIP-based
- builds for .NET framweork version 3.5, 4.5, 4.6.1
- sample code used binaries built against .NET 4.5
- changed main reference binaries and sample code from 4.6.2 due to incompatibility with some Windows 10 images
- all binaries digitally signed by CardWerk Technologies
- smartcard.dll file properties reflect .NET framework version

1.1.5 Version 5.0.19.1204 (rev.04DEC2019)

- replaces public release 5.0.19.1029
- removed installer; all distributions ZIP-based
- built against .NET framweork version 4.6.2
- added DESFire EV1 sample code to SDK distribution
- added EMV card sample code to SDK distribution

1.1.6 Version 5.0.19.1029 (rev.29OCT2019)

- replaces public release 5.0.17.602
- renamed sample code - only one "Hello" sample remaining and that's HelloCard
- sample code now shows usage of lost/found terminal
- each sample code folder contains its local copy of libraries necessary for this sample
- new: CardModule.MemoryCard supports OMNIKEY 5122/5422
- fix: pcsc-level error "access denied" during establish context
- new: DESFire EV1 in legacy mode: DES with weak crypto keys supported; TDES support
- new: DESFire EV1 Authenticate() FormatPICC(), CreateApplication() via CardModule.DESFire
- new: DESFire EV1 AES encrypted read/write
- fix: Windows 2008 Server RDP session dis/reconnect triggers PC/SC Win32 error
- new: MemoryCard: access to SLE4442 protected memory
- fix: CardModule.ICLASS GetRawPacsBit() does not close secure session
- new: CardModule.ICLASS SEOS support on OMNIKEY 5023 and OMNIKEY 5x27CK readers

1.1.7 Version 5.0.17.602

- replaces public release 4.9.6.0811
- new: all assemblies are built against .NET framework version 4
- new: HelloCard.Console sample to make it even easier to become familiar with SmartCardAPI
- new: CardModule.MemoryCard to access synchronous cards on HID OMNIKEY readers (Xchip and Aviator support)
- new: CardModule.NfcTag to access Mifare Ultralight, NTAG
- new: HID iCLASS support via OMNIKEY 5022 and OMNIKEY 5x27 CK readers (avail. upon request)
- new: move CAC specific sample code to HelloCAC sample code
- new: Windows Help file now also available as PDF document "SmartCardApi.ApiDoc.pdf"
- new: version info reflects year, month and day it was built; 5.0.17.602 stands for version 5.0 built 02JUN2017

1.1.8 Version 4.9.6.0811 (rev.18AUG2015)

- replaces public release 4.9.3.1030
- fix: broken links to CardModule and SmartCardAPI references in HelloPIV sample code
- fix: introduce ReaderModules, DataModules, SecurityModules for better code separation
- new: ReaderModule.Omnikey to support memory cards and iCLASS cards via OMNIKEY proprietary DLLs & APDUs
- new: DataModule.Wiegand for greatly improved PACS card decoding of raw Wiegand data
- new: CardModule.HID.PROX to access HID PROX cards on HID OMNIKEY readers
- new: CardModule.HID.ICLASS to access HID iCLASS cards on HID OMNIKEY 5x21 readers (upon request)
- new: CardModule.Mifare to access NXP Mifare classic cards on PC/SC part 3 compliant contactless card readers
- fix: introduce 0x648x SW for readers that respond with I/O errors to unsupported commands
- new: all modules are built with strong names
- new: sample: HelloiCLASS (upon request)
- new: GetInstructionName() supports PC/SC part 3 instruction name
- change: HelloSyncAPI renamed to HelloMemoryCard
- change: new installer
- new: using Doxygen to generate API documentation
- new: API documentation available in PDF format

1.1.9 Version 4.9.3.1030 (rev.30OCT2014)

- replaces public release 4.9.2.0
- fix: support card terminals that return unexpected PC/SC errors with some attribute requests
- new: CardModule.xxx built with strong names

1.1.10 Version 4.9.2.0 (rev.02JAN2014)

- replaces public release 4.9.0.1
- fix: EndTransaction accepts if card or reader handle are already null (no more exception thrown)
- new: iCLASS PACS bit access via OMNIKEY 5x21 and sync API
- new: sample code with transaction protection
- new: sample programs handle reader, smart card service recovery
- new: HelloProx support for OMNIKEY 5427 CK multi frequency reader

1.1.11 Version 4.9.0.1 (rev.21NOV2013)

- replaces public release 4.8.1.0
- new: Smartcard.dll with new terminal found event
- new: Smartcard.dll terminal polling after singleton start until at least one terminal found
- new: Smartcard.dll offers transaction protection
- new: HelloPIV with protected card access
- new: HelloProx support for OMNIKEY 5025 PROX card reader in CCID mode

1.1.12 Version 4.8.1.0 (rev.05AUG2013)

- replaces public release 4.8.0.0
- fix: Smartcard.dll: CardHex.FromArray() truncates LSB bytes causing invalid raw Wiegand data
- new: HelloProx with CardModule.CLICS integration for PC/SC Part 3 compliant reader access

1.1.13 Version 4.8.0.0 (rev.15MAY2013)

- replaces public release 4.7.2.3
- new: Smartcard.dll: CardPcscPart3 and PCSC_PART3_XYZ constants for PC/SC 2.01 Part 3 compliant card system access
- new: HelloPIV: GetData() now supports T=1 cards that don't return data with the first response APDU
- new: CardModule.CLICS with card edge for PC/SC compliant contactless storage cards. Ideal for Mifare Classic, Ultralight
- new: HelloMifare: sample code for Mifare card now accesses care reader via CardModule.CLICS

1.1.14 Version 4.7.2.3 (rev.06FEB2013)

- replaces public release 4.7.2.2
- HelloPIV: add X509 certificate access, card capability container
- CardModule.PIV: add PivCardCertificate() class to read and display X509 certificates
- CardModule.PIV: add PivCardCapability() class to read and display card capability container
- CardModule.PIV: add BER-TLV tags for PIV data all containers

1.1.15 Version 4.7.2.2

- replaces public release 4.7.2.1
- smartcard.dll: supports systems that don't allow creation of registry.xml configuration file

1.1.16 Version 4.7.2.1

- replaces public release 4.7.2.0
- HelloPIV: DoD CACv2 support
- CardModule.PIV/CAC: change SelectCardApplication() to SelectApplet() to be in line with PIV/CAC specs
- setup: add HelloProx
- setup: add link to run HelloProx
- setup: smartcard.dll: add method to extract bit sequence from byte array to support cards with Wiegand data

1.1.17 Version 4.7.2.1

- replaces public release 4.7.1.6
- HelloPIV: DoD CAC support
- SmartCard.dll: move PIV related code to CardModule.PIV
- CardModule.PIV: introduce SelectCardApplication() to allow PIV module to explore TWIC/CAC cards
- setup: fix link to SmartCardAPI documentation
- setup: add link to run HelloCard

1.1.18 Version 4.7.1.6

- replaces public release 4.7
- HelloPIV: demonstrate PIN and printed info access
- SmartCard.dll: stopping scardsvr triggers terminal lost event
- HelloXYZ: SmartCard.dll reference to lib folder

1.1.19 Version 4.7

- replaces public release 4.6
- PIV card support: CHUID, FASCN, GUID,GUID
- Samples names start with Hello
- Add HelloCard Sample (C#, VB.NET)
- Rename MctTrivial to HelloMct
- Rename SimTrivial to HelloSim
- Rename Taschenkartenleser to HelloGeldkarte

1.1.20 Version 4.6

No major changes. Copyrights transferred to CardWerk Technologies.

- replaces public release 4.5

1.1.21 Version 4.5

- replaces public release 4.3
- Extended the MctTrivia sample application to also handle the new German eGK Gesundheitskarte.

1.1.22 Version 4.3

- replaces public release 4.2
- Added CardPcScTerminalBase.ATR property for easier access to the connected card's ATR.
- Removed methods that launch the CardTerminalConfigurator.exe.
- Fixed problems wit MCARD API 4.0 and later.

1.1.23 Version 4.2

- replaces public release 4.1
- new: property CardPcScTerminal.SharedAccess controls whether the native PC/SC card connection will be Shared or Exclusive.
- new: REG_SZ registry value [HKLM] SharedAccess controls the default access mode. If, and only if, this registry value is "1", then the default access mode is Shared access.

1.1.24 Version 4.1

- replaces public release 4.0
- improved: Greatly simplified the CardTerminalSlot state tracking. Now the CardTerminalSlotState enumeration only has the states Empty, Inserted and Accepted. This improves the realiability of the Card Inserted and Card Removed events. Now these events are also reliably fired when a card reader is physically disconnected or connected.
- removed: fRetryCards parameter from CardTerminalManager.RequestCard method as this is no longer needed by the simplified card reader state tracking.
- improved: ATR handling and added CardHandle.GetHistoricalBytes method.
- new: CardHandle.SelectDirectory method.
- new: overloaded CardCommandAPDU.AddData(byte) method.
- Improved automatic GET RESPONSE handling in CardHandle class in order to deal with GSM SIM cards.
- fix: GSM SIM status codes are correctly interpreted in CardResponseAPDU class.
- improved CardTerminalRegistry.AutoRegisterPcSc method such that it now also automatically eliminates any entries of smart card readers that no longer exist in the system.

1.1.25 Version 4.0

This is a major update which new features a complete rework of the code base. The most prominent new feature is the support for PC/SC Workgroup specification 2.0.1 part 10, which includes Secure PIN Entry (SPE).

- replaces public release 3.3
- Support for secure PIN entry according to the PC/SC Workgroup specification 2.0.1 part 10. Secure PIN entry is supported through all means defined in this specification.
- The previously internal class CardTerminalSlot is now public and is now the primary means for acquiring a Card←
Handle. Events have been changed to supply a CardTerminalSlot reference instead of a ICardTerminal reference and a slot number. Generally this makes handling card insertions and removals much more convenient and straightforward.
- The CardCriteria class has been removed completely. Instead the CardTypes value is passed to AcquireCard and RequestCard.
- Proprietary ACR-80 support has been removed completely. Instead it is recommended to use the standard P←
C/SC interface for it.
- The CardHandle methods ReadBinary and ReadRecord now handle the SW codes 0x61xx and 0x6Cxx automatically by issuing a GET RESPONSE or re-issuing the command with the correct length byte.
- All DLLs that make up the SmartCard API (Professional) have now been merged into a single SmartCard.dll. This eases deployment and development.
- Many small improvements that increase the robustness.

1.1.26 Version 5.0.21.117 (rev.17JAN2021)

For release 3.3 the SmartCard API was made security aware. This means that it is no longer required to always have a fully trusted caller in order to access the SmartCard API. To achive this goal the following individual changes have been made:

- replaces public release 3.2
- new: Added AllowPartiallyTrustedCallers attribute to all assemblies and also added security permission assertions for the following permissions whereever these permissions are needed: UnmanagedCode; Reflection; FileIO access to the card terminal configuration XML file. As a result it is now possible to use the SmartCard API from .NET applications that have been deployed through the Intranet or Internet, and even from applications that are hosted inside the Internet Explorer. To make this work all assemblies of the SmartCard API must be either added to the Global Assembly Cache, or must be granted full trust based on their string name using the Microsoft .NET Configuration tool.
- new: The card terminal registry file has been moved from the LocalApplicationData to the CommonApplication↩ Data folder. This means that all settings are now applicable to all users. This also means that a system administrator can now set up the card readers for his users.
- new: SmartCard API itself now only demands and asserts read access to the card terminal registry file. Only the CardTerminalConfigurator demands and asserts write access. This means that the configuration can now be protected by appropriate security policies.
- new: CardTerminalConfigurator logging is now directed to the CommonApplicationData folder instead of "C:\". Also, if the logging file yould not be created, then the CardTerminalConfigurator simply continues without logging.
- fix: MCardHandle.Connect method leaves MCardHandle in an unusable state when the connection failed.
- fix: CardTerminalRegistry.LoadRegistry does not work with empty files - XML errors in the registry file are ignored for more robust code
- fix: The SCardException now maps the Windows error code ERROR_UNRECOGNIZED_MEDIA to SCard↩ ResponseCode.UnresponsiveCard. The error code ERROR_UNRECOGNIZED_MEDIA is returned by some readers when a card is inserted the wrong way.
- new: CT-API card terminal number (CTN) can now be configured in the CardTerminalConfigurator.

1.1.27 Version 5.0.20.124 (rev.24JAN2020)

- replaces last public release 3.0.0.3
- Improved compatibility with .NET Framework 2.0.
- new: support for additional memory cards.
- Improved CardDataObject
- new: add CardDataObjectList for better and easier parsing of BER-TLV encoded card data.
- Improved MctTrivia sample application.

1.2 Card Reader Support

Basically all PC/SC Workgroup or CT-API compliant card readers should work. Card readers listed in this section have been tested successfully in our lab. Even if you don't find your reader on this list, it is very likely that it is supported. Contact us for if you need further integration of vendor specific features.

1.2.1 Cherry

- Cherry Smart Board G83-6744LUADE based on OMNIKEY CardMan 3x21; driver Cherry-SmartDeviceSetup_↩ 16Rev01_DE.exe; SPE PIN entry initiated via FEATURE_VERIFY_PIN_START
- Cherry ST-2000: based on SCM SPR3322K; driver CD 644-0365-V1.7; SPE PIN entry initiated via FEATURE↩ _VERIFY_PIN_START
- Cherry ST-1144: tested with Microsoft CCID class driver
- Cherry KC 1000 SC: tested with Microsoft CCID class driver

1.2.2 Gemalto

- Gemalto PC USB-SL: tested with driver v. 4.1.2.1

1.2.3 Kobil

- KOBIL KAAN Professional: driver KOBILTreiberSetup_V200s.exe; PIN entry initiated via FEATURE_MCT_RE↩ ADERDIRECT
- KOBIL KAAN Advanced: PIN entry initiated via FEATURE_VERIFY_PIN_START

1.2.4 HID OMNIKEY

SmartCardAPI has been thoroughly tested with OMNIKEY desktop readers for contact, contactless and PROX cards. All OMNIKEY readers are PC/SC compliant.

- OMNIKEY 3021: xchip version tested with driver v. 1.2.24.27; Aviator version tested with Microsoft CCID class driver
- OMNIKEY 3121: xchip version tested with driver v. 1.2.24.27; Aviator version tested with Microsoft CCID class driver
- OMNIKEY 3621: tested with driver v. 1.2.24.27; SPE PIN entry initiated via FEATURE_VERIFY_PIN_START
- OMNIKEY 3821: tested with driver v. 1.2.24.27; SPE PIN entry initiated via FEATURE_VERIFY_PIN_DIRECT
- OMNIKEY 4121: tested with driver v. 1.2.9.2
- OMNIKEY 4321: tested with driver v. 1.2.6.5
- OMNIKEY 5021: tested with driver v. 1.2.27.145
- OMNIKEY 5022 CL: tested with Microsoft CCID class driver
- OMNIKEY 5025: tested with Microsoft CCID class driver (default reader configuration)
  This reader can also be operated in legacy mode, which requires OMNIKEY 5x2x driver 1.2.27.145.
- OMNIKEY 5125 CL: tested with driver v. 1.2.27.145
- OMNIKEY 5325 CL: tested with driver v. 1.2.27.145
- OMNIKEY 5321: tested with driver v. 1.2.27.145
- OMNIKEY 5326 DFR: tested with driver v. 1.0.0.9

- OMNIKEY 5421 CK: tested with driver v. 1.2.26.145
- OMNIKEY 5127 CK: tested with Microsoft CCID class driver
- OMNIKEY 5127 CK Mini: tested with Microsoft CCID class driver
- OMNIKEY 5427 CK: tested with Microsoft CCID class driver

1.2.5 Identive

Formerly known as SCM. Most of these smart card readers work great with our API. Both contact and contactlsee readers are flashable. Therefore it is important to have firmware and matching driver.

- SCM SCR 532: a/k/a CHIPDRIVE pinpad 532: used SPR532_setup_V1.81.zip; SPE PIN entry initiated via FE↩ ATURE_VERIFY_PIN_START
- SCM SCR 531: keyboard reader: installed SCRx31_inst_German_V8.06.zip

1.2.6 Rainer SCT

- REINER SCT cyberJack pinpad; installed cjBase_V5.5.3.exe and cjPCSC_V2.1.1.exe; PIN entry initiated via SCardTransmit(PERFORM VERIFICATION)
- REINER SCT cyberJack e-com: installed cjBase_V5.5.3.exe, cjPCSC_V2.1.1.exe; PIN entry initiated via SCard↩ Transmit(PERFORM VERIFICATION)

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

Namespace Documentation

5.1 SmartCardAPI Namespace Reference

Namespaces

- namespace CardModule

5.2 SmartCardAPI.CardModule Namespace Reference

Namespaces

- namespace CAC
- namespace PIV

5.3 SmartCardAPI.CardModule.CAC Namespace Reference

Classes

- class **CAC_OBJECTID**

  File/Object IDs for CAC data.
- class **CACC**

  We expose global CAC constants through this class.
- class CacCardEdge

  Exposes high level PIV card edge in line with End-Point client application programming interface defined in SP 800-73-3. ref: `http://csrc.nist.gov/publications/nistpubs/800-73-3/sp800-73-3_PART1←_piv-card-applic-namespace-date-model-rep.pdf` *Method names are conform with 800-73-3 as much as possible to indicate the exact PIV API middleware function targeted with this implementation.CardTerminalException.*
- class CacCardException

  Whenever a SmartCardAPI.CardModule.CAC API error is returned an exception of this class is thrown.
- class CacCardPersonInstance

  This is a helper class to parse, analyze and display CAC Person Instance data. It processes data that typically resides in the Person Instance data containter. However, it does not require a physical card connection but rather is a method to process raw data read from the card in previous transaction.

- class CacCardPersonnelInstance

  This is a helper class to parse, analyze and display CAC Personnel Instance data. It processes data that typically resides in the Personnel Instance data containter. However, it does not require a physical card connection but rather is a method to process raw data read from the card in previous transaction.

Enumerations

- enum CAC_TAG : byte {
  CAC_TAG.PERSON_FIRST_NAME = 0x01, CAC_TAG.PERSON_MIDDLE_NAME = 0x02, CAC_TAG.PERS↩ON_LAST_NAME = 0x03, CAC_TAG.PERSON_CADENCY_NAME = 0x04,
  CAC_TAG.PERSON_IDENTIFIER = 0x05, CAC_TAG.DATE_OF_BIRTH = 0x06, CAC_TAG.SEX_CATEGOR↩Y_CODE = 0x07, CAC_TAG.PERSON_IDENTIFIER_TYPE_CODE = 0x08,
  CAC_TAG.BLOOD_TYPE_CODE = 0x11, CAC_TAG.DOD_EDI_PERSON_IDENTIFIER = 0x17, CAC_TAG.↩ORGAN_DONOR = 0x18, CAC_TAG.DOD_CONTRACTOR_FUNCTION_CODE = 0x19,
  CAC_TAG.US_GOVERNMENT_AGENCY_SUBAGENCY_CODE = 0x20, CAC_TAG.BRANCH_OF_SERVIC↩E_CODE = 0x24, CAC_TAG.PAY_GRADE_CODE = 0x25, CAC_TAG.RANK_CODE = 0x26,
  CAC_TAG.PERSONNEL_CATEGORY_CODE = 0x34, CAC_TAG.NON_US_GOVERNMENT_AGENCY_SU↩BAGENCY_CODE = 0x35, CAC_TAG.PAY_PLAN_CODE = 0x36, CAC_TAG.IDENTIFICATION_CARD_ISS↩UE_DATE = 0x62,
  CAC_TAG.IDENTIFICATION_CARD_EXPIRATION_DATE = 0x63, CAC_TAG.DATE_DEMOGRAPHIC_DAT↩A_WAS_LOADED_ON_CHIP = 0x65, CAC_TAG.DATE_DEMOGRAPHIC_DATA_ON_CHIP_EXPIRES = 0x66,
  CAC_TAG.CARD_INSTANCE_IDENTIFIER = 0x67,
  CAC_TAG.PERSONNEL_ENTITLEMENT_CONDITION_CODE = 0xD3 }

  CAC TAG constants used throughout NIST PIV card edge implementations.
- enum CacCardErrorCode {
  CacCardErrorCode.OK = 0, CacCardErrorCode.InternalError, CacCardErrorCode.UnknownCard, CacCard↩ErrorCode.UnknownError,
  CacCardErrorCode.NotImplemented }

  The error codes defined by the PIVCard API.

5.3.1 Enumeration Type Documentation

5.3.1.1 enum SmartCardAPI.CardModule.CAC.CAC_TAG : byte [strong]

CAC TAG constants used throughout NIST PIV card edge implementations.

Enumerator

PERSON_FIRST_NAME	card holder first name tag used in Person Instance File/Buffer
PERSON_MIDDLE_NAME	card holder middle name tag used in Person Instance File/Buffer
PERSON_LAST_NAME	card holder last name tag used in Person Instance File/Buffer
PERSON_CADENCY_NAME	card holder cadency name tag used in Person Instance File/Buffer
PERSON_IDENTIFIER	card holder identifiier tag used in Person Instance File/Buffer
DATE_OF_BIRTH	card holder date of birth tag used in Person Instance File/Buffer
SEX_CATEGORY_CODE	Person Instance File/Buffer Tag used for Sex category code
PERSON_IDENTIFIER_TYPE_CODE	Person Instance File/Buffer Tag used for person identifier type code
BLOOD_TYPE_CODE	Person Instance File/Buffer Tag used for blood type code
DOD_EDI_PERSON_IDENTIFIER	Person Instance File/Buffer Tag used for DoD EDI Person Identifier
ORGAN_DONOR	Person Instance File/Buffer Tag used for organ donor

DOD_CONTRACTOR_FUNCTION_CODE Personnel Instance File/Buffer Tag used for DoD Contractor Function Code

US_GOVERNMENT_AGENCY_SUBAGENCY_CODE Personnel Instance File/Buffer Tag used for US Government Agency/Subagency Code

BRANCH_OF_SERVICE_CODE Personnel Instance File/Buffer Tag used for Branch of Service Code

PAY_GRADE_CODE Personnel Instance File/Buffer Tag used for Pay Grade Code

RANK_CODE Personnel Instance File/Buffer Tag used for Rank Code

PERSONNEL_CATEGORY_CODE Personnel Instance File/Buffer Tag used for Personnel Category Code

NON_US_GOVERNMENT_AGENCY_SUBAGENCY_CODE Personnel Instance File/Buffer Tag used for Non-↩ US Government Agency/Subagency Code

PAY_PLAN_CODE Personnel Instance File/Buffer Tag used for Pay Plan Code

IDENTIFICATION_CARD_ISSUE_DATE Person Instance File/Buffer Tag used for ID card issue date

IDENTIFICATION_CARD_EXPIRATION_DATE Person Instance File/Buffer Tag used for ID card expiration date

DATE_DEMOGRAPHIC_DATA_WAS_LOADED_ON_CHIP Person Instance File/Buffer Tag used for date demographic data was loaded on chip

DATE_DEMOGRAPHIC_DATA_ON_CHIP_EXPIRES Person Instance File/Buffer Tag used for date demographic data on chip expires

CARD_INSTANCE_IDENTIFIER Person Instance File/Buffer Tag used for card issuance identifier

PERSONNEL_ENTITLEMENT_CONDITION_CODE Personnel Instance File/Buffer Tag used for Personnel Entitlement Condition Code

5.3.1.2 enum SmartCardAPI.CardModule.CAC.CacCardErrorCode `[strong]`

The error codes defined by the PIVCard API.

Enumerator

OK PIV OK

InternalError An internal error has occurred.

UnknownCard Unknown card. This is not a PIV card.

UnknownError An unknown error has occurred.

NotImplemented Not implemented.

5.4 SmartCardAPI.CardModule.PIV Namespace Reference

Classes

- class **PIV_DATA_MODEL**

  Data models according to PIV II SP 800-73 v1 (plus CAC_V1)
- class **PIV_OBJECTID**

  Object IDs for PIV card data
- class PivAddonAuthenticator

  A piv card crypto pkcs #1.
- class **PIVC**

  We expose global PIV constants through this class.

- class PivCardAddonPkcs1

  A piv card crypto pkcs #1.
- class PivCardCapabilites

  Helper class to parse, analyze and display BER encoded PIV card capability data. This class allows easy access to any element within the PIV card capability container. Note that this class does not provide card access methods. Data must be read external to this class and then passed during instantiation. Ref SP 800-73-3 Part 1, page 20, , SP800-73-3.
- class PivCardCertificate

  Helper class to parse X509 data coming from a card data container.
- class PivCardChuid

  This is a helper class to parse, analyze and display PIV CHUID data. It processes data that typically resides in the CHUID data containter. However, it does not require a physical card connection.
- class **PivCardCrypto**

  Helper class for cryptographic operations with PIV card.
- class PivCardCryptoAes

  Easy-to-use entry point for AES crypto processed by cryptographic service provider.
- class PivCardCryptoHash

  Helper class for cryptographic operations requiring hash
- class PivCardCryptoRsa

  Easy-to-use entry point for RSA crypto processed by cryptographic service provider.
- class PivCardEdge

  Exposes high level PIV card edge in line with End-Point client application programming interface defined in SP 800-73-4. ref: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-73-4.↩ pdf` *Method names are conform with 800-73-4 as much as possible to indicate the exact PIV API middleware function targeted with this implementation.*
- class PivCardException

  Whenever a SmartCardAPI.CardModule.PIV API error is returned an exception of this class is thrown.
- class PivCardGzip

  This class exposes method required to compress/decompress PIV card data.
- class PivCardPrintedInformation

  This is a helper class to parse, analyze and display PIV PrintedInformation data. It processes data that typically resides in the Printed Information data containter. This class allows easy access to any element within the PIV Printed Information container. Ref SP 800-73- 3 However, it does not require a physical card connection.
- class PivCardSecurityObject

  Security Object ASN.1 OID=2.16.840.1.101.3.7.2.144.0 This is a helper class to parse, analyze and display PIV security object data. This method does not require a physical card connection. The Security Object is in accordance with Appendix C of PKI for Machine Readable Travel Documents Offering ICC Read-Only Access Version 1.1. [5] Tag “0xBA” is used to map the ContainerIDs in the PIV data model to the 16 Data Groups specified in the Machine Readable Travel Document (MRTD). The mapping enables the Security Object to be fully compliant for future activities with identity documents.

Enumerations

- enum PIV_TAG : byte {
  PIV_TAG.NAME = 0x01, PIV_TAG.EMPLOYEE_AFFILIATION = 0x02, PIV_TAG.EMPLOYEE_AFFILIATION↩ _LINE2 = 0x03, PIV_TAG.EXPIRATION_DATE = 0x04,
  PIV_TAG.AGENCY_CARD_SERIAL_NUMBER = 0x05, PIV_TAG.ISSUER_IDENTIFICATION = 0x06, PIV_T↩ AG.ORGANIZATION_AFFILIATIOIN = 0x07, PIV_TAG.ORGANIZATION_AFFILIATIOIN_LINE2 = 0x08,
  PIV_TAG.CONTAINER_ID = 0x5C, PIV_TAG.FASCN = 0x30, PIV_TAG.ORG_ID = 0x32, PIV_TAG.DUNS = 0x33,
  PIV_TAG.GUID = 0x34, PIV_TAG.EXP_DATE = 0x35, PIV_TAG.UUID = 0x36, PIV_TAG.AUTH_KEY_MAP =

0x3D,
PIV_TAG.ISSUER_ASYM_SIG = 0x3E, PIV_TAG.PIV_AID = 0x4F, PIV_TAG.CONTAINER_DATA = 0x53, PI↩
V_TAG.CERTIFICATE = 0x70,
PIV_TAG.CERTIFICATE_INFO = 0x71, PIV_TAG.MSCUID = 0x72, PIV_TAG.AUTHENTICATION_TEMPLATE = 0x7C, PIV_TAG.SECURITY_OBJECT_BUFFER = 0xB4,
PIV_TAG.MAPPING_DG2CONTAINERID = 0xBA, PIV_TAG.SECURITY_OBJECT = 0xBB, PIV_TAG.BIOME↩
TRIC_DATA = 0xBC, PIV_TAG.FINGERPRINT = BIOMETRIC_DATA,
PIV_TAG.IMAGE_VISUAL = BIOMETRIC_DATA, PIV_TAG.IMAGE_IRIS = BIOMETRIC_DATA, PIV_TAG.KE↩
Y_WITH_ONCARD_CERT = 0xC1, PIV_TAG.KEY_WITH_OFFCARD_CERT = 0xC2,
PIV_TAG.BUFFER_LENGTH = 0xEE, PIV_TAG.CARD_IDENTIFIER = 0xF0, PIV_TAG.CAPABILITY_CONT↩
AINER_VERSION_NUMBER = 0xF1, PIV_TAG.CAPABILITY_GRAMMAR_VERSION_NUMBER = 0xF2,
PIV_TAG.URL = 0xF3, PIV_TAG.APPLICATIONS_CARD_URL = URL, PIV_TAG.OFFCARD_CERT_URL = U↩
RL, PIV_TAG.PKCS_15 = 0xF4,
PIV_TAG.REGISTERED_DATA_MODEL_NUMBER = 0xF5, PIV_TAG.ACCESS_CONTROL_RULE_TABLE = 0xF6, PIV_TAG.CARD_APDUS = 0xF7, PIV_TAG.REDIRECTION_TAG = 0xFA,
PIV_TAG.CAPABILITY_TUPLES = 0xFB, PIV_TAG.STATUS_TUPLES = 0xFC, PIV_TAG.NEXT_CCC = 0xFD,
PIV_TAG.EXTENDED_APPLICATION_CARD_URL = 0xE3,
PIV_TAG.ERROR_DETECTION_CODE = 0xFE }

PIV TAG constants used throughout NIST PIV card edge implementations.

- enum PivCardErrorCode {
PivCardErrorCode.OK = 0, PivCardErrorCode.InternalError, PivCardErrorCode.UnknownCard, PivCardError↩
Code.UnknownError,
PivCardErrorCode.NotImplemented }

  The error codes defined by the PIVCard API.

5.4.1 Enumeration Type Documentation

5.4.1.1 enum SmartCardAPI.CardModule.PIV.PIV_TAG : byte [strong]

PIV TAG constants used throughout NIST PIV card edge implementations.

Enumerator

NAME card holder name tag used in PRINTED information

EMPLOYEE_AFFILIATION employee affiliation tag used in PRINTED information

EMPLOYEE_AFFILIATION_LINE2 employee affiliation line 2 tag used in PRINTED information This is a depre-ceated item.

EXPIRATION_DATE expiration date tag used in PRINTED information This is a depreceated item.

AGENCY_CARD_SERIAL_NUMBER agency card serial number tag used in PRINTED information

ISSUER_IDENTIFICATION issuer identification tag used in PRINTED information

ORGANIZATION_AFFILIATIOIN organization affiliation (line 1) tag used in PRINTED information

ORGANIZATION_AFFILIATIOIN_LINE2 organization affiliation (line 2) tag used in PRINTED information

CONTAINER_ID BER-TLV container ID

FASCN FASCN - part of CHUID data

ORG_ID Organization Identifier

DUNS DUNS

GUID GUID a/k/a card UUID - part of CHUID data

EXP_DATE CHUID expiration date - part of CHUID data

UUID Cardholder UUID - part of CHUID data mandatory since SP 800-73-4, 2015

AUTH_KEY_MAP CHUID authentication key map; optional i.e length can be zero. Part of CHUID data

ISSUER_ASYM_SIG CHUID issuer's asymetric signature. Part of CHUID data.

PIV_AID PIV application ID. Part of discovery object.

CONTAINER_DATA Container data such as CHUID, X509 certificate. Note that when writing container data in a chain, it is mandatory to make both, CONTAINER_DATA tag and CONTAINER_ID tag with their respective length fields as part of the first CLA=0x10. This way you ensure that the card allocates the space correctly and knows when the chain should end i.e how much data has to follow in subsequent data fields.

CERTIFICATE Certificate data. Note that the TAG 0x70 on-card data is encrypted. Therefore data can only be used for X509 related tasks after decryption.

CERTIFICATE_INFO TAG 0x70 reflecting certificate info. 1 byte fixed length

MSCUID Optional tag for PIV on-card certificates. Variable length up to 38 bytes.

AUTHENTICATION_TEMPLATE Tag for dynamic authentication template; Used in General Authenticate

SECURITY_OBJECT_BUFFER Part of card capability data (48 byte, optional).

MAPPING_DG2CONTAINERID Mapping from DG to container ID. Found in Security object container

SECURITY_OBJECT Security object storedin mandatory Security Object Container

BIOMETRIC_DATA Indicates biometric data (Fingerprint, iris, mugshot)

FINGERPRINT Fingerprint 1 & 2 (recommended length 4000 byte).

IMAGE_VISUAL Image for visual verification (mugshot) (recommended length 12704 byte).

IMAGE_IRIS Images for Iris (recommended length 7100 byte).

KEY_WITH_ONCARD_CERT Keys with on-card certsificates. Part of Key history object.

KEY_WITH_OFFCARD_CERT Keys with off-card certificates. Part of Key history object.

BUFFER_LENGTH Buffer length

CARD_IDENTIFIER Card identifier Part of card capability data (21 bytes fixed).

CAPABILITY_CONTAINER_VERSION_NUMBER Part of card capability data (1 byte fixed).

CAPABILITY_GRAMMAR_VERSION_NUMBER Part of card capability data (1 byte fixed).

URL URL tag

APPLICATIONS_CARD_URL Part of card capability data (128 byte variable).

OFFCARD_CERT_URL Part of key history object data (118 byte variable).

PKCS_15 Part of card capability data (1 byte fixed).

REGISTERED_DATA_MODEL_NUMBER Part of card capability data (1 byte fixed).

ACCESS_CONTROL_RULE_TABLE Part of card capability data (17 byte fixed).

CARD_APDUS Part of card capability data (0 byte).

REDIRECTION_TAG Part of card capability data (0 byte).

CAPABILITY_TUPLES Part of card capability data (0 byte).

STATUS_TUPLES Part of card capability data (0 byte).

NEXT_CCC Part of card capability data (0 byte).

EXTENDED_APPLICATION_CARD_URL Part of card capability data (48 byte fixed, optional).

ERROR_DETECTION_CODE CHUID error detection; optional i.e. tag||zero might not be present. Part of CHUID data, capability data. (0 byte)

5.4.1.2 enum SmartCardAPI.CardModule.PIV.PivCardErrorCode `[strong]`

The error codes defined by the PIVCard API.

Enumerator

OK PIV OK

InternalError An internal error has occurred.

UnknownCard Unknown card. This is not a PIV card.

UnknownError An unknown error has occurred.

NotImplemented Not implemented.

5.5 Subsembly Namespace Reference

Namespaces

- namespace SmartCard

5.6 Subsembly.SmartCard Namespace Reference

Namespaces

- namespace PcSc

Classes

- class CardCommandAPDU

  Generic utility class for building and handling of smart card APDUs according to ISO 7816-4. Command APDUs and response APDUs are implemented.
- class CardCrypto

  A card crypto.
- class CardCtApiConfig

  Collects information about a CT-API card terminal configuration. This information usually is stored in the CardTerminal←↪ Registry.
- class CardDataObject

  Generic utility class for evaluating and generating Basic-Encoding-Rules (BER) encoded Tag- Length-Value (TLV) data objects.
- class CardDataObjectList

  Generic utility class for evaluating and generating a sequence of Tag-Length-Value (TLV) data objects which are stored as CardDataObject instances.
- class CardHandle

  Exclusive handle to a particular card that was introduced to the system by insertion into a card terminal slot. An application shall access smart cards only through instances of this or a derived class.
- class CardHex

  Helper class to make life with card hex data easier.
- class **CardISO7816**

  This class contains constants valuable for ISO7816 compliant card edge implementations.

- class CardPcScConfig

  Collects information about a PC/SC card terminal configuration. This information usually is stored in the CardTerminal←Registry.

- class CardPcscPart3

  Provides ATR parsing capabilities for PCSC 2.01 Part 3 compliant card systems.

- class CardPcScTerminal

  Implementation of the high-level ICardTerminal abstraction based on the PC/SC Workgroup API.;

- class CardPcScTerminalBase

  Higher level smart card access class based on the PC/SC access class SCardComm.

- class CardPinControl

  Collection of attributes and methods that control the PIN handling and its transformation into a command APDU sent to the smart card.

- class CardQuickXs

  Higher level smart card access class based on the PC/SC access classSCardComm. It is meant for quick access to smart card system in a way that is still relatively close to the native API.

- class CardResponseAPDU

  Generic utility class for evaluating and processing of smart card response APDUs according to ISO 7816-4.

- class CardTerminalBase

  Common base class to simplify the implementation of the ICardTerminal interface. This class is intended for card terminal manufacturers and not for application programmers.

- class CardTerminalConfig

  Collects information about a card terminal configuration that is stored in the CardTerminalRegistry

- class CardTerminalEventArgs

  This class provides the event information for the card terminal events that are raised by the CardTerminalManager. Events can be triggered by card insertion, removal as well as a reader's request for card removal (from a the lip of a motorized reader or from an RF field).

- class CardTerminalException

  Whenever an error occurs an exception of this class is thrown.

- class CardTerminalManager

  CardTerminalManager class is a singleton that is the one and only instance that manages card terminals and tracks card insertions and removals.

- class CardTerminalRegistry

  The CardTerminalRegistry class is a singleton that is the one and only instance that manages the persistent storage of card terminal configuration values of the SmartCard Subsembly in a registration file.

- class CardTerminalSlot

  Internal class that is used by the CardTerminalManager to store the context for tracking the state of a card terminal slot.

- class **CardTexts**

  A card texts. This class cannot be inherited.

- interface ICardTerminal

  This interface supports abstract access to any smart card terminal, regardless of the actual driver technology. An instance of this interface is always bound to a particular card terminal unit installed in the system.

- interface ICardTerminalClass2

  This interface provides methods that are only applicable to card terminals that comply at least with ZKA class 2, i.e. that have a PIN pad and support for secure PIN entry. Card terminals supporting class 2 functionality implement this interface alongside the ICardTerminal interface.

- class LogHelper

  Easy-to-use helper class to log (debug or trace) info

- class **PCSC_PART3**

  Constants related to PC/SC 2.x This class might eventually be moved to a shared module as it can be used for any contactless card system that is PC/SC 2.0x compliant.

- class **PCSC_PART3_CARD_IDENTIFIER**

    Card Identifier as defined in PC/SC Part 3 rev.2.01.08 (June 2011), p2-4.
- class **PCSC_PART3_KEY_STRUCTURE**

    Key structure for load key command as defined in PC/SC 2.x Part 3, p28 b7..b0 reflect bits in MSB-LSB order. The LSB nibble of the key structure byte contains the reader key number sor secured transmission between host application and reader.
- class **PCSC_PART3_STANDARD**

    Constants reflecting PC/SC part3 compliant card standard byte 'SS'. ref: PC/SC Part 3 rev.2.01.09 (June 2013)
- class **SMARTCARDAPI**

    We expose global constants through this class.

Enumerations

- enum CardDataObjectEncoding { CardDataObjectEncoding.Undefined, CardDataObjectEncoding.BER = 1, CardDataObjectEncoding.Simple = 2, CardDataObjectEncoding.ZkaNotepadF0 = 0x10000 }

    Used to select a particular encoding for Tag Length Value (TLV) encoded data objects.
- enum CardPinEncoding {
  CardPinEncoding.BCD = 0, CardPinEncoding.Ascii = 1, CardPinEncoding.Format2PinBlock = 2, CardPin←
  Encoding.EMVCo = 3,
  CardPinEncoding.Binary = 4, CardPinEncoding.Unicode = 5 }

    Specifies how a smart card PIN shall be encoded when transmitted inside the command APDU.
- enum CardTerminalType { CardTerminalType.UNKNOWN = 0, CardTerminalType.PCSC = 1, CardTerminal←
  Type.CTAPI = 2, CardTerminalType.MCARD = 4 }

    Enumerates the standard card terminal types.
- enum CardTerminalExceptionCode {
  CardTerminalExceptionCode.InvalidAPDU, CardTerminalExceptionCode.CardWithdrawn, CardTerminal←
  ExceptionCode.CardCommunicationError, CardTerminalExceptionCode.DriverError,
  CardTerminalExceptionCode.DriverException, CardTerminalExceptionCode.DriverIncompatible, CardTerminal←
  ExceptionCode.TerminalUnavailable }

    These are the possible high level exception codes that can be thrown in a CardTerminalException
- enum CardTerminalSlotState {
  CardTerminalSlotState.Empty, CardTerminalSlotState.RequiresRemoval, CardTerminalSlotState.Inserted,
  CardTerminalSlotState.Accepted,
  CardTerminalSlotState.NoTerminal }

    The possible states of a card terminal slot. The state indicates whether a card is present and whether a present card is currently owned by a CardHandle instance. For motorized readers or readers with card position sensors in addition to the card switch also support a removable state to indicate that user interaction is required.
- enum CardTypes {
  CardTypes.Invalid = 0, CardTypes.T0 = 0x00000001, CardTypes.T1 = 0x00000002, CardTypes.ProcessorCards = (T0 | T1),
  CardTypes.SLE4406 = 0x010000, CardTypes.SLE4418 = 0x020000, CardTypes.SLE4428 = 0x030000, Card←
  Types.SLE4432 = 0x040000,
  CardTypes.SLE4436 = 0x050000, CardTypes.SLE4442 = 0x060000, CardTypes.SLE5536 = 0x070000, Card←
  Types.AT24C01ASC = 0x080000,
  CardTypes.AT24C02SC = 0x090000, CardTypes.AT24C04SC = 0x0A0000, CardTypes.AT24C08SC = 0x0←
  B0000, CardTypes.AT24C16SC = 0x0C0000,
  CardTypes.AT24C32SC = 0x0D0000, CardTypes.AT24C64SC = 0x0E0000, CardTypes.AT24C128SC = 0x0←
  F0000, CardTypes.AT24C256SC = 0x100000,
  CardTypes.AT24C512SC = 0x110000, CardTypes.AT88SC153 = 0x120000, CardTypes.AT88SC1608 = 0x130000, CardTypes.SLE4404 = 0x140000,
  CardTypes.MemoryCards = 0x7FFF0000 }

Used to select a card type to connect (ICardTerminal.ConnectCard or reconnect (ICardTerminal.ReconnectCard) to.

- enum CardTransactionResult { CardTransactionResult.Success = 0, CardTransactionResult.NotImplemented = -1, CardTransactionResult.ResetCard = -2 }

  constants to represent card transaction results.

- enum CardActivationResult {
  CardActivationResult.Success = 0, CardActivationResult.NoCard = -1, CardActivationResult.UnresponsiveCard = -2, CardActivationResult.UnsupportedCard = -3,
  CardActivationResult.InUse = -4, CardActivationResult.Cancelled = -7, CardActivationResult.NotImplemented = -8, CardActivationResult.RemovableCard = -9,
  CardActivationResult.JammedCard = -10, CardActivationResult.Unknown = -11 }

  Values that represent card activation results.

- enum CardPosition {
  CardPosition.Unknown = 0x00000000, CardPosition.NoCard = 0x00000080, CardPosition.Front = 0x000000010, CardPosition.Middle = 0x00000020,
  CardPosition.Rear = 0x00000040 }

  Flags to indicate the card position. The flags are mostly used in motorized card readers. In motorized readers that fire on both, the middle and rear sensors to indicate that the card is in the correct position to power a card, the CardInserted shall be used to indicate card insertion.

- enum CardTerminalFunctionalUnits {
  CardTerminalFunctionalUnits.None = 0x00000000, CardTerminalFunctionalUnits.PinPad = 0x00000001, CardTerminalFunctionalUnits.Display = 0x00000002, CardTerminalFunctionalUnits.TransportMechanism = 0x00000010,
  CardTerminalFunctionalUnits.CardSensorFront = 0x00000020, CardTerminalFunctionalUnits.CardSensorMiddle = 0x00000040, CardTerminalFunctionalUnits.CardSensorRear = 0x00000080 }

  Indicates which special functional devices are available by a card terminal. By these indications it is possible to determine the ZKA class that a card terminal complies with. It also provides an entry point for features that are specific to motorized readers with internal card transport mechanism.

Functions

- delegate void CardTerminalEventHandler (object aSender, CardTerminalEventArgs aEventArgs)

  Delegate for sinking card terminal events raised by the CardTerminalManager.

5.6.1 Enumeration Type Documentation

5.6.1.1 enum Subsembly.SmartCard.CardActivationResult [strong]

Values that represent card activation results.

Enumerator

Success An asynchronous card was successfully reset and activated. Its ATR is returned.

NoCard No card is inserted in the requested card slot, hence no card activation was attempted.

UnresponsiveCard The inserted card is mute. It is either inserted the wrong way, completely broken, or not a smart card at all (e.g. the user inserted a magnetic stripe card that does not contain a chip.

UnsupportedCard The inserted card is a smart card, but the card terminal cannot handle it. Either the card communication protocol is not understood or some other requirement of the card cannot be satisfied by the card terminal.

InUse The card slot is already in exclusive use by another application. Therefore a card activation was not attempted.

Cancelled The card activation was manually cancelled by the user or timed out. This implies that no card was inserted. This can only be returned from activation methods that have a timeout or a GUI that can be cancelled.

NotImplemented The requested CardTypes is not supported by the actual ICardTerminal implementation.

RemovableCard Card detected that can be removed from the reader. This activation result typically requires a motorized reader with card sensor at its insertion slot.

JammedCard Card detected that does not move inspite multiple attempts to move it to the reader exit. This activation result typically requires a motorized reader with card sensors at its insertion slot, middle and rear.

Unknown We don't know the activation result. Most likely because there hasn'ty been one within the current reader session.

5.6.1.2 enum Subsembly.SmartCard.CardDataObjectEncoding [strong]

Used to select a particular encoding for Tag Length Value (TLV) encoded data objects.

Enumerator

Undefined Undefined value.

BER Basic Encoding Rules (BER-TLV) encoding.

Simple SIMPLE-TLV encoding.

ZkaNotepadF0 Special BER like encoding used by the EF_NOTEPAD file on German ZKA smart cards. This special value is required as the ZKA smart cards use an errorneous BER encoding.

5.6.1.3 enum Subsembly.SmartCard.CardPinEncoding [strong]

Specifies how a smart card PIN shall be encoded when transmitted inside the command APDU.

Note that the smart card usually defines the PIN codings that it accepts and the application must follow.

Enumerator

BCD The PIN is BCD encoded, i.e. each digit occupies four bits. This encoding only supports numeric PINs.

Ascii The PIN is ASCII encoded, e.g. '31' for the digit "1". The most significant bit of each byte is zero. This encoding supports alphanumeric PINs.

Format2PinBlock The PIN is encoded in a special "Format 2 PIN Block". This encoding only supports numeric PINs.

EMVCo The PIN is encrypted according to the EMVCo specification. This encoding is currently not supported.

Binary New in version 4.0: The PIN is encoded as bytes, e.g. '01' for the digit "1". This encoding is NOT supported by CardPinControl.BuildTerminalCommandAPDU and CardPinControl.BuildTerminalModifyCommandAPDU.

Unicode New in version 4.0: The PIN is encoded as 2 byte unicode characters. This encoding is NOT supported by CardPinControl.BuildTerminalCommandAPDU and CardPinControl.BuildTerminalModifyCommandAPDU.

5.6.1.4 enum Subsembly.SmartCard.CardPosition [strong]

Flags to indicate the card position. The flags are mostly used in motorized card readers. In motorized readers that fire on both, the middle and rear sensors to indicate that the card is in the correct position to power a card, the CardInserted shall be used to indicate card insertion.

Enumerator

- ***Unknown*** The card position is not known. Use this state as default and fall-back position when instantiating a card terminal object.
- ***NoCard*** The card is not anywhere in the reader. This flag should overwrite i.e. set to zero all lower flags.
- ***Front*** The card is in the reader front
- ***Middle*** The card is in the reader middle
- ***Rear*** The card is in the reader rear.

5.6.1.5 enum Subsembly.SmartCard.CardTerminalExceptionCode [strong]

These are the possible high level exception codes that can be thrown in a CardTerminalException

Enumerator

- ***InvalidAPDU*** A command APDU or response APDU was invalid according to the requirements of the method that did raise this exception.
- ***CardWithdrawn*** The card was unexpectedly removed from the card terminal. This exception only happens if there already was a connection to the card. It is not thrown when an attempt is made to power up a card for the very first time.
- ***CardCommunicationError*** An error occured while communicating with the card. The card must be reset.
- ***DriverError*** The lower layer driver returned an error code. This must only be used, if the drivers error code could not be mapped to some other, more meaningful CardTerminalExceptionCode.
- ***DriverException*** An exception from a lower layer driver was caught and is propagated as the inner exception of this exception. This must only be used, if the caught exception could not be mapped to some other, more meaningful CardTerminalExceptionCode.
- ***DriverIncompatible*** The lower layer driver returned some unexpected or unknown result, or some other unexpected driver behaviour occured.
- ***TerminalUnavailable*** This is to indicate that a reader got lost; most likely disconnected after it was already successfully added to the terminal slots that are tracked.

5.6.1.6 enum Subsembly.SmartCard.CardTerminalFunctionalUnits [strong]

Indicates which special functional devices are available by a card terminal. By these indications it is possible to determine the ZKA class that a card terminal complies with. It also provides an entry point for features that are specific to motorized readers with internal card transport mechanism.

Enumerator

- ***None*** The card terminal consists of nothing but a single primary card slot. A card terminal that includes nothing but the card slot complies with ZKA class 1.
- ***PinPad*** The card terminal includes a numeric PIN pad which can be used for secure PIN entry. The PIN pad layout is assumed to be accoring to EMVCo. A card terminal that includes a PIN pad for secure PIN entry complies with ZKA class 2.
- ***Display*** The card terminal includes a display for prompting the user and for feedback. In accordance with CT-API and EMVCo, the display must be acpable of displaying at least two lines of text with 16 characters each. A card terminal that includes a PIN pad and a display complies with ZKA class 3.
- ***TransportMechanism*** The card terminal includes a transport mechanism to pull/eject cards. This reader feature is typically combined with front/mid/rear card sensors.

CardSensorFront The card terminal includes a front sensor to detect a card at the gate (front entry slot) of the reader. This sensor does not fire when the card is in position to be powered up.

CardSensorMiddle The card terminal includes a mid sensor to detect a card in the middle section of the reader's transport mechanism. This sensor does not fire when the card is in position to be powered up.

CardSensorRear The card terminal includes a rear sensor to detect a card in power-up position.

5.6.1.7 enum Subsembly.SmartCard.CardTerminalSlotState [strong]

The possible states of a card terminal slot. The state indicates whether a card is present and whether a present card is currently owned by a CardHandle instance. For motorized readers or readers with card position sensors in addition to the card switch also support a removable state to indicate that user interaction is required.

Enumerator

Empty The card terminal slot is empty and there are no outstanding CardHandle instances connected to this card terminal slot.

RequiresRemoval A card is physically present and not owned by a CardHandle. This slot state is typically used by motorized readers to indicate that there is a card in the reader fron that requires to be removed. Such readers have multiple card position sensors to detect the card position within the transport mechanism.
This slot state can also be helpful to ensure that a contactless card can be powered down by removing it from the RF filed which is necessary if the reader does not provide a way to switch on/off the field.

Inserted A card is physically inserted but is not owned by a CardHandle. The card is properly positioned for operation. The card may be acquired through CardTerminalSlot.AcquireCard. However, it is not powered up in this state. This is the only state that indicates that the card is positioned correctly. Note that readers with multiple card position sensors can detect card presence before and after the card slot is in the Inserted state. The Inserted state is a/k/a/ SWALLOWED.

Accepted An inserted card was successfully acquired by CardTerminalSlot.AcquireCard. It is owned by the referenced CardHandle instance. The card is powered up in this state (a/k/a POWERED)

NoTerminal A previously registered slot became unavailable. Maybe reader was removed?

5.6.1.8 enum Subsembly.SmartCard.CardTerminalType [strong]

Enumerates the standard card terminal types.

The SmartCard Subsembly includes two standard ICardTerminal implementations. One that is based on the CT-API and another based on PC/SC. This enumeration identifies those standard implementations.

Enumerator

UNKNOWN This identifies a non-standard ICardTerminal implementation.

PCSC This identifies the PC/SC Workgroup API based standard ICardTerminal implementation. The classname of this implementation is CardTerminalConfig.PCSCCARDTERMINALCLASSNAME.

CTAPI This identifies the CT-API based standard ICardTerminal implementation. The classname of this implementation is CardTerminalConfig.CTAPICARDTERMINALCLASSNAME.

MCARD This identifies the PC/SC and MCard API based standard ICardTerminal implementation. The classname of this implementation is CardTerminalConfig.MCARDCARDTERMINALCLASSNAME.

5.6.1.9 enum Subsembly.SmartCard.CardTransactionResult `[strong]`

constants to represent card transaction results.

Enumerator

Success The transaction request was processed without any errors.

NotImplemented Not Implemented

ResetCard Another process or thread has reset the card. attempted.

5.6.1.10 enum Subsembly.SmartCard.CardTypes `[strong]`

Used to select a card type to connect (ICardTerminal.ConnectCard or reconnect (ICardTerminal.ReconnectCard) to.

The processor card enumeration values T0 and T1 are equal to the PC/SC standard SCARD_PROTOCOL_T0 and SCARD_PROTOCOL_T1 values and can also be used in combination in order to indicate either of both protocols.

All memory card enumerations values are equal to the corresponding MCARD API definitions, shifted to the left by 16 bits.

Enumerator

Invalid Invalid value.

T0 Connect to asynchronous card, preferring the T=0 protocol. If the presented card is an asynchronous card that only supports the T=1 protocol, then the connection is made using the T=1 protocol.

T1 Connect to asynchronous card, preferring the T=1 protocol. If the presented card is an asynchronous card that only supports the T=0 protocol, then the connection is made using the T=0 protocol.

ProcessorCards Bitwise or of all processor card types.

SLE4406

SLE4418 Connect to SLE 4418 (or compatible) memory card.
SLE 4418 is 3-Wire-Bus with 1024 Bytes.

SLE4428 Connect to SLE 4428 (or compatible) memory card.
SLE 4428 is 3-Wire-Bus with 1024 Bytes and PIN write-protection.

SLE4432 Connect to SLE 4432 (or compatible) memory card.
SLE 4432 is 2-Wire-Bus with 256 Bytes.

SLE4436

SLE4442 Connect to SLE 4442 (or compatible) memory card.
SLE 4442 is 2-Wire-Bus with 256 Bytes and PIN write-protection.

SLE5536

AT24C01ASC Connect to Atmel AT24C01ASC memory card.
The AT24C01ASC chip is a 1K (128 x 8), two-wire serial EEPROM.

AT24C02SC Connect to Atmel AT24C02SC memory card.
The AT24C02SC chip is a 2K (256 x 8), two-wire serial EEPROM.

AT24C04SC Connect to Atmel AT24C04SC memory card.
The AT24C04SC chip is a 4K (512 x 8), two-wire serial EEPROM.

AT24C08SC Connect to Atmel AT24C08SC memory card.
The AT24C08SC chip is a 8K (1024 x 8), two-wire serial EEPROM.

AT24C16SC Connect to Atmel AT24C16SC memory card.
The AT24C16SC chip is a 16K (2048 x 8), two-wire serial EEPROM.

AT24C32SC Connect to Atmel AT24C32SC memory card.
The AT24C32SC chip is a 32K (4096 x 8), two-wire serial EEPROM.

AT24C64SC Connect to Atmel AT24C64SC memory card.
The AT24C64SC chip is a 64K (8192 x 8), two-wire serial EEPROM.

AT24C128SC Connect to Atmel AT24C128SC memory card.
The AT24C128SC chip is a 128K (16384 x 8), two-wire serial EEPROM.

AT24C256SC Connect to Atmel AT24C256SC memory card.
The AT24C256SC chip is a 256K (32768 x 8), two-wire serial EEPROM.

AT24C512SC Connect to Atmel AT24C512SC memory card.
The AT24C512SC chip is a 512K (65536 x 8), two-wire serial EEPROM.

AT88SC153

AT88SC1608

SLE4404

MemoryCards Bitwise or of all memory card types. This can be used to quickly check whether a particular value refers to a memory card type.

5.6.2 Function Documentation

5.6.2.1 delegate void Subsembly.SmartCard.CardTerminalEventHandler (object *aSender*, CardTerminalEventArgs *aEventArgs*)

Delegate for sinking card terminal events raised by the CardTerminalManager.

When this delegate is invoked, the event sender object is always the CardTerminalManager singleton.

All card terminal events are fired from a non UI background worker thread. Therefore it is not possible to call any UI methods from the System.Windows.Forms namespace from within an event handler without using Control.BeginInvoke!

Parameters

aSender	The sender.
aEventArgs	Card terminal event information.

5.7 Subsembly.SmartCard.PcSc Namespace Reference

Classes

- class SCardComm

  This object encapsulates a communication interface to a specific card or reader. It provides methods for managing the connections, controlling transactions, sending and receiving commands, and determining card state information.
- class SCardException

  Exception for signalling smart card API errors.
- class SCardPinBlock

  Representation of the PC/SC PINBLOCK structure as defined in PC/SC 2.0 part 9 "IFDs with Extended Capabilities".
- class SCardResourceManager

  The Resource Manager is a key component of the PC/SC Workgroups architecture. It is responsible for managing all PC/SC smart card resources.
- struct SCardTrackStateInfo

  Structure that holds information for the purpose of tracking the card reader state. This kind of structure is used by the SCardResourceManager.GetStatusChange method.

Enumerations

- enum SCardAccessMode { SCardAccessMode.Null = 0, SCardAccessMode.Shared = (int)WinSCard.SCA↵RD_SHARE_SHARED, SCardAccessMode.Exclusive = (int)WinSCard.SCARD_SHARE_EXCLUSIVE, SCard↵AccessMode.Direct = (int)WinSCard.SCARD_SHARE_DIRECT }

  Values that represent card access modes.
- enum SCardProtocolIdentifiers {
  SCardProtocolIdentifiers.Undefined = (int)WinSCard.SCARD_PROTOCOL_UNDEFINED, SCardProtocol↵Identifiers.Any = (int)(WinSCard.SCARD_PROTOCOL_T0 | WinSCard.SCARD_PROTOCOL_T1), SCard↵ProtocolIdentifiers.Default = unchecked((int)WinSCard.SCARD_PROTOCOL_DEFAULT), SCardProtocol↵Identifiers.Optimal = (int)WinSCard.SCARD_PROTOCOL_OPTIMAL,
  SCardProtocolIdentifiers.T0 = (int)WinSCard.SCARD_PROTOCOL_T0, SCardProtocolIdentifiers.T1 = (int)Win↵SCard.SCARD_PROTOCOL_T1, SCardProtocolIdentifiers.Raw = (int)WinSCard.SCARD_PROTOCOL_RAW
  }

  This enumeration defines the protocols used in communication with the card.
- enum SCardDisposition {
  SCardDisposition.LeaveCard = (int)WinSCard.SCARD_LEAVE_CARD, SCardDisposition.ResetCard = (int)WinSCard.SCARD_RESET_CARD, SCardDisposition.UnpowerCard = (int)WinSCard.SCARD_UNPOW↵ER_CARD, SCardDisposition.EjectCard = (int)WinSCard.SCARD_EJECT_CARD,
  SCardDisposition.Confiscate = (int)WinSCard.SCARD_CONFISCATE_CARD }

  Used to indicate the desired disposition of the card following a transaction or when a connection is terminated.
- enum SCardReaderState {
  SCardReaderState.Unknown = (int)WinSCard.SCARD_UNKNOWN, SCardReaderState.Absent = (int)WinS↵Card.SCARD_ABSENT, SCardReaderState.Present = (int)WinSCard.SCARD_PRESENT, SCardReaderState.↵Swallowed = (int)WinSCard.SCARD_SWALLOWED,
  SCardReaderState.Powered = (int)WinSCard.SCARD_POWERED, SCardReaderState.Negotiable = (int)WinS↵Card.SCARD_NEGOTIABLE, SCardReaderState.Specific = (int)WinSCard.SCARD_SPECIFIC }

  Used to indicate the state of the card in the reader and the current protocol status of the card. For the latter, this is used to specify whether the card is willing to negotiate a new protocol or not as defined in ISO/IEC 7816.
- enum SCardContextScope { SCardContextScope.User = (int)WinSCard.SCARD_SCOPE_USER, SCard↵ContextScope.Terminal = (int)WinSCard.SCARD_SCOPE_TERMINAL, SCardContextScope.System = (int)Win↵SCard.SCARD_SCOPE_SYSTEM }

  Scope of the card context. Used to designate the scope of access desired within a given smart card resource manager context.
- enum SCardTrackStates {
  SCardTrackStates.Unaware = (int)WinSCard.SCARD_STATE_UNAWARE, SCardTrackStates.Ignore = (int)WinSCard.SCARD_STATE_IGNORE, SCardTrackStates.Changed = (int)WinSCard.SCARD_STATE_C↵HANGED, SCardTrackStates.Unknown = (int)WinSCard.SCARD_STATE_UNKNOWN,
  SCardTrackStates.Unavailable = (int)WinSCard.SCARD_STATE_UNAVAILABLE, SCardTrackStates.Empty = (int)WinSCard.SCARD_STATE_EMPTY, SCardTrackStates.Present = (int)WinSCard.SCARD_STATE_PRES↵ENT, SCardTrackStates.AtrMatch = (int)WinSCard.SCARD_STATE_ATRMATCH,
  SCardTrackStates.Exclusive = (int)WinSCard.SCARD_STATE_EXCLUSIVE, SCardTrackStates.InUse = (int)WinSCard.SCARD_STATE_INUSE, SCardTrackStates.Mute = (int)WinSCard.SCARD_STATE_MUTE, S↵CardTrackStates.Unpowered = (int)WinSCard.SCARD_STATE_UNPOWERED }

  The tracking state provides information about the reader state in a tracking operation such as SCardResourceManager.↵GetStatusChange. Do not confuse this enumeration with the reader state SCardReaderState.
- enum SCardChannelType {
  SCardChannelType.Unknown = 0, SCardChannelType.Serial = (int)WinSCard.SCARD_READER_TYPE_SER↵IAL, SCardChannelType.Parallel = (int)WinSCard.SCARD_READER_TYPE_PARALELL, SCardChannelType.↵Keyboard = (int)WinSCard.SCARD_READER_TYPE_KEYBOARD,
  SCardChannelType.SCSI = (int)WinSCard.SCARD_READER_TYPE_SCSI, SCardChannelType.IDE = (int)Win↵SCard.SCARD_READER_TYPE_IDE, SCardChannelType.USB = (int)WinSCard.SCARD_READER_TYPE_U↵

SB, SCardChannelType.PCMCIA = (int)WinSCard.SCARD_READER_TYPE_PCMCIA, SCardChannelType.Vendor = (int)WinSCard.SCARD_READER_TYPE_VENDOR }

Enumerates the different channel types that the card reader could be connected to.

- enum SCardMechanicalCharacteristics { SCardMechanicalCharacteristics.NothingSpecial = 0, SCard↩ MechanicalCharacteristics.Swallows = (int)WinSCard.SCARD_READER_SWALLOWS, SCardMechanical↩ Characteristics.Ejects = (int)WinSCard.SCARD_READER_EJECTS, SCardMechanicalCharacteristics.Captures = (int)WinSCard.SCARD_READER_CONFISCATES }

  A list of bit flags that indicate various mechanical characteristics of a card reader.
- enum SCardAuthDevices {
  SCardAuthDevices.NoDevices = 0x00000000, SCardAuthDevices.ReservedFU = 0x00000001, SCardAuth↩ Devices.Numeric = 0x00000002, SCardAuthDevices.Keyboard = 0x00000004,
  SCardAuthDevices.Fingerprint = 0x00000008, SCardAuthDevices.Retinal = 0x00000010, SCardAuthDevices.↩ Image = 0x00000020, SCardAuthDevices.Voice = 0x00000040,
  SCardAuthDevices.Display = 0x00000080, SCardAuthDevices.EncryptedInput = 0x00008000 }

  A list of bit flags that inform about the authentication devices available by a card reader.
- enum SCardReaderCapability {
  SCardReaderCapability.VendorName = (int)WinSCard.SCARD_ATTR_VENDOR_NAME, SCardReader↩ Capability.VendorIFDType = (int)WinSCard.SCARD_ATTR_VENDOR_IFD_TYPE, SCardReaderCapability.↩ VendorIFDVersion = (int)WinSCard.SCARD_ATTR_VENDOR_IFD_VERSION, SCardReaderCapability.Vendor↩ IFDSerialNo = (int)WinSCard.SCARD_ATTR_VENDOR_IFD_SERIAL_NO,
  SCardReaderCapability.ChannelID = (int)WinSCard.SCARD_ATTR_CHANNEL_ID, SCardReaderCapability.↩ ProtocolTypes = (int)WinSCard.SCARD_ATTR_PROTOCOL_TYPES, SCardReaderCapability.ProtocolDefault↩ CLK = (int)WinSCard.SCARD_ATTR_DEFAULT_CLK, SCardReaderCapability.ProtocolMaxCLK = (int)WinS↩ Card.SCARD_ATTR_MAX_CLK,
  SCardReaderCapability.ProtocolDefaultDataRate = (int)WinSCard.SCARD_ATTR_DEFAULT_DATA_RATE, SCardReaderCapability.ProtocolMaxDataRate = (int)WinSCard.SCARD_ATTR_MAX_DATA_RATE, SCard↩ ReaderCapability.ProtocolMaxIFSD = (int)WinSCard.SCARD_ATTR_MAX_IFSD, SCardReaderCapability.↩ PowerMgmtSupport = (int)WinSCard.SCARD_ATTR_POWER_MGMT_SUPPORT,
  SCardReaderCapability.UserToCardAuthDevice = (int)WinSCard.SCARD_ATTR_USER_TO_CARD_AUTH↩ _DEVICE, SCardReaderCapability.UserAuthInputDevice = (int)WinSCard.SCARD_ATTR_USER_AUTH_IN↩ PUT_DEVICE, SCardReaderCapability.Characteristics = (int)WinSCard.SCARD_ATTR_CHARACTERISTICS, SCardReaderCapability.CurrentProtocolType = (int)WinSCard.SCARD_ATTR_CURRENT_PROTOCOL_TYPE,
  SCardReaderCapability.CurrentCLK = (int)WinSCard.SCARD_ATTR_CURRENT_CLK, SCardReader↩ Capability.CurrentF = (int)WinSCard.SCARD_ATTR_CURRENT_F, SCardReaderCapability.CurrentD = (int)WinSCard.SCARD_ATTR_CURRENT_D, SCardReaderCapability.CurrentN = (int)WinSCard.SCARD_A↩ TTR_CURRENT_N,
  SCardReaderCapability.CurrentW = (int)WinSCard.SCARD_ATTR_CURRENT_W, SCardReaderCapability.↩ CurrentIFSC = (int)WinSCard.SCARD_ATTR_CURRENT_IFSC, SCardReaderCapability.CurrentIFSD = (int)WinSCard.SCARD_ATTR_CURRENT_IFSD, SCardReaderCapability.CurrentBWT = (int)WinSCard.SC↩ ARD_ATTR_CURRENT_BWT,
  SCardReaderCapability.CurrentCWT = (int)WinSCard.SCARD_ATTR_CURRENT_CWT, SCardReader↩ Capability.CurrentEBCEncoding = (int)WinSCard.SCARD_ATTR_CURRENT_EBC_ENCODING, SCard↩ ReaderCapability.ExtendedBWT = (int)WinSCard.SCARD_ATTR_EXTENDED_BWT, SCardReaderCapability.↩ ICCPresence = (int)WinSCard.SCARD_ATTR_ICC_PRESENCE,
  SCardReaderCapability.ICCInterfaceStatus = (int)WinSCard.SCARD_ATTR_ICC_INTERFACE_STATUS, S↩ CardReaderCapability.ATRString = (int)WinSCard.SCARD_ATTR_ATR_STRING, SCardReaderCapability.IC↩ CTypePerATR = (int)WinSCard.SCARD_ATTR_ICC_TYPE_PER_ATR, SCardReaderCapability.CurrentIOState = (int)WinSCard.SCARD_ATTR_CURRENT_IO_STATE,
  SCardReaderCapability.EscReset = (int)WinSCard.SCARD_ATTR_ESC_RESET, SCardReaderCapability.Esc↩ Cancel = (int)WinSCard.SCARD_ATTR_ESC_CANCEL, SCardReaderCapability.EscAuthRequest = (int)WinS↩ Card.SCARD_ATTR_ESC_AUTHREQUEST, SCardReaderCapability.MaxInput = (int)WinSCard.SCARD_ATT↩ R_MAXINPUT,
  SCardReaderCapability.DeviceUnit = (int)WinSCard.SCARD_ATTR_DEVICE_UNIT, SCardReaderCapability.↩ DeviceInUse = (int)WinSCard.SCARD_ATTR_DEVICE_IN_USE, SCardReaderCapability.DeviceFriendlyName

= (int)WinSCard.SCARD_ATTR_DEVICE_FRIENDLY_NAME_A, SCardReaderCapability.DeviceSystemName = (int)WinSCard.SCARD_ATTR_DEVICE_SYSTEM_NAME_A,
SCardReaderCapability.SuppressT1IFSRequest = (int)WinSCard.SCARD_ATTR_SUPRESS_T1_IFS_REQU↩
EST, SCardReaderCapability.PerfNumTransmissions = (int)WinSCard.SCARD_PERF_NUM_TRANSMISSIONS,
SCardReaderCapability.PerfBytesTransmitted = (int)WinSCard.SCARD_PERF_BYTES_TRANSMITTED, S↩
CardReaderCapability.PerfTransmissionTime = (int)WinSCard.SCARD_PERF_TRANSMISSION_TIME }

Enumeration of tags for querying or setting reader capabilities.

- enum SCardControlCodes {
SCardControlCodes.Power = (int)WinSCard.IOCTL_SMARTCARD_POWER, SCardControlCodes.GetAttribute = (int)WinSCard.IOCTL_SMARTCARD_GET_ATTRIBUTE, SCardControlCodes.SetAttribute = (int)WinSCard.↩
IOCTL_SMARTCARD_SET_ATTRIBUTE, SCardControlCodes.Confiscate = (int)WinSCard.IOCTL_SMARTC↩
ARD_CONFISCATE,
SCardControlCodes.Transmit = (int)WinSCard.IOCTL_SMARTCARD_TRANSMIT, SCardControlCodes.Eject = (int)WinSCard.IOCTL_SMARTCARD_EJECT, SCardControlCodes.Swallow = (int)WinSCard.IOCTL_SMART↩
CARD_SWALLOW, SCardControlCodes.Read = (int)WinSCard.IOCTL_SMARTCARD_READ,
SCardControlCodes.Write = (int)WinSCard.IOCTL_SMARTCARD_WRITE, SCardControlCodes.Present = (int)WinSCard.IOCTL_SMARTCARD_IS_PRESENT, SCardControlCodes.Absent = (int)WinSCard.IOCTL_↩
SMARTCARD_IS_ABSENT, SCardControlCodes.Protocol = (int)WinSCard.IOCTL_SMARTCARD_SET_PRO↩
TOCOL,
SCardControlCodes.State = (int)WinSCard.IOCTL_SMARTCARD_GET_STATE, SCardControlCodes.Get↩
LastError = (int)WinSCard.IOCTL_SMARTCARD_GET_LAST_ERROR, SCardControlCodes.GetPerfCntr = (int)WinSCard.IOCTL_SMARTCARD_GET_PERF_CNTR }

Predefined control codes for the SCardComm.Control method.

- enum SCardResponseCode {
SCardResponseCode.InternalError = (int)(WinSCard.SCARD_F_INTERNAL_ERROR & 0x0000FFFF), S↩
CardResponseCode.InvalidHandle = (int)(WinSCard.SCARD_E_INVALID_HANDLE & 0x0000FFFF), SCard↩
ResponseCode.InvalidParameter = (int)(WinSCard.SCARD_E_INVALID_PARAMETER & 0x0000FFFF), S↩
CardResponseCode.InvalidValue = (int)(WinSCard.SCARD_E_INVALID_VALUE & 0x0000FFFF),
SCardResponseCode.Cancelled = (int)(WinSCard.SCARD_E_CANCELLED & 0x0000FFFF), SCardResponse↩
Code.NoMemory = (int)(WinSCard.SCARD_E_NO_MEMORY & 0x0000FFFF), SCardResponseCode.↩
InsufficientBuffer = (int)(WinSCard.SCARD_E_INSUFFICIENT_BUFFER & 0x0000FFFF), SCardResponse↩
Code.UnknownReader = (int)(WinSCard.SCARD_E_UNKNOWN_READER & 0x0000FFFF),
SCardResponseCode.Timeout = (int)(WinSCard.SCARD_E_TIMEOUT & 0x0000FFFF), SCardResponse↩
Code.SharingViolation = (int)(WinSCard.SCARD_E_SHARING_VIOLATION & 0x0000FFFF), SCardResponse↩
Code.NoSmartCard = (int)(WinSCard.SCARD_E_NO_SMARTCARD & 0x0000FFFF), SCardResponseCode.↩
UnknownCard = (int)(WinSCard.SCARD_E_UNKNOWN_CARD & 0x0000FFFF),
SCardResponseCode.ProtoMismatch = (int)(WinSCard.SCARD_E_PROTO_MISMATCH & 0x0000FFF↩
F), SCardResponseCode.NotReady = (int)(WinSCard.SCARD_E_NOT_READY & 0x0000FFFF), SCard↩
ResponseCode.SystemCancelled = (int)(WinSCard.SCARD_E_SYSTEM_CANCELLED & 0x0000FFFF), S↩
CardResponseCode.NotTransacted = (int)(WinSCard.SCARD_E_NOT_TRANSACTED & 0x0000FFFF),
SCardResponseCode.ReaderUnavailable = (int)(WinSCard.SCARD_E_READER_UNAVAILABLE & 0x0000↩
FFFF), SCardResponseCode.UnsupportedCard = (int)(WinSCard.SCARD_W_UNSUPPORTED_CARD & 0x0000FFFF), SCardResponseCode.UnresponsiveCard = (int)(WinSCard.SCARD_W_UNRESPONSIVE_↩
CARD & 0x0000FFFF), SCardResponseCode.UnpoweredCard = (int)(WinSCard.SCARD_W_UNPOWERED_↩
CARD & 0x0000FFFF),
SCardResponseCode.ResetCard = (int)(WinSCard.SCARD_W_RESET_CARD & 0x0000FFFF), SCard↩
ResponseCode.RemovedCard = (int)(WinSCard.SCARD_W_REMOVED_CARD & 0x0000FFFF), SCard↩
ResponseCode.NoReader = (int)(WinSCard.SCARD_E_NO_READERS_AVAILABLE & 0x0000FFFF), SCard↩
ResponseCode.NoService = (int)(WinSCard.SCARD_E_NO_SERVICE & 0x0000FFFF),
SCardResponseCode.ServiceStopped = (int)(WinSCard.SCARD_E_SERVICE_STOPPED & 0x0000FFFF), S↩
CardResponseCode.CommunicationLost = (int)(WinSCard.SCARD_E_COMM_DATA_LOST & 0x0000FFFF),
SCardResponseCode.UnknownInterface = (int)(WinSCard.SCARD_E_SYSTEM_INTERFACE_UNKNOWN & 0x0000FFFF), SCardResponseCode.ReaderCommunicationAborted = (int)(WinSCard.SCARD_E_SYSTEM_↩

DEVICE_IO_ABORTED_UPON_REQUEST & 0x0000FFFF), SCardResponseCode.ReaderNotFunctioning = (int)(WinSCard.SCARD_E_SYSTEM_DEVICE_NOT_FUNCTI←ONING & 0x0000FFFF), SCardResponseCode.ReaderCallFailed = (int)(WinSCard.SCARD_E_SYSTEM_DE←VICE_CALL_FAILED & 0x0000FFFF) }

These are the well known error response codes that may appear in a SCardException.

- enum SCardPinAttributes {
  SCardPinAttributes.Numeric = 0x0000, SCardPinAttributes.Alphanumeric = 0x0100, SCardPinAttributes.Upper←Alphanumeric = 0x0200, SCardPinAttributes.Ascii = 0x0002,
  SCardPinAttributes.Unicode = 0x0000, SCardPinAttributes.BCD = 0x0001, SCardPinAttributes.LeftJustified = 0x0000, SCardPinAttributes.RightJustified = 0x0004 }

  Values that represent card pin attributes.

5.7.1 Enumeration Type Documentation

5.7.1.1 enum **Subsembly.SmartCard.PcSc.SCardAccessMode** [strong]

Values that represent card access modes.

Enumerator

Null Value of an uninitialized instance of this enumeration.

Shared Application is willing to share access to card with other applications. This is equivalent to the PC/SC constant SCARD_SHARE_SHARED.

Exclusive Application requires exclusive access to the card. This is equivalent to the PC/SC constant SCARD←_SHARE_EXCLUSIVE.

Direct Application requires connection to reader whether or not card is present. Implies exclusive access. This is equivalent to the PC/SC constant SCARD_DIRECT.

5.7.1.2 enum **Subsembly.SmartCard.PcSc.SCardAuthDevices** [strong]

A list of bit flags that inform about the authentication devices available by a card reader.

In addition to the bit values defined in this list, there may be vendor defined bits in the range 0x00000100 through 0x00004000 for vendor defined devices.

Enumerator

NoDevices No authentication devices available.

ReservedFU Reserved for future use.

Numeric Numeric (that is, PIN) pad is available.

Keyboard Full Keyboard is available.

Fingerprint Fingerprint scanner is available.

Retinal Retinal scanner is available.

Image Image scanner is available.

Voice Voice print scanner is available.

Display A display device, e.g. for showing prompts, is available.

EncryptedInput Used to indicate that encrypted input is supported.

5.7.1.3 enum Subsembly.SmartCard.PcSc.SCardChannelType [strong]

Enumerates the different channel types that the card reader could be connected to.

The actual channel type is available through the SCardComm.ChannelType property.

In addition to the channel types defined by this enumeration, a vendor defined channel type in the value range 0xF0 through 0xFF may appear. In this case the SCardComm.ChannelNumber holds a vendor defined value, too.

Enumerator

Unknown The channel type is not known or determined, yet.

Serial Card reader connected to a serial COM port. The SCardComm.ChannelNumber is the port number.

Parallel Card reader connected to a parallel LPT port. The SCardComm.ChannelNumber is the port number.

Keyboard Card reader connected through the PS/2 keyboard port. The SCardComm.ChannelNumber is zero in this case.

SCSI Card reader connected to a SCSI port. The SCardComm.ChannelNumber is the SCSI ID number. (This is Unlikely)

IDE Card reader connected to an IDE port. The SCardComm.ChannelNumber is the device number. (This is completely absurd)

USB Card reader connected to an USB port. The SCardComm.ChannelNumber is the device number.

PCMCIA Card reader connected to PCMCIA slot. The value of SCardComm.ChannelNumber is not documented by PC/SC in this case, hence you cannot count on anything useful there.

Vendor Another vendor specific channel type. This is just the first of 16 possible values in the range from 0xF0 through 0xFF.

5.7.1.4 enum Subsembly.SmartCard.PcSc.SCardContextScope [strong]

Scope of the card context. Used to designate the scope of access desired within a given smart card resource manager context.

Enumerator

User Operate within the users scope. This is equivalent to the PC/SC constant SCARD_SCOPE_USER.

Terminal Operate within the scope of a terminal. This is equivalent to the PC/SC constant SCARD_SCOPE↩ _TERMINAL. Although this constant is defined by PC/SC and in the Win32 header files, it appears to be unsupported.

System Operate within the system-wide scope. This is equivalent to the PC/SC constant SCARD_SCOPE_SY↩ STEM.

5.7.1.5 enum Subsembly.SmartCard.PcSc.SCardControlCodes [strong]

Predefined control codes for the SCardComm.Control method.

These are the control codes as they are defined in the `winsmcrd.h` header file. Any additional proprietary control codes can be easily produced by the static SCardComm.SCardCtlCode method, that resembles the behaviour of the `SCARD_CTL_CODE` macro also from `winsmcrd.h`.

Enumerator

Power IOCTL_SMARTCARD_POWER

GetAttribute IOCTL_SMARTCARD_GET_ATTRIBUTE

SetAttribute IOCTL_SMARTCARD_SET_ATTRIBUTE

Confiscate IOCTL_SMARTCARD_CONFISCATE

Transmit IOCTL_SMARTCARD_TRANSMIT

Eject IOCTL_SMARTCARD_EJECT

Swallow IOCTL_SMARTCARD_SWALLOW

Read IOCTL_SMARTCARD_READ

Write IOCTL_SMARTCARD_WRITE

Present IOCTL_SMARTCARD_IS_PRESENT

Absent IOCTL_SMARTCARD_IS_ABSENT

Protocol IOCTL_SMARTCARD_SET_PROTOCOL

State IOCTL_SMARTCARD_GET_STATE

GetLastError IOCTL_SMARTCARD_GET_LAST_ERROR

GetPerfCntr IOCTL_SMARTCARD_GET_PERF_CNTR

5.7.1.6 enum Subsembly.SmartCard.PcSc.SCardDisposition `[strong]`

Used to indicate the desired disposition of the card following a transaction or when a connection is terminated.

Enumerator

LeaveCard Dont alter card state. This is equivalent to the PC/SC constant SCARD_LEAVE_CARD.

ResetCard Reset the card. This is equivalent to the PC/SC constant SCARD_RESET_CARD.

UnpowerCard Unpower and terminate access to the card. This is equivalent to the PC/SC constant SCARD_U↩ NPOWER_CARD.

EjectCard Eject the card from the reader. This is equivalent to the PC/SC constant SCARD_EJECT_CARD.

Confiscate Used to indicate that a sophisticated commercial reader should move the card to the confiscation bin and not return it to the user. This is equivalent to the PC/SC constant SCARD_CONFISCATE_CARD. This constant is not defined in the Win32 smart card header files, hence it is probably not supported and wrong.

5.7.1.7 enum Subsembly.SmartCard.PcSc.SCardMechanicalCharacteristics `[strong]`

A list of bit flags that indicate various mechanical characteristics of a card reader.

See the PC/SC Workgroup specification part 3 for more information.

Enumerator

NothingSpecial The card reader has no special mechanical characteristics, or the mechanical characteristics are not known or determined, yet.

Swallows The card reader has a swallow mechanism.

Ejects The card reader has an ejection mechanism.

Captures The card reader has a capture mechanism that can be used to confiscate cards.

5.7.1.8 enum Subsembly.SmartCard.PcSc.SCardPinAttributes `[strong]`

Values that represent card pin attributes.

Enumerator

Numeric PIN is only composed of digits.

Alphanumeric PIN can be composed of digits and letters.

UpperAlphanumeric PIN can be composed of digits and uppercase letters.

Ascii Characters are coded in ASCII format.

Unicode Characters are coded in UNICODE format.

BCD Characters (only digits) are coded in BCD format.

LeftJustified PIN is left-justified.

RightJustified PIN is right-justified.

5.7.1.9 enum Subsembly.SmartCard.PcSc.SCardProtocolIdentifiers `[strong]`

This enumeration defines the protocols used in communication with the card.

These enum values are defined such that a protocol maps to a specific bit position so that multiple protocols may be specified by combining them with a bitwise OR operation.

Enumerator

Undefined All zero value that indicates that the protocol is not defined or known, yet. This is equivalent to the PC/SC constant SCARD_PROTOCOL_UNDEFINED.

Any Convenience value to allow T=0 or T=1

Default Special value to provide a hint to reader that it should use default communication parameters to establish communication with the card. This is equivalent to the PC/SC constant SCARD_PROTOCOL_DEFAULT.

Optimal Provides hint to reader that it should attempt to negotiate optimal communications settings with the card. This is equivalent to the PC/SC constant SCARD_PROTOCOL_OPTIMAL.

T0 ISO/IEC 7186 T=0 protocol. This is equivalent to the PC/SC constant SCARD_PROTOCOL_T0.

T1 ISO/IEC 7816 T=1 protocol. This is equivalent to the PC/SC constant SCARD_PROTOCOL_T1.

Raw Used for direct access. This is equivalent to the PC/SC constant SCARD_PROTOCOL_RAW.

5.7.1.10 enum Subsembly.SmartCard.PcSc.SCardReaderCapability `[strong]`

Enumeration of tags for querying or setting reader capabilities.

Note that whereas the PC/SC specification uses the terms Device Capabilities or Reader Capabilities, the Win32 Smart Card API just calls these Attributes.

Note also, that the more interesting capabilities are directly available as properties of the SCardComm class. Therefore this enumeraton is only needed for advanced applications.

Enumerator

VendorName The vendor name. ASCII string with a max length of 32 characters. The string is not necessarily nul terminated, but it may be.

VendorIFDType Vendor specified card reader type. ASCII string with a max length of 32 characters. The string is not necessarily nul terminated, but it may be.

VendorIFDVersion Vendor specified card reader version number. A four byte unsigned integer encoded as 0X↩ MMmmbbbb where: MM = major version, mm = minor version, and bbbb = build number.

VendorIFDSerialNo Serial number of the card reader. ASCII string with a max length of 32 characters. The string is not necessarily zero terminated, but it may be.

ChannelID Communications channel identification. A four byte unsigned integer encoded as 0xDDDDCCC↩ C where: DDDD = data channel type, and CCCC = channel number. See SCardChannelType enumeration for appropriate values.

ProtocolTypes Asynchronous Protocol Types supported. A four byte unsigned integer encoded as 0x0RRRP↩ PPP where: RRR is reserved for future use and should be 0x000. PPPP encodes the supported protocol types. A 1 in a given bit position indicates support for the associated ISO protocol. Example: 0x00000003 indicates support for T=0 and T=1. This is the only compliant value that may be returned by devices at this time. All other values (T=2, T=14, T=15, and so on), are not specified by PC/SC, yet, and must be handled by vendor-supplied drivers.

ProtocolDefaultCLK Default smart card clock frequency in KHz encoded as a four byte little endian integer value. Example: 3.58 MHz is encoded as the integer value 3580.

ProtocolMaxCLK Maximum supported smart card clock frequency in KHz encoded as a four byte little endian integer value.

ProtocolDefaultDataRate Default smart card I/O data rate in bits per second encoded as a four byte little endian integer.

ProtocolMaxDataRate Maximum supported smart card I/O date rate in bits per second encoded as a four byte little endian integer.

ProtocolMaxIFSD Unsigned four byte integer indicating maximum IFSD supported by the card reader.

PowerMgmtSupport Four byte integer. If 0, device does not support power down while smart card inserted. If non-zero, device does support it.

UserToCardAuthDevice User to Card Authentication Devices. Unsigned four byte integer that is the result of a bitwise OR operation performed on the applicable values from the enumeration SCardAuthDevices.

UserAuthInputDevice User Authentication Input Device. Unsigned four byte integer that is the result of a bitwise OR operation performed on the applicable values from the enumeration SCardAuthDevices.

Characteristics Mechanical characteristics supported. Unsigned four byte integer that is the result of a bitwise OR operation. See SCardMechanicalCharacteristics enumeration for appropriate flag values.

CurrentProtocolType Current Protocol Type. A four byte unsigned integer encoded in same manner as ProtocolTypes. It is illegal to specify more than a single protocol in this value.

CurrentCLK Current smart card clock frequency in KHz encoded as a four byte little endian integer value. Example: 3.58 MHz is encoded as the integer value 3580.

CurrentF Current F (clock conversion factor) encoded as a four byte little endian integer.

CurrentD Current D (bit rate conversion factor) encoded as a four byte little endian integer.

CurrentN Current N (guard time factor) encoded as a four byte little endian integer.

CurrentW Current W (work waiting time) encoded as a four byte little endian integer. Only valid if current protocol is to T=0.

CurrentIFSC Current IFSC (information field size card) encoded as a four byte little endian integer. Only valid if current protocol is T=1.

CurrentIFSD Current IFSD (information field size Reader) encoded as a four byte little endian integer. Only valid if current protocol is T=1.

CurrentBWT Current BWT (block waiting time) encoded as a four byte little endian integer. Only valid if current protocol is T=1.

CurrentCWT Current CWT (character waiting time) encoded as a four byte little endian integer. Only valid if current protocol is T=1.

CurrentEBCEncoding Current EBC encoding encoded as 0 = LRC and 1 = CRC. Only valid if current protocol is T=1.

ExtendedBWT Extended BWT (block waiting time) encoded as a four byte little endian integer. Only valid if current protocol is T=1. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

ICCPresence Smart card presence status. A single byte encoded according to te following table:

0	not present
1	card present but not swallowed (applies only if the card reader supports card swallowing)
2	card present (and swallowed if the card reader supports card swallowing)
4	card confiscated

ICCInterfaceStatus Smart card interface status. A single byte where 0 means contact inactive and 1 means contact active.

ATRString Contains the ATR string (sequence of bytes) as returned by the smart card.

ICCTypePerATR Smart card type, based on ATR sequence. A single byte where 0 = unknown smart card type, 1 = 7816 Asynchronous, and 2 = 7816 Synchronous. Other values are reserved.

CurrentIOState Current I/O state. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

EscReset ESC Reset. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

EscCancel ESC Cancel. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

EscAuthRequest ESC Authentication Request. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

MaxInput Max Input. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

DeviceUnit Instance of this vendor's reader attached to the computer. The first instance will be device unit 0, the next will be unit 1 (if it is the same brand of reader) and so on. Two different brands of readers will both have 0 for this value. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

DeviceInUse Reserved for future use. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

DeviceFriendlyName Reader's friendly name returned as an ASCII string. Note that the max length of the returned string is not specified. I have an example where the returned string even exceeds the max length of 32 bytes for capability attrbutes.

This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

DeviceSystemName Reader's system name returned as an ASCII string. Note that the max length of the returned string is not specified. I have an example where the returned string even exceeds the max length of 32 bytes for capability attrbutes.

This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

SuppressT1IFSRequest Suppress sending of T=1 IFSD packet from the reader to the card. (Can be used if the currently inserted card does not support an IFSD request.) This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

PerfNumTransmissions Number of transmission. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

PerfBytesTransmitted Number of bytes transmitted. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

PerfTransmissionTime Transmission time. This capability is not defined by the PC/SC specification, but by the Win32 Smart Card API header files.

5.7.1.11 enum Subsembly.SmartCard.PcSc.SCardReaderState [strong]

Used to indicate the state of the card in the reader and the current protocol status of the card. For the latter, this is used to specify whether the card is willing to negotiate a new protocol or not as defined in ISO/IEC 7816.

Do not confuse this enumeration with card tracking state SCardTrackStates.

Enumerator

Unknown Special all zero value that implies that current state of the reader is not known. This value is not defined by PC/SC but is used in the Win32 Smart Card API.

Absent No card is in the reader. This is equivalent to the PC/SC constant SCARD_ABSENT.

Present A card presence is detected in the reader. However, the card is not fully inserted, yet, and therefore is not properly positioned for operation. This is equivalent to the PC/SC constant SCARD_PRESENT.

Swallowed A card is in the reader and is properly positioned for operation. However, the card is not powered, yet. This is equivalent to the PC/SC constant SCARD_SWALLOWED.

Powered A card is in the reader and has been powered. This is equivalent to the PC/SC constant SCARD_PO←WERED.

Negotiable This value implies the card has been reset and is awaiting PTS negotiation. This is equivalent to the PC/SC constant SCARD_NEGOTIABLEMODE.

Specific This value implies the card has been reset and specific communication protocols have been established. This is equivalent to the PC/SC constant SCARD_SPECIFICMODE.

5.7.1.12 enum Subsembly.SmartCard.PcSc.SCardResponseCode [strong]

These are the well known error response codes that may appear in a SCardException.

Note that other values, not defined in this enumeration, may appear inside a SCardException. Other values are derived from the actual native response code, if it lies in the range 0x80100000 through 0x8010FFFF simply by taking the low word as an unsigned value.

Enumerator

InternalError In internal error occurred. Example: reader does not respond with SW to an invalid command; GetUID() on 5x25

InvalidHandle The supplied handle was invalid.

InvalidParameter One or more of the supplied parameters could not be properly interpreted.

InvalidValue One or more of the supplied parameters values could not be properly interpreted.

Cancelled The action was cancelled by an SCardTrack.Cancel request.

NoMemory Not enough memory available to complete this command.

InsufficientBuffer The data buffer to receive returned data is too small for the returned data.

UnknownReader The specified reader name is not recognized.

Timeout The specified timeout value has expired.

SharingViolation The smart card cannot be accessed because of other connections outstanding

NoSmartCard The operation requires a smart card, but no smart card is currently in the device.

UnknownCard The specified smart card name is not recognized

ProtoMismatch The requested protocols are incompatible with the protocol currently in use with the smart card.

NotReady The reader or smart card is not ready to accept commands.

SystemCancelled The action was cancelled by the system, presumably to log off or shut down.

NotTransacted An attempt was made to end a non-existent transaction.

ReaderUnavailable The specified reader is not currently available for use.

UnsupportedCard The reader cannot communicate with the card, due to ATR configuration conflicts. This error may be cleared by the SCardComm.Reconnect service.

UnresponsiveCard The smart card is not responding to a reset. This error may be cleared by the SCardComm.↩ Reconnect service.

UnpoweredCard Power has been removed from the card, so that further communication is not possible. This error may be cleared by the SCardComm.Reconnect service.

ResetCard The smart card has been reset, so any shared state information is invalid. This error may be cleared by the SCardComm.Reconnect service.

RemovedCard The smart card has been removed, so that further communication is not possible. This error may be cleared by the SCardComm.Reconnect service.

NoReader Cannot find a smart card reader. This error can occur if there is no reader at all or if a reader gets disconnected;

NoService The Smart Card Resource Manager is not running.

ServiceStopped The Smart Card Resource Manager has shut down.

CommunicationLost A communications error with the smart card has been detected. Retry the operation.

UnknownInterface The Smart Card Resource Manager interface is not available. This code can be generated by the system if the service is shut down during operation.

ReaderCommunicationAborted The I/O operation has been aborted because of either a thread exit or an application request. This can happen when the reader is disconnected during communication with the card.

ReaderNotFunctioning A reader that is believed to be connected is not functioning. This can happen when the reader is disconnected during communication with the card.

ReaderCallFailed A remote procedure call during communication with a reader that is believed to be connected failed. This can happen when the reader is disconnected during communication with the card.

5.7.1.13 enum Subsembly.SmartCard.PcSc.SCardTrackStates `[strong]`

The tracking state provides information about the reader state in a tracking operation such as SCardResource↩ Manager.GetStatusChange. Do not confuse this enumeration with the reader state SCardReaderState.

It seems that the high word of the reader state returned from the Win32 Smart Card API holds a count that is incremented for each state change. This is probably there to detect situations such as when a card is removed and reinserted again within two queries, which would look as if the physical state never changed. Therefore you will have to isolate the low word of the actual value in order to see the individual flags.

Enumerator

Unaware The application is unaware of the current state, and would like to know. The use of this value results in an immediate return from state transition-monitoring services. This is represented by all bits set to zero. This is equivalent to the PC/SC constant SCARD_STATE_UNAWARE.

Ignore The application is not interested in this reader, and it should not be considered during monitoring operations. If this bit value is set, all other bits are ignored. This is equivalent to the PC/SC constant SCARD_ST↩ ATE_IGNORE.

Changed This implies that there is a difference between the state input by the calling application, and the current state. When this bit is set, the application may assume a significant state change has occurred on this reader. This is equivalent to the PC/SC constant SCARD_STATE_CHANGED.

Unknown This implies that the given reader name is not recognized by the smart card resource manager. If this bit is set, then Changed will also be set. This is equivalent to the PC/SC constant SCARD_STATE_UNKNOWN.

Unavailable This implies that the actual state of this reader is not available. If this bit is set, then neither of the following bits will be set. This is equivalent to the PC/SC constant SCARD_STATE_UNAVAILABLE.

Empty This implies that there is no card in the reader. If this bit is set, all the following bits will be clear. This is equivalent to the PC/SC constant SCARD_STATE_EMPTY.

Present This implies that there is a card in the reader. This is equivalent to the PC/SC constant SCARD_STAT↩
E_PRESENT.

AtrMatch This implies that there is a card in the reader with an ATR matching one of the registered target cards. This is equivalent to the PC/SC constant SCARD_STATE_ATRMATCH. This bit is returned only by the PC/↩
SC LocateCard() method which is not supported by the smartcard.dll Class Library. Hence it will never show up.

Exclusive This implies that the card in the reader is allocated for exclusive use by another application. If this bit is set, Present will also be set. This is equivalent to the PC/SC constant SCARD_STATE_EXCLUSIVE.

InUse This implies that the card in the reader is in use by one or more other applications, but may be connected to in shared mode. If this bit is set, Present will also be set. This is equivalent to the PC/SC constant SCAR↩
D_STATE_INUSE.

Mute There is an unresponsive card in the reader. This flag is only defined by the Win32 Smart Card API and not by PC/SC.

Unpowered This implies that the card in the reader has not been powered up. This flag is only defined by the Win32 Smart Card API and not by PC/SC.

Chapter 6

Class Documentation

6.1 SmartCardAPI.CardModule.CAC.CacCardEdge Class Reference

Exposes high level PIV card edge in line with End-Point client application programming interface defined in SP 800-73-3. ref: `http://csrc.nist.gov/publications/nistpubs/800-73-3/sp800-73-3_PART1_↩ piv-card-applic-namespace-date-model-rep.pdf` Method names are conform with 800-73-3 as much as possible to indicate the exact PIV API middleware function targeted with this implementation.CardTerminalException.

Public Member Functions

- CacCardEdge (CardHandle hCard)

    Constructor for high-level DoD CAC card access.
- byte[ ] GetProperties ()

    Helper function to retrieve CAC Access Control Rule properties using Get ACR APDU described in GSC-IS p5-32.
- byte[ ] GetData (byte[ ] objectID)

    Reads DoD CAC data object.
- string GetVersion ()

    For SP 800-73-3 Part 3 conformant PIV Middleware this method would just return "800-73-3 Client API". We add Card↩ Werk to the string to indicate a proprietary API version. Our goal though is to implement a PIV API close to SP 800-73-3 specs.
- bool SelectObject (byte[ ] objectId)

    Select container managed by an applet on a VM card using Select Object APDU described in GSC- IS p5-28.
- bool SelectApplet (byte[ ] aid)

    Selects application on a multi-application card. The AID can be an RID or RID + PIX. This method can be used to explore the card and find out what applets are loaded in case the host application wants to support multiple card configurations. This method can also be used as main entry point for data access as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3 Note that cards might select a default applet at power-up. Selecting a different applet might change scope, exposing a different card edge.

6.1.1 Detailed Description

Exposes high level PIV card edge in line with End-Point client application programming interface defined in SP 800-73-3. ref: `http://csrc.nist.gov/publications/nistpubs/800-73-3/sp800-73-3_PART1_↩ piv-card-applic-namespace-date-model-rep.pdf` Method names are conform with 800-73-3 as much as possible to indicate the exact PIV API middleware function targeted with this implementation.CardTerminalException.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 SmartCardAPI.CardModule.CAC.CacCardEdge.CacCardEdge (CardHandle *hCard*) [inline]

Constructor for high-level DoD CAC card access.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

hCard	The card.

6.1.3 Member Function Documentation

6.1.3.1 byte [ ] SmartCardAPI.CardModule.CAC.CacCardEdge.GetData (byte[ ] *objectID*) [inline]

Reads DoD CAC data object.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ApplicationException	Thrown when an Application error condition occurs.

Parameters

objectID	Identifier for the object.

Returns

An array of byte.

6.1.3.2 byte [ ] SmartCardAPI.CardModule.CAC.CacCardEdge.GetProperties ( ) [inline]

Helper function to retrieve CAC Access Control Rule properties using Get ACR APDU described in GSC-IS p5-32.

Returns

An array of byte.

6.1.3.3 string SmartCardAPI.CardModule.CAC.CacCardEdge.GetVersion ( ) [inline]

For SP 800-73-3 Part 3 conformant PIV Middleware this method would just return “800-73-3 Client API”. We add Card↩
Werk to the string to indicate a proprietary API version. Our goal though is to implement a PIV API close to SP 800-73-3 specs.

Returns

The version.

6.1.3.4 bool SmartCardAPI.CardModule.CAC.CacCardEdge.SelectApplet (byte[ ] *aid*) [inline]

Selects application on a multi-application card. The AID can be an RID or RID + PIX. This method can be used to explore the card and find out what applets are loaded in case the host application wants to support multiple card configurations. This method can also be used as main entry point for data access as defined in PIV End-Point Client-↩ Application Programming Interface SP 800-73-3 Note that cards might select a default applet at power-up. Selecting a different applet might change scope, exposing a different card edge.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

aid	The aid.

Returns

True if it succeeds, false if it fails.

6.1.3.5 bool SmartCardAPI.CardModule.CAC.CacCardEdge.SelectObject (byte[ ] *objectId*) [inline]

Select container managed by an applet on a VM card using Select Object APDU described in GSC- IS p5-28.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

objectId	Identifier for the object.

Returns

True if it succeeds, false if it fails.

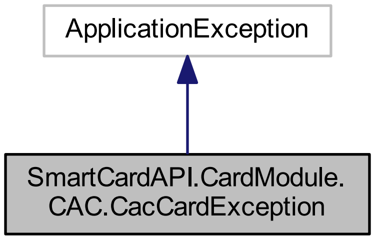
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/CA↩ C/CacCardEdge.cs

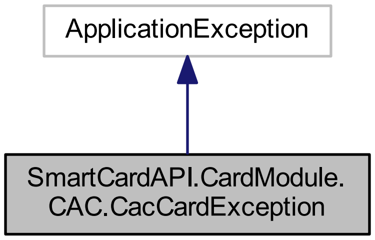
6.2 SmartCardAPI.CardModule.CAC.CacCardException Class Reference

Whenever a SmartCardAPI.CardModule.CAC API error is returned an exception of this class is thrown.

Inheritance diagram for SmartCardAPI.CardModule.CAC.CacCardException:



Collaboration diagram for SmartCardAPI.CardModule.CAC.CacCardException:



Public Member Functions

- CacCardException (CacCardErrorCode nErrorCode)

  Creates a new PivCardException instance from a SmartCardAPI.CardModule.PIV API error code.
- override string ToString ()

  Builds a string that describes the PIV API error that caused this exception.

Properties

- CacCardErrorCode ErrorCode `[get]`

  The error code that was returned by the failing SmartCardAPI.CardModule.CAC API call.

6.2.1 Detailed Description

Whenever a SmartCardAPI.CardModule.CAC API error is returned an exception of this class is thrown.

See also

T:System.ApplicationException

6.2.2 Constructor & Destructor Documentation

6.2.2.1 SmartCardAPI.CardModule.CAC.CacCardException.CacCardException (CacCardErrorCode *nErrorCode*) [inline]

Creates a new PivCardException instance from a SmartCardAPI.CardModule.PIV API error code.

Parameters

nErrorCode	The error indicating integer value returned by a SmartCardAPI.CardModule.PIV API call. Use a type cast to coerce the integer value into a PivCardErrorCode type.

6.2.3 Member Function Documentation

6.2.3.1 override string SmartCardAPI.CardModule.CAC.CacCardException.ToString ( ) [inline]

Builds a string that describes the PIV API error that caused this exception.

Returns

Friendly description of an error on PIV API level.

See also

M:System.Exception.ToString()

6.2.4 Property Documentation

6.2.4.1 CacCardErrorCode SmartCardAPI.CardModule.CAC.CacCardException.ErrorCode [get]

The error code that was returned by the failing SmartCardAPI.CardModule.CAC API call.

The error code.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/CA↩ C/CacCardException.cs

6.3 SmartCardAPI.CardModule.CAC.CacCardPersonInstance Class Reference

This is a helper class to parse, analyze and display CAC Person Instance data. It processes data that typically resides in the Person Instance data containter. However, it does not require a physical card connection but rather is a method to process raw data read from the card in previous transaction.

Public Member Functions

- CacCardPersonInstance (byte[ ] personInstanceData)
  Constructor.
- override string ToString ()
  Formats this response APDU, including all its data, in a nice human readable fashion.

Properties

- string PersonFirstName `[get]`
  Allows public access to card holder first name.
- string PersonMiddleName `[get]`
  Allows public access to card holder middle name.
- string PersonLastName `[get]`
  Allows public access to card holder last name.
- string PersonCadencyName `[get]`
  Allows public access to card holder cadency name.
- string PersonIdentifier `[get]`
  Allows public access to card holder identifier.
- string DateOfBirth `[get]`
  Allows public access to date of birth.
- string SexCategoryCode `[get]`
  Allows public access to SexCategoryCode.
- string PersonIdentifierTypeCode `[get]`
  Allows public access to PersonIdentifierTypeCode.
- string BloodTypeCode `[get]`
  Allows public access to BloodTypeCode.
- string DodEdiPersonIdentifier `[get]`
  Allows public access to BloodTypeCode.
- string OrganDonor `[get]`
  Gets the organ donor info.
- string IdentificationCardIssueDate `[get]`
  Gets the identification card issue date.
- string IdentificationCardExpirationDate `[get]`
  Gets the identification card expiration date.
- string DateDemographicDataWasLoadedOnChip `[get]`
  Gets the date demographic data was loaded on chip.
- string DateDemographicDataOnChipExpires `[get]`
  Gets the date demographic data on chip expires.
- string CardInstanceIdentifier `[get]`
  Gets the identifier of the card instance.

6.3.1 Detailed Description

This is a helper class to parse, analyze and display CAC Person Instance data. It processes data that typically resides in the Person Instance data containter. However, it does not require a physical card connection but rather is a method to process raw data read from the card in previous transaction.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 SmartCardAPI.CardModule.CAC.CacCardPersonInstance.CacCardPersonInstance (byte[ ] *personInstanceData*) [inline]

Constructor.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

personInstance←Data	Information describing the person instance.

6.3.3 Member Function Documentation

6.3.3.1 override string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.ToString ( ) [inline]

Formats this response APDU, including all its data, in a nice human readable fashion.

Returns

Returns a nicely formatted string usable for tracing output.

See also

M:System.Object.ToString()

6.3.4 Property Documentation

6.3.4.1 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.BloodTypeCode [get]

Allows public access to BloodTypeCode.

The blood type code.

6.3.4.2 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.CardInstanceIdentifier [get]

Gets the identifier of the card instance.

The identifier of the card instance.

6.3.4.3 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.DateDemographicDataOnChipExpires [get]

Gets the date demographic data on chip expires.

The date demographic data on chip expires.

6.3.4.4 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.DateDemographicDataWasLoadedOnChip [get]

Gets the date demographic data was loaded on chip.

The date demographic data was loaded on chip.

6.3.4.5 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.DateOfBirth `[get]`

Allows public access to date of birth.

The date of birth.

6.3.4.6 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.DodEdiPersonIdentifier `[get]`

Allows public access to BloodTypeCode.

The identifier of the dod edi person.

6.3.4.7 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.IdentificationCardExpirationDate `[get]`

Gets the identification card expiration date.

The identification card expiration date.

6.3.4.8 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.IdentificationCardIssueDate `[get]`

Gets the identification card issue date.

The identification card issue date.

6.3.4.9 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.OrganDonor `[get]`

Gets the organ donor info.

The organ donor.

6.3.4.10 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.PersonCadencyName `[get]`

Allows public access to card holder cadency name.

The name of the person cadency.

6.3.4.11 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.PersonFirstName `[get]`

Allows public access to card holder first name.

The name of the person first.

6.3.4.12 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.PersonIdentifier `[get]`

Allows public access to card holder identifier.

The identifier of the person.

6.3.4.13 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.PersonIdentifierTypeCode `[get]`

Allows public access to PersonIdentifierTypeCode.

The person identifier type code.

6.3.4.14 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.PersonLastName `[get]`

Allows public access to card holder last name.

The name of the person last.

6.3.4.15 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.PersonMiddleName `[get]`

Allows public access to card holder middle name.

The name of the person middle.

6.3.4.16 string SmartCardAPI.CardModule.CAC.CacCardPersonInstance.SexCategoryCode `[get]`

Allows public access to SexCategoryCode.

The sex category code.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/CA↩ C/CacCardPersonInstance.cs

6.4 SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance Class Reference

This is a helper class to parse, analyze and display CAC Personnel Instance data. It processes data that typically resides in the Personnel Instance data containter. However, it does not require a physical card connection but rather is a method to process raw data read from the card in previous transaction.

Public Member Functions

- CacCardPersonnelInstance (byte[ ] personnelInstanceData)

  Constructor.
- override string ToString ()

  Formats this response APDU, including all its data, in a nice human readable fashion.

Properties

- string DoDContractorFunctionCode `[get]`

  Retrieves the DoD Contractor Function Code for the cardholder. Allows public access to DoD Contractor Function Code described in CAC API section 4.2.
- string USGovernmentAgencySubagencyCode `[get]`

  Retrieves the U. S. Government agency code for the cardholder. Allows public access to US Government Agency/↩ Subagency Code described in CAC API section 4.2.

- string BranchOfServiceCode `[get]`

  Retrieves the branch of service for the cardholder. Allows public access to Branch of Service Code described in CAC API section 4.2.
- string PayGradeCode `[get]`

  Retrieves the pay grade code for the cardholder. Allows public access to Pay Grade Code described in CAC API section 4.2.
- string RankCode `[get]`

  Retrieves the rank code for the cardholder Allows public access to Rank Code described in CAC API section 4.2.
- string PersonnelCategoryCode `[get]`

  Retrieves the personnel category code for the cardholder. Allows public access to Personnel Category Code described in CAC API section 4.2.
- string NonUSGovernmentAgencySubagencyCode `[get]`

  Retrieves the non-U. S. Government agency code for the cardholder. Allows public access to Non-US Government Agency/Subagency Code described in CAC API section 4.2.
- string PayPlanCode `[get]`

  Retrieves the pay plan code for the cardholder. Allows public access to Pay Plan Code described in CAC API section 4.2.
- string PersonnelEntitlementConditionCode `[get]`

  Retrieves the personnel entitlement condition type code for the cardholder. Allows public access to Personnel Entitlement Condition Code described in CAC API section 4.2.

6.4.1 Detailed Description

This is a helper class to parse, analyze and display CAC Personnel Instance data. It processes data that typically resides in the Personnel Instance data containter. However, it does not require a physical card connection but rather is a method to process raw data read from the card in previous transaction.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.CacCardPersonnelInstance (byte[ ] *personnelInstanceData*) `[inline]`

Constructor.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

personnel↩ InstanceData	Information describing the personnel instance.

6.4.3 Member Function Documentation

6.4.3.1 override string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.ToString ( ) `[inline]`

Formats this response APDU, including all its data, in a nice human readable fashion.

Returns

Returns a nicely formatted string usable for tracing output.

See also

M:System.Object.ToString()

6.4.4 Property Documentation

6.4.4.1 string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.BranchOfServiceCode [get]

Retrieves the branch of service for the cardholder. Allows public access to Branch of Service Code described in CAC API section 4.2.

The branch of service code.

6.4.4.2 string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.DoDContractorFunctionCode [get]

Retrieves the DoD Contractor Function Code for the cardholder. Allows public access to DoD Contractor Function Code described in CAC API section 4.2.

The do d contractor function code.

6.4.4.3 string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.NonUSGovernmentAgencySubagencyCode [get]

Retrieves the non-U. S. Government agency code for the cardholder. Allows public access to Non-US Government Agency/Subagency Code described in CAC API section 4.2.

The non us government agency subagency code.

6.4.4.4 string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.PayGradeCode [get]

Retrieves the pay grade code for the cardholder. Allows public access to Pay Grade Code described in CAC API section 4.2.

The pay grade code.

6.4.4.5 string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.PayPlanCode [get]

Retrieves the pay plan code for the cardholder. Allows public access to Pay Plan Code described in CAC API section 4.2.

The pay plan code.

6.4.4.6 string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.PersonnelCategoryCode [get]

Retrieves the personnel category code for the cardholder. Allows public access to Personnel Category Code described in CAC API section 4.2.

The personnel category code.

6.4.4.7 string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.PersonnelEntitlementConditionCode [get]

Retrieves the personnel entitlement condition type code for the cardholder. Allows public access to Personnel Entitlement Condition Code described in CAC API section 4.2.

The personnel entitlement condition code.

6.4.4.8 string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.RankCode [get]

Retrieves the rank code for the cardholder Allows public access to Rank Code described in CAC API section 4.2.

The rank code.

6.4.4.9 string SmartCardAPI.CardModule.CAC.CacCardPersonnelInstance.USGovernmentAgencySubagencyCode [get]

Retrieves the U. S. Government agency code for the cardholder. Allows public access to US Government Agency/↩ Subagency Code described in CAC API section 4.2.

The us government agency subagency code.

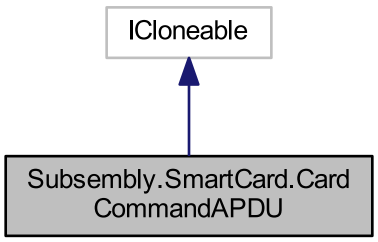
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/CA↩ C/CarCardPersonnelInstance.cs

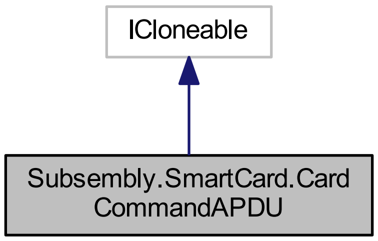
6.5 Subsembly.SmartCard.CardCommandAPDU Class Reference

Generic utility class for building and handling of smart card APDUs according to ISO 7816-4. Command APDUs and response APDUs are implemented.

Inheritance diagram for Subsembly.SmartCard.CardCommandAPDU:



Collaboration diagram for Subsembly.SmartCard.CardCommandAPDU:



Public Member Functions

- CardCommandAPDU (byte nCLA, byte nINS, byte nP1, byte nP2)

    Construct CardCommandAPDU instance for an ISO 7816-4 case 1 "NODATA_NODATA" command.
- CardCommandAPDU (byte nCLA, byte nINS, byte nP1, byte nP2, int nLe)

    Construct CardCommandAPDU instance for an ISO 7816-4 (short or extended) case 2 "NODATA_DATA" command.
- CardCommandAPDU (byte nCLA, byte nINS, byte nP1, byte nP2, byte[ ] vbData)

    Construct CardCommandAPDU instance for an ISO 7816-4 (short or extended) case 3 "DATA_NODATA" command.
- CardCommandAPDU (byte nCLA, byte nINS, byte nP1, byte nP2, byte[ ] vbData, int nLe)

    Construct CardCommandAPDU instance for an ISO 7816-4 (short or extended) case 4 "DATA_DATA" command.
- object Clone ()

    Creates a new CardCommandAPDU instance that is a complete and deep copy of this CardCommandAPDU instance.
- byte[ ] GenerateBytes ()

    Builds the final command APDU into a newly allocated byte array.
- byte[ ] GetData ()

    Returns the data field of an ISO 7816-4 command APDU.
- void SetData (byte[ ] vbData)

    Adds or replaces the data field of this command APDU.
- void SetData (byte[ ] vbData, int nOffset, int nLength)

    Adds or replaces the data field of this command APDU.
- void AppendData (byte bData)

    Appends a single data byte to the data field of this command APDU.
- void AppendData (byte[ ] vbData)

    Appends the given data block to the data field of this command APDU.
- void AppendData (byte[ ] vbData, int nOffset, int nLength)

    Appends the given data block to the data field of this command APDU.
- override string ToString ()

    Formats this command APDU, including all its data, in a nice human readable fashion.
- string GetInstructionName ()

    Returns the ISO 7816-4 instruction name for the instruction byte in this command APDU.

Static Public Member Functions

- static CardCommandAPDU ParseBytes (byte[ ] vbCommandAPDU)

  Creates a CardCommandAPDU instance by parsing a command APDU given as a byte array.
- static CardCommandAPDU ParseBytes (byte[ ] vbCommandAPDU, int nOffset, int nLength)

  Creates a CardCommandAPDU instance by parsing a command APDU given as a range inside a byte array.
- static string GetInstructionName (byte nCLA, byte nINS)

  Returns the instruction name for a given instruction byte. We currently support names defined in ISO 7816-4 1995 as well as PC/SC part 3. PC/SC part 3 kicks in, when CLA=0xFF is detected.

Public Attributes

- const int MAXSHORTLENGTH = 261

  The max length in bytes of an ISO 7816-4 command APDU with short length fields.
- const int MAXEXTENDEDLENGTH = 65544

  The max length in bytes of an ISO 7816-4 command APDU with extended length fields.
- const int MINLENGTH = 4

  The min length of any ISO 7816-4 command APDU.
- const int MAXSHORTLC = 255

  The max size of the data field of an ISO 7816-4 command APDU with short length fields.
- const int MAXEXTENDEDLC = 65535

  The max size of the data field of an ISO 7816-4 command APDU with extended length fields.
- const int MAXSHORTLE = 256

  The max value for the field Le of an ISO 7816-4 command APDU with short length fields.
- const int MAXEXTENDEDLE = 65536

  The max value for the field Le of an ISO 7816-4 command APDU with extended length fields.

Properties

- byte CLA `[get, set]`

  Access to the class byte of an ISO 7816-4 command APDU.
- byte INS `[get, set]`

  Access to the instruction byte of an ISO 7816-4 command APDU.
- byte P1 `[get, set]`

  Access to the first parameter byte of an ISO 7816-4 command APDU.
- byte P2 `[get, set]`

  Access to the second parameter byte of an ISO 7816-4 command APDU.
- int Lc `[get]`

  Access to the length field Lc of an ISO 7816-4 command APDU. This is the actual length of the data field.
- int Le `[get, set]`

  Access to the length field Le of an ISO 7816-4 command APDU. This is the expected length of the response data field.
- int Case `[get]`

  Indicates the "case" of this ISO 7816-4 command APDU.
- bool IsExtended `[get]`

  Indicates whether this is an extended ISO 7816-4 command APDU.

6.5.1 Detailed Description

Generic utility class for building and handling of smart card APDUs according to ISO 7816-4. Command APDUs and response APDUs are implemented.

This class supports command APDUs with short and extended length fields. If the data field size is up to and including a maximum size of 255 bytes, and the length of the expected data Le is in the range from 1 through 256, then a command APDU with a short length fields will be built. If either the data field size is 256 bytes or more, or Le is 257 or larger, then a command APDU with extended length fields will be built.

This class defines several useful constants that provide minimum and maximum limits of command APDU sizes and fields.

In order to create an all new command APDU from scratch start with one of the constructors of this class. In order to parse a raw command APDU from a byte array use one of the overloaded static ParseBytes methods.

Use the method GenerateBytes in order to build a raw command APDU byte array to be sent to a smart card. Note that parsing and rebuilding a commmand APDU may lead to a different byte representation.

See also

T:System.ICloneable

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Subsembly.SmartCard.CardCommandAPDU.CardCommandAPDU (byte *nCLA*, byte *nINS*, byte *nP1*, byte *nP2*) [inline]

Construct CardCommandAPDU instance for an ISO 7816-4 case 1 "NODATA_NODATA" command.

Parameters

nCLA	Class byte.
nINS	Instruction byte.
nP1	First parameter byte.
nP2	Second parameter byte.

6.5.2.2 Subsembly.SmartCard.CardCommandAPDU.CardCommandAPDU (byte *nCLA*, byte *nINS*, byte *nP1*, byte *nP2*, int *nLe*) [inline]

Construct CardCommandAPDU instance for an ISO 7816-4 (short or extended) case 2 "NODATA_DATA" command.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nCLA	Class byte.
nINS	Instruction byte.
nP1	First parameter byte.

nP2	Second parameter byte.
nLe	

Length of expected data. If the value is in the range 0 through 256, then a command APDU with a short length field will be built. If the value is in the range 257 through 65536, then a command APDU with an extended length field will be built.

If no length field Le shall be included, then the case 1 constructor must be used.

6.5.2.3 Subsembly.SmartCard.CardCommandAPDU.CardCommandAPDU (byte *nCLA,* byte *nINS,* byte *nP1,* byte *nP2,* byte[ ] *vbData*) [inline]

Construct CardCommandAPDU instance for an ISO 7816-4 (short or extended) case 3 "DATA_NODATA" command.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange←Exception	Thrown when one or more arguments are outside the required range.

Parameters

nCLA	Class byte.
nINS	Instruction byte.
nP1	First parameter byte.
nP2	Second parameter byte.
vbData	Complete data field of command APDU. Up to and including a maximum length of 255 bytes, a command APDU with a short length field will be built. For data lengths from 256 bytes up to and including 65535 bytes, a command APDU with an extended length field will be built. Longer byte arrays and empty byte arrays are not permitted.

6.5.2.4 Subsembly.SmartCard.CardCommandAPDU.CardCommandAPDU (byte *nCLA,* byte *nINS,* byte *nP1,* byte *nP2,* byte[ ] *vbData,* int *nLe*) [inline]

Construct CardCommandAPDU instance for an ISO 7816-4 (short or extended) case 4 "DATA_DATA" command.

If either the length of vbData or nLe exceed the limit for a command APDU with short length fields, then a command APDU with extended length fields will be built. Only if both are within the limits for short length fields, then a command APDU with short length fields will be built.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange←Exception	Thrown when one or more arguments are outside the required range.

Parameters

nCLA	Class byte.
nINS	Instruction byte.
nP1	First parameter byte.

nP2	Second parameter byte.
vbData	Complete data field of command APDU. Up to and including a maximum length of 255 bytes, a command APDU with a short length field will be built. For data lengths from 256 bytes up to and including 65535 bytes, a command APDU with an extended length field will be built. Longer byte arrays and empty byte arrays are not permitted.
nLe	

Length of expected data. If the value is in the range 0 through 256, then a command APDU with a short length field will be built. If the value is in the range 257 through 65536, then a command APDU with an extended length field will be built.

If no length field Le shall be included, then the case 1 constructor must be used.

6.5.3 Member Function Documentation

6.5.3.1 void Subsembly.SmartCard.CardCommandAPDU.AppendData (byte *bData*) [inline]

Appends a single data byte to the data field of this command APDU.

Parameters

bData	The data byte to be appended.

6.5.3.2 void Subsembly.SmartCard.CardCommandAPDU.AppendData (byte[ ] *vbData*) [inline]

Appends the given data block to the data field of this command APDU.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	

The data to be appended to the data field for this command APDU. The length Lc is incremented by the total length of this byte array. This parameter must not be null.

Up to and including a total maximum length of 255 bytes, a command APDU with a short length field will be built. For data lengths from 256 bytes up to and including 65535 bytes, a command APDU with an extended length field will be built. Longer total sizes are not permitted.

6.5.3.3 void Subsembly.SmartCard.CardCommandAPDU.AppendData (byte[ ] *vbData*, int *nOffset*, int *nLength*) [inline]

Appends the given data block to the data field of this command APDU.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	A byte array that provides the data to be appended to the data field of this command APDU. This parameter must not be null. This method allocates a new array and copies the data from the given array. It does not keep any reference to the given array.
nOffset	Offset into byte array where the first byte shall be taken from. This must not be negative and must be within the given byte array.
nLength	Number of bytes to take from the given byte array starting at nOffset. This must not be zero or negative. The entire length must fall inside the given byte array. The length must not be longer than MAXEXTENDEDLC = 65535 bytes.

6.5.3.4 object Subsembly.SmartCard.CardCommandAPDU.Clone ( ) `[inline]`

Creates a new CardCommandAPDU instance that is a complete and deep copy of this CardCommandAPDU instance.

Returns

Clone.

See also

M:System.ICloneable.Clone()

6.5.3.5 byte [ ] Subsembly.SmartCard.CardCommandAPDU.GenerateBytes ( ) `[inline]`

Builds the final command APDU into a newly allocated byte array.

Returns

Byte array that contains the complete command APDU. This is what is actually sent to the smart card. This is never null. The minimum size of the returned array is MINLENGTH = 4 the maximum size of the returned array is MAXEXTENDEDLENGTH = 65544.

6.5.3.6 byte [ ] Subsembly.SmartCard.CardCommandAPDU.GetData ( ) `[inline]`

Returns the data field of an ISO 7816-4 command APDU.

Returns

If the command APDU does not include a data field, then null is returned. Otherwise a reference to the byte array with the data field of the command APDU is returned.

6.5.3.7 string Subsembly.SmartCard.CardCommandAPDU.GetInstructionName ( ) `[inline]`

Returns the ISO 7816-4 instruction name for the instruction byte in this command APDU.

Accessing the instruction byte implicitly causes that the entire command APDU will be verified whether it is a correct ISO 7816-4 command APDU. If it is not a correct ISO 7816-4 command APDU, then an exception will be thrown.

Returns

Returns a string with the all uppercase instruction name as defined in ISO 7816-4 1995 Table 11. If the instruction byte is not known, then null is returned.

See also

GetInstructionName(byte,byte)

6.5.3.8 static string Subsembly.SmartCard.CardCommandAPDU.GetInstructionName (byte *nCLA*, byte *nINS*) `[inline]`, `[static]`

Returns the instruction name for a given instruction byte. We currently support names defined in ISO 7816-4 1995 as well as PC/SC part 3. PC/SC part 3 kicks in, when CLA=0xFF is detected.

Parameters

nCLA	The class byte to indicate ISO 7816, PC/SC part 3.
nINS	The instruction byte.

Returns

Returns a string with the all uppercase instruction name as defined in ISO 7816-4 1995 Table 11 or PC/SC part 3 if applicable. If the instruction byte is not known, then null is returned.

6.5.3.9 static CardCommandAPDU Subsembly.SmartCard.CardCommandAPDU.ParseBytes (byte[ ] *vbCommandAPDU*) `[inline]`, `[static]`

Creates a CardCommandAPDU instance by parsing a command APDU given as a byte array.

Exceptions

ArgumentNullException	The parameter vbCommandAPDU was `null`.

Parameters

vbCommandA↔PDU	The complete command APDU just as it shall be sent to the smart card. The parameter must not be null.

Returns

Returns a new CardCommandAPDU instance that holds te parsed representation of the given raw command APDU.

6.5.3.10 static CardCommandAPDU Subsembly.SmartCard.CardCommandAPDU.ParseBytes (byte[ ] *vbCommandAPDU*, int *nOffset*, int *nLength*) `[inline]`, `[static]`

Creates a CardCommandAPDU instance by parsing a command APDU given as a range inside a byte array.

Exceptions

ArgumentNullException	The parameter vbCommandAPDU was `null`.
ArgumentOutOfRange↩ Exception	The given command APDU is not a valid ISO 7816-4 command APDU.

Parameters

vbCommandA↩ PDU	Array that contains the command APDU. The parameter must not be null.
nOffset	Offset into the byte array where the command APDU begins.
nLength	Length of the command APDU inside the byte array.

Returns

Returns a new CardCommandAPDU instance that holds the parsed representation of the given raw command APDU.

6.5.3.11 void Subsembly.SmartCard.CardCommandAPDU.SetData (byte[ ] *vbData*) [inline]

Adds or replaces the data field of this command APDU.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	The data field to be set for this command APDU. The length Lc is derived from the total length of this byte array. If this parameter is null, then any existing data field will be removed.

The CardCommandAPDU will keep a direct reference to the given byte array and will not make a copy of it.

Up to and including a maximum length of 255 bytes, a command APDU with a short length field will be built. For data lengths from 256 bytes up to and including 65535 bytes, a command APDU with an extended length field will be built. Longer byte arrays are not permitted.

6.5.3.12 void Subsembly.SmartCard.CardCommandAPDU.SetData (byte[ ] *vbData*, int *nOffset*, int *nLength*) [inline]

Adds or replaces the data field of this command APDU.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	A byte array that provides the data for the data field for this command APDU. This parameter must not be null. This method allocates a new array and copies the data from the given array. It does not keep any reference to the given array.

nOffset	Offset into byte array where the first byte shall be taken from. This must not be negative and must be within the given byte array.
nLength	Number of bytes to take from the given byte array starting at nOffset. This must not be zero or negative. The entire length must fall inside the given byte array. The length must not be longer than MAXEXTENDEDLC = 65535 bytes.

6.5.3.13 override string Subsembly.SmartCard.CardCommandAPDU.ToString ( ) `[inline]`

Formats this command APDU, including all its data, in a nice human readable fashion.

Returns

Returns a nicely formatted string usable for tracing output. If the command contains an ISO VERIFY command (INS = 0x20), then the command data (the PIN) will not be included in the returned string.

See also

M:System.Object.ToString()

6.5.4 Member Data Documentation

6.5.4.1 const int Subsembly.SmartCard.CardCommandAPDU.MINLENGTH = 4

The min length of any ISO 7816-4 command APDU.

6.5.5 Property Documentation

6.5.5.1 int Subsembly.SmartCard.CardCommandAPDU.Case `[get]`

Indicates the "case" of this ISO 7816-4 command APDU.

Command header only.

Command header plus Le field.

Command header plus Lc with data field.

Command header plus Lc with data field and Le field.

ISO 7816-4 divides command APDUs into four cases as follows: <list type="table"> <item> 1

The case.

6.5.5.2 byte Subsembly.SmartCard.CardCommandAPDU.CLA `[get]`, `[set]`

Access to the class byte of an ISO 7816-4 command APDU.

The class byte is always the first byte of an ISO 7816-4 command APDU.

6.5.5.3 byte Subsembly.SmartCard.CardCommandAPDU.INS `[get]`, `[set]`

Access to the instruction byte of an ISO 7816-4 command APDU.

The instruction byte is always the second byte of an ISO 7816-4 command APDU.

6.5.5.4 bool Subsembly.SmartCard.CardCommandAPDU.IsExtended `[get]`

Indicates whether this is an extended ISO 7816-4 command APDU.

True if this object is extended, false if not.

6.5.5.5 int Subsembly.SmartCard.CardCommandAPDU.Lc `[get]`

Access to the length field Lc of an ISO 7816-4 command APDU. This is the actual length of the data field.

Depending on whether the command APDU uses short or extended length fields the range of this value is from zero up to 255 (short length field) or 65535 (extended length field). A length of zero means that no data field and no length field Lc is present.

6.5.5.6 int Subsembly.SmartCard.CardCommandAPDU.Le `[get]`, `[set]`

Access to the length field Le of an ISO 7816-4 command APDU. This is the expected length of the response data field.

Depending on whether the command APDU uses short or extended length fields the range of this value is from zero up to 256 (short length field) or 65536 (extended length field). A negative length means that no length field Le is present.

6.5.5.7 byte Subsembly.SmartCard.CardCommandAPDU.P1 `[get]`, `[set]`

Access to the first parameter byte of an ISO 7816-4 command APDU.

The first parameter byte is always the third byte of an ISO 7816-4 command APDU.

6.5.5.8 byte Subsembly.SmartCard.CardCommandAPDU.P2 `[get]`, `[set]`

Access to the second parameter byte of an ISO 7816-4 command APDU.

The second parameter byte is always the fourth byte of an ISO 7816-4 command APDU.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardCommand↩
APDU.cs

6.6 Subsembly.SmartCard.CardCrypto Class Reference

A card crypto.

6.6.1 Detailed Description

A card crypto.

not implemented.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardCrypto.cs

6.7 Subsembly.SmartCard.CardCtApiConfig Class Reference

Collects information about a CT-API card terminal configuration. This information usually is stored in the CardTerminal↩ Registry.

Public Member Functions

- CardCtApiConfig ()

  Construct CardCtApiConfig with all default values.
- CardCtApiConfig (string sCtApiDll, int nCtApiPort)

  Convenience constructor.
- void Load (string sXmlConfig)

  Loads the CT-API configuration XML string.
- string Save ()

  Gets the save.
- override bool Equals (object rhs)

  Tests if this object is considered equal to another.
- override int GetHashCode ()

  Calculates a hash code for this object.

Properties

- string CtApiDll `[get, set]`

  Name of the native CT-API DLL file. The name must be given such that it can be loaded by the native Win32 LoadLibrary function. Hence it must either include a complete path, or the DLL must be located in the DLL search path of the system.
- int CtApiCTN `[get, set]`

  Arbitrary card terminal number to be passed to the CT-API.
- int CtApiPort `[get, set]`

  Port number that identifies the physical connector that the desired card terminal is connected to.
- string CtInitProcName `[get, set]`

  Actual procedure name of the CT_init function.
- string CtDataProcName `[get, set]`

  Actual procedure name of the CT_data function.
- string CtCloseProcName `[get, set]`

  Actual procedure name of the CT_close function.

6.7.1 Detailed Description

Collects information about a CT-API card terminal configuration. This information usually is stored in the CardTerminal↩ Registry.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Subsembly.SmartCard.CardCtApiConfig.CardCtApiConfig ( ) `[inline]`

Construct CardCtApiConfig with all default values.

6.7.2.2 Subsembly.SmartCard.CardCtApiConfig.CardCtApiConfig (string *sCtApiDll*, int *nCtApiPort*) [inline]

Convenience constructor.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange←Exception	Thrown when one or more arguments are outside the required range.

Parameters

sCtApiDll	.
nCtApiPort	.

6.7.3 Member Function Documentation

6.7.3.1 override bool Subsembly.SmartCard.CardCtApiConfig.Equals (object *rhs*) [inline]

Tests if this object is considered equal to another.

Parameters

rhs	.

Returns

True if the objects are considered equal, false if they are not.

See also

M:System.Object.Equals(object)

6.7.3.2 override int Subsembly.SmartCard.CardCtApiConfig.GetHashCode ( ) [inline]

Calculates a hash code for this object.

Returns

A hash code for this object.

See also

M:System.Object.GetHashCode()

6.7.3.3 void Subsembly.SmartCard.CardCtApiConfig.Load (string *sXmlConfig*) [inline]

Loads the CT-API configuration XML string.

Parameters

sXmlConfig	.

6.7.3.4 string Subsembly.SmartCard.CardCtApiConfig.Save ( ) [inline]

Gets the save.

Returns

A string.

6.7.4 Property Documentation

6.7.4.1 int Subsembly.SmartCard.CardCtApiConfig.CtApiCTN [get], [set]

Arbitrary card terminal number to be passed to the CT-API.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

The value is passed through to all CT-API function calls. According to the CT-API specification the card terminal number must be in the range from zero through 65535 inclusive. Some readers may need special values.

6.7.4.2 string Subsembly.SmartCard.CardCtApiConfig.CtApiDll [get], [set]

Name of the native CT-API DLL file. The name must be given such that it can be loaded by the native Win32 LoadLibrary function. Hence it must either include a complete path, or the DLL must be located in the DLL search path of the system.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

The ct API DLL.

6.7.4.3 int Subsembly.SmartCard.CardCtApiConfig.CtApiPort [get], [set]

Port number that identifies the physical connector that the desired card terminal is connected to.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

This can be the number of a serial port or some other manufacturer specific assignment. The actual meaning of the port number is not defined by the CT-API and always is manufacturer specific.

The port number must be in the range from zero through 65535 inclusive.

6.7.4.4 string Subsembly.SmartCard.CardCtApiConfig.CtCloseProcName [get], [set]

Actual procedure name of the CT_close function.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

If this is null, then the default name "CT_close" will be used.

6.7.4.5 string Subsembly.SmartCard.CardCtApiConfig.CtDataProcName `[get], [set]`

Actual procedure name of the CT_data function.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

If this is null, then the default name "CT_data" will be used.

6.7.4.6 string Subsembly.SmartCard.CardCtApiConfig.CtInitProcName `[get], [set]`

Actual procedure name of the CT_init function.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

If this is null, then the default name "CT_init" will be used.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardCtApi↩ Config.cs

6.8 Subsembly.SmartCard.CardDataObject Class Reference

Generic utility class for evaluating and generating Basic-Encoding-Rules (BER) encoded Tag- Length-Value (TLV) data objects.

Public Member Functions

- int GetIntegerValue ()

  Retrieve the primitive value of this data object as an integer encoded according to ASN.1.
- int GetBitStringValue ()

  Retrieve the primitive value of this data object as a bit string encoded according to ASN.1.
- string GetStringValue (Encoding aEncoding)

  Retrieve the primitive value of this data object as a string.
- void SetStringValue (Encoding aEncoding, string sValue)

  Set the primitive value of this data object from a string.
- int ComputeTotalSize (CardDataObjectEncoding nEncoding)

  Computes the total size of the encoded data object including its tag, the length field and the value data.
- CardDataObject (int nTag, byte[ ] vbValue)

Constructs a primitive data object with the given tag and data.
- CardDataObject (int nTag)

  Constructs an empty constructed data object.
- byte[ ] Generate (CardDataObjectEncoding nEncoding)

  Encodes this data object using TLV encoding into a byte array.
- CardDataObject Find (int nTag, bool fRecurse)

  Finds a contained data object with a given tag.
- override string ToString ()

  Convert this object into a string representation.

Static Public Member Functions

- static CardDataObject Parse (byte[ ] vbData, ref int nOffset, ref int nLength, CardDataObjectEncoding nEncoding)

  Parses a single primitive or constructed TLV encoded data object from the given raw data.
- static CardDataObject Parse (byte[ ] vbData)

  Parses a single primitive or constructed BER-TLV encoded data object from the given raw data.
- static CardDataObject ParseSimple (byte[ ] vbData)

  Parses a single SIMPLE-TLV encoded data object from the given raw data.

Properties

- int Tag `[get, set]`

  Provides access to the tag of this data object.
- int Length `[get]`

  Provides the length of the value data of a primitive data object.
- byte[ ] Value `[get, set]`

  Provides access to the value data of a primitive data object.
- CardDataObjectList List `[get]`

  Provides the list of data objects contained in this constructed data object.
- bool IsConstructed `[get]`

  Indicates whether this data object is a constructed data object.

6.8.1 Detailed Description

Generic utility class for evaluating and generating Basic-Encoding-Rules (BER) encoded Tag- Length-Value (TLV) data objects.

This utilitiy class is very simple and may not suite all needs for dealing with TLV structures. Anyway it can be used to process most BER-TLV or SIMPLE-TLV encoded data objects.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Subsembly.SmartCard.CardDataObject.CardDataObject (int *nTag,* byte[ ] *vbValue*) `[inline]`

Constructs a primitive data object with the given tag and data.

Exceptions

ArgumentOutOfRange←→ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nTag	Complete tag. This must not be the special zero tag. This must be in the range from 1 through 16777215 (0xFFFFFF). If the data object shall be encoded using SIMPLE-TLV encoding rules, then the tag must be in the range from 1 through 254.
vbValue	Complete value data. May be null if this shall be an empty data object. The new CardDataObject will not copy that array, but assumes logical ownership of the passed array reference. This array must be bigger than 65535 bytes.

6.8.2.2 Subsembly.SmartCard.CardDataObject.CardDataObject (int *nTag*) `[inline]`

Constructs an empty constructed data object.

This constructor must be used to create constructed data objects. After creating an empty constructed data object its contents may be added through the List property.

Exceptions

ArgumentOutOfRange←→ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nTag	Tag of the constructed data object or zero to indicate that this shall be an untagged data object list. Thus the value must be in the range from 0 through 16777215 (0xFFFFFF).

6.8.3 Member Function Documentation

6.8.3.1 int Subsembly.SmartCard.CardDataObject.ComputeTotalSize (CardDataObjectEncoding *nEncoding*) `[inline]`

Computes the total size of the encoded data object including its tag, the length field and the value data.

Parameters

nEncoding	Selects the encoding for which the total size shall be computed. Different encodings may produce a different total size.

Returns

Returns the total byte size which is at least 2.

6.8.3.2 CardDataObject Subsembly.SmartCard.CardDataObject.Find (int *nTag,* bool *fRecurse*) `[inline]`

Finds a contained data object with a given tag.

Calling this method implies that this data object is a constructed data object that contains other data objects. If it is not constructed, then an InvalidOperationException will be thrown.

Exceptions

InvalidOperationException	Thrown when the requested operation is invalid.

Parameters

nTag	Complete tag to find.
fRecurse	If true, then any contained data objects are checked.

Returns

If the sought data object was found, then it is wrapped with a new instance of this class and returned to the caller. If the data object was not found, then null is returned.

6.8.3.3 byte [ ] Subsembly.SmartCard.CardDataObject.Generate (CardDataObjectEncoding *nEncoding*) `[inline]`

Encodes this data object using TLV encoding into a byte array.

If this data object represents an untagged data object list, then a packed sequence BER- TLV encoded data objects will be generated.

Parameters

nEncoding	Selects the TLV encoding to generate.

Returns

A byte[].

6.8.3.4 int Subsembly.SmartCard.CardDataObject.GetBitStringValue ( ) `[inline]`

Retrieve the primitive value of this data object as a bit string encoded according to ASN.1.

Exceptions

FormatException	The value field was longer than the possible representation.

Returns

The bit string value.

6.8.3.5 int Subsembly.SmartCard.CardDataObject.GetIntegerValue ( ) `[inline]`

Retrieve the primitive value of this data object as an integer encoded according to ASN.1.

Exceptions

FormatException	The value field was longer than the possible representation.

Returns

The integer value.

6.8.3.6 string Subsembly.SmartCard.CardDataObject.GetStringValue (Encoding *aEncoding*) [inline]

Retrieve the primitive value of this data object as a string.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

aEncoding	The text encoding that shall be used in order to convert the binary data into a string representation. This must not be `null`.

Returns

If the Value is `null`, then `null` is returned. Otherwise the value is converted to a string representation and the resulting string is returned.

6.8.3.7 static CardDataObject Subsembly.SmartCard.CardDataObject.Parse (byte[ ] *vbData*, ref int *nOffset*, ref int *nLength*, CardDataObjectEncoding *nEncoding*) `[inline]`, `[static]`

Parses a single primitive or constructed TLV encoded data object from the given raw data.

If the data is BER-TLV encoded and the parsed tag indicates a constructed data object, then it will be recursively parsed until all contained data objects have also been parsed. Thus effectively returning a tree of CardDataObject instances.

Any empty filler bytes with the value 0x00 or 0xFF that are preceeding the tag will be skipped. Inside constructed data objects any empty filler bytes before, between or after the actual BER-TLV data objects are skipped, too.

If the parsed data is an invalid TLV encoding, or is an incomplete TLV data object, then a System.FormatException will be thrown.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.
FormatException	Thrown when the format of the ? is incorrect.

Parameters

vbData	Provides the raw binary data to be parsed. Must not be null.
nOffset	[in,out] Offset in data array where parsing shall begin. This is updated to the offset of the first byte after the parsed data object on return. Thus it is possible to repeatedly call this method in order to parse a sequence of data objects.
nLength	[in,out] Maximum number of bytes to consume from the given data array. If the parsed data object is bigger than the remaining length, then a System.FormatException will be thrown. On return this parameter is reduced by the amount of bytes actually consumed. The initial must not be gibber than 65535 or be negative.
nEncoding	The expected type of TLV encoding. Through this parameter the caller can specify whether the data is BER-TLV or SIMPLE-TLV encoded.

Returns

Returns the parsed TLV data object inside a newly created CardDataObject instance. All data is copied from the given vbData array into newly created arrays owned by the created CardDataObject instance.
If the nLength was zero, or the data only contains empty filler bytes, then null is returned. In this case the returned final nLength will be zero, too.

6.8.3.8 static CardDataObject Subsembly.SmartCard.CardDataObject.Parse (byte[ ] *vbData*) `[inline], [static]`

Parses a single primitive or constructed BER-TLV encoded data object from the given raw data.

Parameters

vbData	Provides the raw binary data to be parsed. Must not be null.

Returns

Returns the parsed BER-TLV data object inside a newly created CardDataObject instance. All data is copied from the given vbData array into newly created arrays owned by the created CardDataObject instance.
If the data contains only empty filler bytes, then null is returned.

See also

Parse(byte[ ],ref int,ref int,CardDataObjectEncoding)

6.8.3.9 static CardDataObject Subsembly.SmartCard.CardDataObject.ParseSimple (byte[ ] *vbData*) `[inline]`, `[static]`

Parses a single SIMPLE-TLV encoded data object from the given raw data.

Parameters

vbData	Provides the raw binary data to be parsed. Must not be null.

Returns

Returns the parsed SIMPLE-TLV data object inside a newly created CardDataObject instance. All data is copied from the given vbData array into a newly created array owned by the created CardDataObject instance.
If the data contains only empty filler bytes, then null is returned.

See also

Parse(byte[ ],ref int,ref int,CardDataObjectEncoding)

6.8.3.10 void Subsembly.SmartCard.CardDataObject.SetStringValue (Encoding *aEncoding*, string *sValue*) `[inline]`

Set the primitive value of this data object from a string.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
InvalidOperationException	Thrown when the requested operation is invalid.

Parameters

aEncoding	The text encoding that shall be used in order to convert the given string into a binary representation. This must not be `null`.
sValue	The string value to set. This may be `null`. If this parameter is `null`, then the Value will be set to `null`, too.

6.8.3.11 override string Subsembly.SmartCard.CardDataObject.ToString ( ) `[inline]`

Convert this object into a string representation.

Returns

A string that represents this object.

See also

M:System.Object.ToString()

6.8.4 Property Documentation

6.8.4.1 bool Subsembly.SmartCard.CardDataObject.IsConstructed `[get]`

Indicates whether this data object is a constructed data object.

This value is true whenever this data object contains other data objects. If this data object contains raw binary data, then this value is false.

6.8.4.2 int Subsembly.SmartCard.CardDataObject.Length `[get]`

Provides the length of the value data of a primitive data object.

Note that the Length can never be set, but is always derived from the actual content of this data object.

Only if this is a primitive data object, then the byte length of its value data is returned. If this is a constructed data object then `null` is returned.

6.8.4.3 CardDataObjectList Subsembly.SmartCard.CardDataObject.List `[get]`

Provides the list of data objects contained in this constructed data object.

Returns a reference to the CardDataObjectList instance that holds the data objects contained in this constructed data object. If this is a primitive (non-constructed) data object, then `null` is returned.

6.8.4.4 int Subsembly.SmartCard.CardDataObject.Tag `[get]`, `[set]`

Provides access to the tag of this data object.

Not all Tag values that can be stored in this property can be encoded with all supported CardDataObjectEncoding schemes. For example, when using CardDataObjectEncoding.Simple encoding, the Tag value must be in the range from 0x01 through 0xFE.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

The Tag value must be in the range from 1 through 0xFFFFFF. An attempt to set any other value will throw an exception.

6.8.4.5 byte [ ] Subsembly.SmartCard.CardDataObject.Value `[get]`, `[set]`

Provides access to the value data of a primitive data object.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.
InvalidOperationException	Thrown when the requested operation is invalid.

Returns a reference to the value data owned by this primitive data object. If this primitive data object does not contain any data then this property may be `null`. If this is a constructed data object, then this property returns `null`.

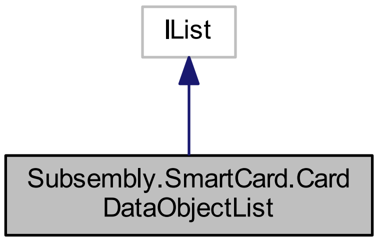
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardData↩ Object.cs

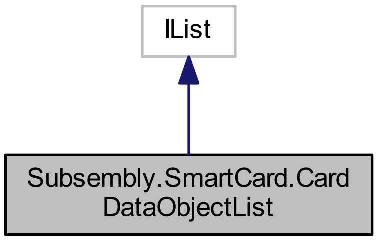
6.9 Subsembly.SmartCard.CardDataObjectList Class Reference

Generic utility class for evaluating and generating a sequence of Tag-Length-Value (TLV) data objects which are stored as CardDataObject instances.

Inheritance diagram for Subsembly.SmartCard.CardDataObjectList:



Collaboration diagram for Subsembly.SmartCard.CardDataObjectList:



Public Member Functions

- CardDataObjectList ()

    Creates an empty data item list.
- void Parse (byte[ ] vbData, ref int nOffset, ref int nLength, CardDataObjectEncoding nEncoding)

    Parses a sequence of TLV encoded data objects.
- void Parse (byte[ ] vbData)

    Parses a sequence of TLV encoded data objects.
- void ParseSimple (byte[ ] vbData)

    Parses a sequence of SIMPLE-TLV encoded data objects.
- int ComputeTotalSize (CardDataObjectEncoding nEncoding)

    Calculates the total byte size required for encoding all data objects in this list.
- byte[ ] Generate (CardDataObjectEncoding nEncoding)

    Generates a byte array for a given TLV encoding type.
- void Add (CardDataObject aDO)

    Adds another child data object to the end of this data object list.
- void Add (int nTag, byte[ ] vbValue)

    Adds a primitive data object with the given tag and data to the end of this data object list.
- CardDataObject Find (int nTag, bool fRecurse)

    Searches for the first match.
- void RemoveAt (int nIndex)

    Removes the T:System.Collections.IList item at the specified index.
- void Insert (int nIndex, object aObject)

    Inserts an item to the T:System.Collections.IList at the specified index.
- void Remove (object aObject)

    Removes the first occurrence of a specific object from the T:System.Collections.IList.
- bool Contains (object aObject)

    Determines whether the T:System.Collections.IList contains a specific value.
- void Clear ()

    Removes all items from the T:System.Collections.IList.
- int IndexOf (object aObject)

    Determines the index of a specific item in the T:System.Collections.IList.
- int Add (object aObject)

    Adds an item to the T:System.Collections.IList.
- void CopyTo (Array aArray, int nIndex)

    Copies to.
- IEnumerator GetEnumerator ()

    Gets the enumerator.

Properties

- bool IsReadOnly `[get]`

    Gets a value indicating whether this object is read only.
- object this[int nIndex] `[get, set]`

    Indexer to get or set items within this collection using array index syntax.
- bool IsFixedSize `[get]`

    Gets a value indicating whether this object is fixed size.

- bool IsSynchronized `[get]`

  Gets a value indicating whether this object is synchronized.
- int Count `[get]`

  Gets the number of.
- object SyncRoot `[get]`

  Gets the synchronize root.

6.9.1 Detailed Description

Generic utility class for evaluating and generating a sequence of Tag-Length-Value (TLV) data objects which are stored as CardDataObject instances.

6.9.2 Member Function Documentation

6.9.2.1 void Subsembly.SmartCard.CardDataObjectList.Add (CardDataObject *aDO*) `[inline]`

Adds another child data object to the end of this data object list.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

aDO	The data object to be added. This must not be null.

6.9.2.2 void Subsembly.SmartCard.CardDataObjectList.Add (int *nTag*, byte[ ] *vbValue*) `[inline]`

Adds a primitive data object with the given tag and data to the end of this data object list.

Parameters

nTag	Complete tag. This must not be the special zero tag. This must be in the range from 1 through 16777215 (0xFFFFFF). If the data object shall be encoded using SIMPLE-TLV encoding rules, then the tag must be in the range from 1 through 254.
vbValue	Complete value data. May be null if this shall be an empty data object. The new CardDataObject will not copy that array, but assumes logical ownership of the passed array reference. This array must be bigger than 65535 bytes.

6.9.2.3 int Subsembly.SmartCard.CardDataObjectList.Add (object *aObject*) `[inline]`

Adds an item to the T:System.Collections.IList.

Parameters

aObject	The object to add to the T:System.Collections.IList.

Returns

The position into which the new element was inserted, or -1 to indicate that the item was not inserted into the collection.

See also

M:System.Collections.IList.Add(object)

6.9.2.4 void Subsembly.SmartCard.CardDataObjectList.Clear ( ) `[inline]`

Removes all items from the T:System.Collections.IList.

See also

M:System.Collections.IList.Clear()

6.9.2.5 int Subsembly.SmartCard.CardDataObjectList.ComputeTotalSize (CardDataObjectEncoding *nEncoding*) `[inline]`

Calculates the total byte size required for encoding all data objects in this list.

Parameters

nEncoding	Selects the encoding for which the total size shall be computed. Different encodings may produce a different total size.

Returns

Returns the byte size in the range from 0 up to the maximum length supported.

6.9.2.6 bool Subsembly.SmartCard.CardDataObjectList.Contains (object *aObject*) `[inline]`

Determines whether the T:System.Collections.IList contains a specific value.

Parameters

aObject	The object to locate in the T:System.Collections.IList.

Returns

true if the T:System.Object is found in the T:System.Collections.IList; otherwise, false.

See also

M:System.Collections.IList.Contains(object)

6.9.2.7 void Subsembly.SmartCard.CardDataObjectList.CopyTo (Array *aArray,* int *nIndex*) `[inline]`

Copies to.

Parameters

aArray	The array.
nIndex	.

6.9.2.8 CardDataObject Subsembly.SmartCard.CardDataObjectList.Find (int *nTag*, bool *fRecurse*) [inline]

Searches for the first match.

Parameters

nTag	Complete tag. This must not be the special zero tag. This must be in the range from 1 through 16777215 (0xFFFFFF). If the data object shall be encoded using SIMPLE-TLV encoding rules, then the tag must be in the range from 1 through 254.
fRecurse	True to process recursively, false to process locally only.

Returns

A CardDataObject.

6.9.2.9 byte [ ] Subsembly.SmartCard.CardDataObjectList.Generate (CardDataObjectEncoding *nEncoding*) [inline]

Generates a byte array for a given TLV encoding type.

Parameters

nEncoding	The expected type of TLV encoding. Through this parameter the caller can specify whether the data is BER-TLV or SIMPLE-TLV encoded.

Returns

A byte[].

6.9.2.10 IEnumerator Subsembly.SmartCard.CardDataObjectList.GetEnumerator ( ) [inline]

Gets the enumerator.

Returns

The enumerator.

6.9.2.11 int Subsembly.SmartCard.CardDataObjectList.IndexOf (object *aObject*) [inline]

Determines the index of a specific item in the T:System.Collections.IList.

Parameters

aObject	The object to locate in the T:System.Collections.IList.

Returns

The index of if found in the list, otherwise, -1.

See also

M:System.Collections.IList.IndexOf(object)

6.9.2.12 void Subsembly.SmartCard.CardDataObjectList.Insert (int *nIndex,* object *aObject*) [inline]

Inserts an item to the T:System.Collections.IList at the specified index.

Parameters

nIndex	The zero-based index at which value should be inserted.
aObject	The object to insert into the T:System.Collections.IList.

See also

M:System.Collections.IList.Insert(int,object)

6.9.2.13 void Subsembly.SmartCard.CardDataObjectList.Parse (byte[ ] *vbData,* ref int *nOffset,* ref int *nLength,* CardDataObjectEncoding *nEncoding*) [inline]

Parses a sequence of TLV encoded data objects.

If the parsed tag of any TLV data object of the parsed sequence indicates a constructed data object, then it will be recursively parsed until all contained data objects have been parsed. Thus effectively creating a tree of CardData←ObjectList and CardDataObject instances.

Any empty filler bytes with the value 0x00 or 0xFF before, between or after the actual TLV data objects are skipped. This is also true for the content of any constructed data objects.

If the parsed data contains an invalid TLV encoding, or ends with an incomplete TLV data object, then a System.←FormatException will be thrown.

All parsed data objects are added at the end of this list. If the nLength was zero, or the data only contains empty filler bytes, then nothing will be added.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange←Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	Provides the raw binary data to be parsed. Must not be null.

nOffset	[in,out] Offset in data array where parsing shall begin. This is updated to the offset of the first byte after the parsed data, which is always the same as adding nLength to the initial value of nOffset.
nLength	[in,out] Number of bytes to consume from the given data array. If the final parsed data object is bigger than the remaining length, then a System.FormatException will be thrown. On return this parameter is reduced by the amount of bytes actually consumed and thus is always be zero.
nEncoding	The expected type of TLV encoding. Through this parameter the caller can specify whether the data is BER-TLV or SIMPLE-TLV encoded.

6.9.2.14 void Subsembly.SmartCard.CardDataObjectList.Parse (byte[ ] *vbData*) `[inline]`

Parses a sequence of TLV encoded data objects.

If the parsed tag of any TLV data object of the parsed sequence indicates a constructed data object, then it will be recursively parsed until all contained data objects have been parsed. Thus effectively creating a tree of CardData↩ObjectList and CardDataObject instances.

Any empty filler bytes with the value 0x00 or 0xFF before, between or after the actual TLV data objects are skipped. This is also true for the content of any constructed data objects.

If the parsed data contains an invalid TLV encoding, or ends with an incomplete TLV data object, then a System.↩FormatException will be thrown.

All parsed data objects are added at the end of this list. If the nLength was zero, or the data only contains empty filler bytes, then nothing will be added.

Parameters

vbData	Provides the raw binary data to be parsed. Must not be null.

See also

Parse(byte[ ],ref int,ref int,CardDataObjectEncoding)

6.9.2.15 void Subsembly.SmartCard.CardDataObjectList.ParseSimple (byte[ ] *vbData*) `[inline]`

Parses a sequence of SIMPLE-TLV encoded data objects.

Parameters

vbData	Provides the raw binary data to be parsed. Must not be null.

See also

Parse(byte[ ],ref int,ref int,CardDataObjectEncoding)

6.9.2.16 void Subsembly.SmartCard.CardDataObjectList.Remove (object *aObject*) `[inline]`

Removes the first occurrence of a specific object from the T:System.Collections.IList.

Parameters

aObject	The object to remove from the T:System.Collections.IList.

See also

M:System.Collections.IList.Remove(object)

6.9.2.17 void Subsembly.SmartCard.CardDataObjectList.RemoveAt (int *nIndex*) [inline]

Removes the T:System.Collections.IList item at the specified index.

Parameters

nIndex	The zero-based index of the item to remove.

See also

M:System.Collections.IList.RemoveAt(int)

6.9.3 Property Documentation

6.9.3.1 int Subsembly.SmartCard.CardDataObjectList.Count [get]

Gets the number of.

The count.

6.9.3.2 bool Subsembly.SmartCard.CardDataObjectList.IsFixedSize [get]

Gets a value indicating whether this object is fixed size.

True if this object is fixed size, false if not.

See also

P:System.Collections.IList.IsFixedSize

6.9.3.3 bool Subsembly.SmartCard.CardDataObjectList.IsReadOnly [get]

Gets a value indicating whether this object is read only.

True if this object is read only, false if not.

6.9.3.4 bool Subsembly.SmartCard.CardDataObjectList.IsSynchronized [get]

Gets a value indicating whether this object is synchronized.

True if this object is synchronized, false if not.

6.9.3.5 object Subsembly.SmartCard.CardDataObjectList.SyncRoot [get]

Gets the synchronize root.

The synchronize root.

6.9.3.6 object Subsembly.SmartCard.CardDataObjectList.this[int nIndex] [get], [set]

Indexer to get or set items within this collection using array index syntax.

Parameters

nIndex	The zero-based index of the element to get or set.

Returns

The indexed item.

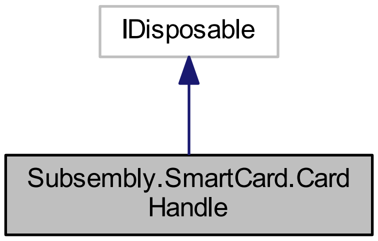
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardData↩ ObjectList.cs

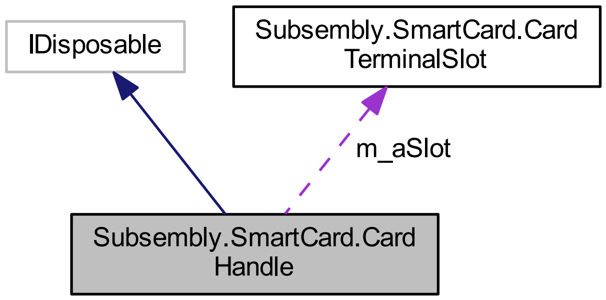
6.10 Subsembly.SmartCard.CardHandle Class Reference

Exclusive handle to a particular card that was introduced to the system by insertion into a card terminal slot. An application shall access smart cards only through instances of this or a derived class.

Inheritance diagram for Subsembly.SmartCard.CardHandle:



Collaboration diagram for Subsembly.SmartCard.CardHandle:


IDisposable
Subsembly.SmartCard.Card TerminalSlot
m_aSlot
Subsembly.SmartCard.Card Handle

Public Member Functions

- void Dispose ()

  Each instance of this class must be orderly disposed by invoking the IDispose.Dispose method when the handle to the card is no longer needed.
- void Disconnect ()

  Disconnects the card and releases the card terminal slot.
- byte[ ] GetATR ()

  Returns the ATR of the card that this CardHandle controls.
- byte[ ] GetHistoricalBytes ()

  Returns the Historical Bytes of the ATR of the card.
- bool Reset ()

  Fully reset the inserted card. This can be useful in order to recover from communication failures or card errors.
- CardResponseAPDU SendCommand (CardCommandAPDU aCommandAPDU)

  Sends the command APDU to the card represented by this card handle and returns its response APDU.
- CardResponseAPDU SelectApplication (byte[ ] vbApplicationID)

  Simplifies the selection of a card application.
- CardResponseAPDU SelectRoot ()

  Generic ISO 7816-4 command for selecting the master file (MF).
- CardResponseAPDU SelectFile (int nFileID)

  Generic ISO 7816-4 command for selecting an elementary file EF.
- CardResponseAPDU SelectDirectory (int nFileID)

  Generic ISO 7816-4 command for selecting a dedicated file DF.
- CardResponseAPDU ReadBinary (int nOffset, byte nLength)

  Generic ISO 7816-4 command for reading data from the currently selected elementary file with transparent structure.
- CardResponseAPDU ReadBinary (byte nSFI, byte nOffset, byte nLength)

  Generic ISO 7816-4 command for reading data from a elementary file (EF) with transparent structure. The EF will be implicitly selected through a short file identifier (SFI).
- CardResponseAPDU UpdateBinary (int nOffset, byte[ ] vbData)

  Generic ISO 7816-4 command for updating data in the currently selected elementary file with transparent structure.

- CardResponseAPDU WriteBinary (int nOffset, byte[ ] vbData)

  Generic ISO 7816-4 command for writing data to the currently selected elementary file with transparent structure.
- CardResponseAPDU ReadRecord (byte nRecNo, byte nLength)

  Generic ISO 7816-4 command for reading a record from the currently selected elementary file with record structure.
- CardResponseAPDU UpdateRecord (byte nRecNo, byte[ ] vbData)

  Generic ISO 7816-4 command for updating data in the currently selected elementary file with record structure.
- CardResponseAPDU VerifyPin (CardPinControl aPinControl, string sPin)

  Performs a PIN verification.
- CardResponseAPDU ChangePin (CardPinControl aPinControl, string sOldPin, string sNewPin)

  Performs PIN verify and PIN change.

Properties

- CardTerminalSlot Slot `[get]`

  The slot where the card was inserted.
- bool IsInvalid `[get]`

  Indicates whether the presented and connected card was unexpectedly removed by the user. If this flag is true, then it is no longer possible to send any commands to the card, even if a card is re-inserted into the card terminal slot.
- byte CLA `[get, set]`

  The class byte to be used when sending generic ISO 7816-4 command APDUs to the card.

6.10.1 Detailed Description

Exclusive handle to a particular card that was introduced to the system by insertion into a card terminal slot. An application shall access smart cards only through instances of this or a derived class.

An instance of the CardHandle class can only be created in response to the insertion of a smart card into a card terminal slot. Subsequently a card handle remains tied to the card terminal slot where the card was originally inserted. If that card is removed, the card handle becomes *invalid* (see IsInvalid) and remains invalid, even if the card is re-inserted again. To handle another card insertion, even if it is actually the same card, a fresh new instance of this class is required.

See also

T:System.IDisposable

6.10.2 Member Function Documentation

6.10.2.1 CardResponseAPDU Subsembly.SmartCard.CardHandle.ChangePin (CardPinControl *aPinControl*, string *sOldPin*, string *sNewPin*) `[inline]`

Performs PIN verify and PIN change.

Exceptions

ArgumentNullException	Any of the arguments was `null`.

Parameters

aPinControl	Control information about the command APDU to be sent to the card. As basically every card has its own subtle way of chaning the PIN, there is no easier way than this.
sOldPin	.
sNewPin	.

Returns

The response APDU from the card command is returned. Usually this contains nothing but the status word which indicates whether the PIN modification was successful.

6.10.2.2 void Subsembly.SmartCard.CardHandle.Dispose ( ) `[inline]`

Each instance of this class must be orderly disposed by invoking the IDispose.Dispose method when the handle to the card is no longer needed.

Disposing the card handle automatically disconects and powers down the card, if it is still present in the card reader. However, the card is not delivered.

See also

M:System.IDisposable.Dispose()

6.10.2.3 byte [ ] Subsembly.SmartCard.CardHandle.GetATR ( ) `[inline]`

Returns the ATR of the card that this CardHandle controls.

Returns

Byte array that contains the complete ATR of the card.

6.10.2.4 byte [ ] Subsembly.SmartCard.CardHandle.GetHistoricalBytes ( ) `[inline]`

Returns the Historical Bytes of the ATR of the card.

If the final check byte TCK is present, then it is verified against the computed check sum. If the check fails, then `null` is returned, too.

Returns

If no ATR is present in this CardHandle, or the ATR cannot be parsed, then `null` is returned. If an ATR is present and can be parsed according to ISO 7816-3, then the Historical Bytes are extracted from it and are returned. If the ATR is correct, but does not contain any Historical Bytes, then an empty byte array is returned. The max length of the returned byte array is 15.

6.10.2.5 CardResponseAPDU Subsembly.SmartCard.CardHandle.ReadBinary (int *nOffset*, byte *nLength*) `[inline]`

Generic ISO 7816-4 command for reading data from the currently selected elementary file with transparent structure.

This method uses the following generic command APDU: `CLA 'B0' HI LO LE`, where HI and LO are the high and low byte of the given offset and LE is the given length byte.

Exceptions

ArgumentOutOfRange↩ Exception	The given parameter nOffset was invalid.
CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nOffset	Byte offset where to begin reading. The offset must be in the range from 0 through hex '7FFF'.
nLength	Number of bytes to read. If this is zero, then all available bytes up to the end of the file are read.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the data was read successfully. The read data is available through the CardResponseAPDU.GetData method.

6.10.2.6 CardResponseAPDU Subsembly.SmartCard.CardHandle.ReadBinary (byte *nSFI,* byte *nOffset,* byte *nLength*) `[inline]`

Generic ISO 7816-4 command for reading data from a elementary file (EF) with transparent structure. The EF will be implicitly selected through a short file identifier (SFI).

This method uses the following generic command APDU: `CLA 'B0' '80'+SFI OF LE`, where SFI is the short file identifier, and OF is the given offset, and LE is the given length byte.

Exceptions

ArgumentOutOfRange↩ Exception	The given parameter nSFI is invalid.

Parameters

nSFI	The short file identifier (SFI) of the EF from which the binary data shall be read. A valid SFI must be in the range from zero through 31.
nOffset	Byte offset where to begin reading.
nLength	Number of bytes to read. If this is zero, then all available bytes up to the end of the file are read.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the data was read successfully. The read data is available through the CardResponseAPDU.GetData method.

6.10.2.7 CardResponseAPDU Subsembly.SmartCard.CardHandle.ReadRecord (byte *nRecNo,* byte *nLength*) `[inline]`

Generic ISO 7816-4 command for reading a record from the currently selected elementary file with record structure.

This method uses the following generic command APDU: `CLA 'B2' RN '04' LE`, where RN is record number byte and LE is the given length byte.

Note that this method does not work with ACOS1 cards which would need the byte value '00' in P2 instead of the correct ISO 7816-4 value '04' which is used here.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nRecNo	Absolute record number of record to be read. The first record usually has the record number 1.
nLength	Number of bytes to read from record. If this is zero, then the entire record is read.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the record was read successfully. The read record is available through the CardResponseAPDU.GetData method.

6.10.2.8 bool Subsembly.SmartCard.CardHandle.Reset ( ) `[inline]`

Fully reset the inserted card. This can be useful in order to recover from communication failures or card errors.

Exceptions

CardTerminalException	

If the card handle is invalid, because the original card was removed from the card terminal, then a CardTerminal↩ Exception with code CardTerminalExceptionCode.CardWithdrawn will be thrown.

Returns

If the card reset was successful, then true is returned. If the attempt to reset the card failed, then false is returned.

6.10.2.9 CardResponseAPDU Subsembly.SmartCard.CardHandle.SelectApplication (byte[ ] *vbApplicationID*) `[inline]`

Simplifies the selection of a card application.

An attempt is made to select the card application with the given application ID (AID). The Selection is attempted using the most simple ISO 7816-4 SELECT command: 'CLA A4 04 0C'.

The class byte for the SELECT command will be taked from the CLA property.

Exceptions

ArgumentNullException	The parameter vbApplicationID was `null`.
ArgumentException	The vbApplicationID byte array was empty or longer than 256 bytes.
CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

vbApplicationID	The application ID of the application to be selected.

Returns

In any case, the response APDU of the selection command will be returned to the caller.

6.10.2.10 CardResponseAPDU Subsembly.SmartCard.CardHandle.SelectDirectory (int *nFileID*) [inline]

Generic ISO 7816-4 command for selecting a dedicated file DF.

The file to be selected must be a dedicated file (DF).

This method uses the following generic command APDU: `CLA 'A4' '01' '0C' FILEID`, where FILEID is the given two byte file ID.

Exceptions

ArgumentOutOfRange↩ Exception	The given nFileID was outside the permitted range.
CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nFileID	The file ID (FID) of the DF to be selected. The file ID must be in the range from hex '0000' through hex 'FFFF'.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the file was selected successfully.

6.10.2.11 CardResponseAPDU Subsembly.SmartCard.CardHandle.SelectFile (int *nFileID*) [inline]

Generic ISO 7816-4 command for selecting an elementary file EF.

The file to be selected must be an elementary file (EF).

This method uses the following generic command APDU: `CLA 'A4' '02' '0C' FILEID`, where FILEID is the given two byte file ID.

Exceptions

ArgumentOutOfRange↩ Exception	The given nFileID was outside the permitted range.
CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nFileID	The file ID (FID) of the EF to be selected. The file ID must be in the range from hex '0000' through hex 'FFFF'.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the file was selected successfully.

6.10.2.12 CardResponseAPDU Subsembly.SmartCard.CardHandle.SelectRoot ( ) [inline]

Generic ISO 7816-4 command for selecting the master file (MF).

This method issues the following generic command APDU: `CLA 'A4' '00' '0C' '02' '3F 00'`.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the file was selected successfully.

6.10.2.13 **CardResponseAPDU** Subsembly.SmartCard.CardHandle.SendCommand (**CardCommandAPDU** *aCommandAPDU*) `[inline]`

Sends the command APDU to the card represented by this card handle and returns its response APDU.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
CardTerminalException	

If the card handle is invalid, because the original card was removed from the card terminal, then a CardTerminal↩ Exception with code CardTerminalExceptionCode.CardWithdrawn will be thrown.

Other CardTerminalExceptionCodes may be thrown as a consequence of sending the command to the card.

Parameters

aCommandAP↩ DU	.

Returns

A CardResponseAPDU.

6.10.2.14 **CardResponseAPDU** Subsembly.SmartCard.CardHandle.UpdateBinary (int *nOffset,* byte[ ] *vbData*) `[inline]`

Generic ISO 7816-4 command for updating data in the currently selected elementary file with transparent structure.

This method uses the following generic command APDU: `CLA 'D6' HI LO LC DATA`, where HI and LO are the high and low byte of the given offset, LC is the length of the given data and DATA is the given data array itself.

With some readers, this method can also be used to update data on a synchronous memory card. The UPDATE BIN↩ ARY APDU sent by this method conforms to the MCT specification part 7 (IC Cards with Synchronouse Transmission, Usage of Interindustry Commands) which can be downloaded from `www.teletrust.de`

Exceptions

ArgumentOutOfRange↩ Exception	The parameter nOffset was outside the permitted range.
ArgumentNullException	The parameter vbData was `null`.
ArgumentException	The parameter vbData was invalid.
CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nOffset	Byte offset where to begin updating data. The offset must be in the range from 0 through hex '7FFF'.
vbData	Data that shall be written to the file at the given offset. Must not be `null` and must not have a zero length or be longer than 256 bytes.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the update was successful.

6.10.2.15 CardResponseAPDU Subsembly.SmartCard.CardHandle.UpdateRecord (byte *nRecNo*, byte[ ] *vbData*) `[inline]`

Generic ISO 7816-4 command for updating data in the currently selected elementary file with record structure.

This method uses the following generic command APDU: `CLA 'DC' RN '04' LC DATA`, where RN is record number byte, LC is the length of the given data and DATA is the given data array itself.

Exceptions

ArgumentNullException	The parameter vbData was `null`.
ArgumentOutOfRange←Exception	Thrown when one or more arguments are outside the required range.
CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nRecNo	Absolute record number of record to be updated. The first record usually has the record number 1.
vbData	Data that shall be written to the record. Must not be `null` and must not have a zero length or be longer than 256 bytes. Usually the entire record must be updated, thus the given data must match the size of the record structure on the smart card.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the update was successful.

6.10.2.16 CardResponseAPDU Subsembly.SmartCard.CardHandle.VerifyPin (CardPinControl *aPinControl*, string *sPin*) `[inline]`

Performs a PIN verification.

Exceptions

ArgumentNullException	Any of the arguments was `null`.
CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

aPinControl	Control information about how to format the PIN and create a verify PIN command APDU.
sPin	PIN to be inserted into the verify PIN command APDI that is built and sent.

Returns

The response APDU from the verify card command is returned. Usually this contains nothing but the status word which indicates whether the PIN verification was successful.
If the given PIN is completely malformed and no card command could be built, then `null` is returned.

6.10.2.17 CardResponseAPDU Subsembly.SmartCard.CardHandle.WriteBinary (int *nOffset*, byte[ ] *vbData*) `[inline]`

Generic ISO 7816-4 command for writing data to the currently selected elementary file with transparent structure.

This method uses the following generic command APDU: `CLA 'D0' HI LO LC DATA`, where HI and LO are the high and low byte of the given offset, LC is the length of the given data and DATA is the given data array itself.

With some readers, this method can also be used to write data to a synchronous memory card. The WRITE BINARY APDU sent by this method conforms to the Kobil B1 smart card reader which is emulated by readers that claim to be B1 compatible.

Exceptions

ArgumentOutOfRange←↩ Exception	The parameter nOffset was outside the permitted range.
ArgumentNullException	The parameter vbData was `null`.
ArgumentException	The parameter vbData was invalid.
CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nOffset	Byte offset where to begin writing data. The offset must be in the range from 0 through hex '7FFF'.
vbData	Data that shall be written to the file at the given offset. Must not be `null` and must not have a zero length or be longer than 256 bytes.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the update was successful.

6.10.3 Property Documentation

6.10.3.1 byte Subsembly.SmartCard.CardHandle.CLA `[get], [set]`

The class byte to be used when sending generic ISO 7816-4 command APDUs to the card.

Depending on the smart card operating system and whether secure messaging or logical channels are used, a different class byte value may be required. For example, GSM SIM cards always use a class byte of 'A0', ACOS1 cards always use a class byte of '80'.

The default class byte value is '00'.

6.10.3.2 bool Subsembly.SmartCard.CardHandle.IsInvalid `[get]`

Indicates whether the presented and connected card was unexpectedly removed by the user. If this flag is true, then it is no longer possible to send any commands to the card, even if a card is re-inserted into the card terminal slot.

This flag remains set, even if the user re-inserts the very same card again. In order to connect to a re-inserted card an application must obtain a fresh new instance of the CardHandle class.

True if this object is invalid, false if not.

6.10.3.3 CardTerminalSlot Subsembly.SmartCard.CardHandle.Slot `[get]`

The slot where the card was inserted.

The value is `null` when the card was withdrawn and thus IsInvalid is `true`.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardHandle.cs

6.11 Subsembly.SmartCard.CardHex Class Reference

Helper class to make life with card hex data easier.

Static Public Member Functions

- static bool Compare (byte[ ] array1, byte[ ] array2, int comparisonLength)

  Compares objects.
- static bool Compare (byte[ ] array1, int offset1, byte[ ] array2, int offset2, int comparisonLength)

  Compares objects.
- static char ToDigit (int nDigit)

  Converts an int representing a nibble to a hex digit character to a given value.
- static int FromByteArray (byte[ ] data, int byteOffset, int bitOffset, int numberOfBits)

  Interprets a byte array as MSB first (leftmost bit) and then allows extraction, of a int (32 bit) value by byte offset, bit offset and number of bits COUNTING BITS FROM LSB to MSB i.e. reight to left.
- static long LongFromByteArray (byte[ ] data, int byteOffset, int bitOffset, int numberOfBits)

  Interprets a byte array as MSB first (leftmost bit) and allows extraction, of a long (64 bit) value by byte offset, bit offset and number of bits COUNTING BITS FROM LSB to MSB i.e. right to left.
- static int FromDigit (char ch)

  Returns the numerical value of the given hex digit character.
- static bool IsDigit (char ch)

  Tests whether the given character is a valid hex digit.
- static bool IsEqual (byte[ ] a1, byte[ ] a2)

  Tests whether two given byte arrays are binary identical. Returns false if either parameter is null.
- static bool IsEqual (byte[ ] a1, int offset1, byte[ ] a2, int offset2, int length)

  Tests whether two sections of byte arrays are binary identical.
- static bool IsHex (string s)

  Checks whether the given string is a valid hex number.
- static byte[ ] ToByteArray (string sHex)

  Converts a string of hex digits into a byte array.

- static void ToByteArray (string sHex, byte[ ] vb, int nOffset)

  Converts a string of hex digits into bytes into a byte array.
- static string FromByteArray (byte[ ] vb, int nOffset, int nLength)

  Converts a range from the given byte array into a string of hex digits.
- static string FromByteArray (byte[ ] vb)

  Converts the entire given byte array into a string of hex digits.
- static string FromByte (byte b)

  Converts a given byte into a string of two hexdigits.
- static string FromWord (int n)

  Converts a given 16 bit word into a string of four hex digits. The most significant nibble will be encoded as the first character, the least significant nibble will be encoded as the fourth character.
- static string FromDword (int n)

  Converts a given 32 bit DWORD into a string of eight hex digits. The most significant nibble will be encoded as the first character, the least significant nibble will be encoded as the eighth character.
- static byte ReverseBitOrder (byte inputByte)

  Reverses the bit order within a byte.
- static void ShiftBits (ref byte[ ] vbData, int numberOfBitsShifted)

  Shifts a byte array by numberOfBitsShifted. Currently only rightShifts are supported.
- static byte[ ] ShiftBit (byte[ ] parameter)

  Rolls the array by one bit.
- static string DataToTrace (byte[ ] vbData)

  Converts binary data into a string representation based on the Windows 1252 encoding.
- static string DataToTrace (byte[ ] vbData, int nOffset, int nLength)

  Converts binary data into a string representation based on the Windows 1252 encoding.
- static byte[ ] XORN (byte[ ] inData, byte[ ] xorData)

  XORes an N-byte block.
- static byte[ ] AND (byte[ ] inData, byte[ ] andData)

  ANDs two equal-size byte arrays.
- static byte[ ] BitWiseAndForArray (byte[ ] arrayA, byte[ ] arrayB)

  Calculate bitwise AND for two arrays.
- static bool BitWiseAndForArrayIsNotZero (byte[ ] arrayA, byte[ ] arrayB)

  Retuns true if bitwise AND of two arrays is not equal to 0.

6.11.1 Detailed Description

Helper class to make life with card hex data easier.

6.11.2 Member Function Documentation

6.11.2.1 static byte [ ] Subsembly.SmartCard.CardHex.AND (byte[ ] *inData,* byte[ ] *andData*) `[inline]`, `[static]`

ANDs two equal-size byte arrays.

Exceptions

ArgumentException	Thrown when one or more arguments have unsupported or illegal values.

Parameters

inData	Information describing the in.
andData	Information describing the and.

Returns

A byte[].

6.11.2.2 static byte [ ] Subsembly.SmartCard.CardHex.BitWiseAndForArray (byte[ ] *arrayA,* byte[ ] *arrayB*) `[inline]`, `[static]`

Calculate bitwise AND for two arrays.

Parameters

arrayA	Array of bytes.
arrayB	Array of bytes.

Returns

Array of bytes with result of AND operation on appropriate bytes of both arrays.

6.11.2.3 static bool Subsembly.SmartCard.CardHex.BitWiseAndForArrayIsNotZero (byte[ ] *arrayA,* byte[ ] *arrayB*) `[inline]`, `[static]`

Retuns true if bitwise AND of two arrays is not equal to 0.

Parameters

arrayA	Array of bytes.
arrayB	Array of bytes.

Returns

True if it succeeds, false if it fails.

6.11.2.4 static bool Subsembly.SmartCard.CardHex.Compare (byte[ ] *array1,* byte[ ] *array2,* int *comparisonLength*) `[inline]`, `[static]`

Compares objects.

Parameters

array1	byte array to compare
array2	byte array to compare
comparison↵ Length	length of comparison

Returns

True if it succeeds, false if it fails.

6.11.2.5 static bool Subsembly.SmartCard.CardHex.Compare (byte[ ] *array1*, int *offset1*, byte[ ] *array2*, int *offset2*, int *comparisonLength*) `[inline]`, `[static]`

Compares objects.

Parameters

array1	byte array to compare
offset1	offset in array 1
array2	byte array to compare
offset2	offset in array 1
comparison←Length	length of comparison

Returns

True if it succeeds, false if it fails.

6.11.2.6 static string Subsembly.SmartCard.CardHex.DataToTrace (byte[ ] *vbData*) `[inline]`, `[static]`

Converts binary data into a string representation based on the Windows 1252 encoding.

Parameters

vbData	.

Returns

Returns a string based on the Windows 1252 encoding where all unprintable characters are replaced by dots.

6.11.2.7 static string Subsembly.SmartCard.CardHex.DataToTrace (byte[ ] *vbData*, int *nOffset*, int *nLength*) `[inline]`, `[static]`

Converts binary data into a string representation based on the Windows 1252 encoding.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange←Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	.
nOffset	.
nLength	.

Returns

Returns a string based on the Windows 1252 encoding where all unprintable characters are replaced by dots.

6.11.2.8 static string Subsembly.SmartCard.CardHex.FromByte (byte *b*) [inline], [static]

Converts a given byte into a string of two hexdigits.

Parameters

b	The byte to be converted.

Returns

String that contains two hex digit characters.

6.11.2.9 static int Subsembly.SmartCard.CardHex.FromByteArray (byte[ ] *data,* int *byteOffset,* int *bitOffset,* int *numberOfBits*) [inline], [static]

Interprets a byte array as MSB first (leftmost bit) and then allows extraction, of a int (32 bit) value by byte offset, bit offset and number of bits COUNTING BITS FROM LSB to MSB i.e. reight to left.

Parameters

data	Byte array to extract bit sequence from.
byteOffset	Byte offset, starting on the LSB side i.e. right side of the byte array.
bitOffset	Bit offset, starting on right side of the byte array plus byteOffset if applicable.
numberOfBits	The number of bits that need to be extracted from the byte array.

Returns

Retuns a 32-bit int value.

6.11.2.10 static string Subsembly.SmartCard.CardHex.FromByteArray (byte[ ] *vb,* int *nOffset,* int *nLength*) [inline], [static]

Converts a range from the given byte array into a string of hex digits.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↵ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vb	This is the byte array that contains the byte sequence to be converted. This must not be null, otherwise an ArgumentNullException will be thrown.
nOffset	This is the offset of the first byte in the byte array that shall be considered.
nLength	Number of bytes to be converted. This must be positive or zero. If it is negative, then an ArgumentException will be thrown.

Returns

A string with the hexadecimal representation of the given byte sequence. The returned representation will use upper case hex digits.

6.11.2.11 static string Subsembly.SmartCard.CardHex.FromByteArray (byte[ ] *vb*) `[inline]`, `[static]`

Converts the entire given byte array into a string of hex digits.

Parameters

vb	This is the byte array that contains the byte sequence to be converted. This must not be null, otherwise an ArgumentNullException will be thrown.

Returns

A string with the hexadecimal representation of the given byte sequence. The returned representation will use upper case hex digits.

6.11.2.12 static int Subsembly.SmartCard.CardHex.FromDigit (char *ch*) `[inline]`, `[static]`

Returns the numerical value of the given hex digit character.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

ch	CardHex digit in any of the ranges '0' through '9', 'a' through 'f', or 'A' through 'F'. If any other character is given, then an ArgumentOutOfRangeException will be thrown.

Returns

Returns the numerical value of the given hex digit in the range from 0 through 15.

6.11.2.13 static string Subsembly.SmartCard.CardHex.FromDword (int *n*) `[inline]`, `[static]`

Converts a given 32 bit DWORD into a string of eight hex digits. The most significant nibble will be encoded as the first character, the least significant nibble will be encoded as the eighth character.

Parameters

n	Integer value to be converted. This may be in the range from 0x80000000 through 0xFFFFFFFF.

Returns

String that contains eight hex digit characters.

6.11.2.14 static string Subsembly.SmartCard.CardHex.FromWord (int *n*) `[inline]`, `[static]`

Converts a given 16 bit word into a string of four hex digits. The most significant nibble will be encoded as the first character, the least significant nibble will be encoded as the fourth character.

Parameters

n	Integer value to be converted. This may be in the range from -32768 (which will be encoded as '8000' through 65535 (which will be encoded as 'FFFF').

Returns

String that contains four hex digit characters.

6.11.2.15 static bool Subsembly.SmartCard.CardHex.IsDigit (char *ch*) `[inline]`, `[static]`

Tests whether the given character is a valid hex digit.

Parameters

ch	Character to test.

Returns

If the given character is in any of the rangeS '0' through '9', 'a' through 'f', or 'A' through 'F', then `true` is returned. Otherwise `false` is returned.

6.11.2.16 static bool Subsembly.SmartCard.CardHex.IsEqual (byte[ ] *a1*, byte[ ] *a2*) `[inline]`, `[static]`

Tests whether two given byte arrays are binary identical. Returns false if either parameter is null.

Parameters

a1	The first byte[].
a2	The second byte[].

Returns

True if equal, false if not.

6.11.2.17 static bool Subsembly.SmartCard.CardHex.IsEqual (byte[ ] *a1*, int *offset1*, byte[ ] *a2*, int *offset2*, int *length*) `[inline]`, `[static]`

Tests whether two sections of byte arrays are binary identical.

Parameters

a1	The first byte[].
offset1	.
a2	The second byte[].
offset2	.
length	The length.

Returns

True if equal, false if not.

6.11.2.18 static bool Subsembly.SmartCard.CardHex.IsHex (string *s*) [inline], [static]

Checks whether the given string is a valid hex number.

Parameters

s	The string to check.

Returns

If the string is a valid hex number, then `true` is returned. Otherwise `false` is returned.

6.11.2.19 static long Subsembly.SmartCard.CardHex.LongFromByteArray (byte[ ] *data*, int *byteOffset*, int *bitOffset*, int *numberOfBits*) [inline], [static]

Interprets a byte array as MSB first (leftmost bit) and allows extraction, of a long (64 bit) value by byte offset, bit offset and number of bits COUNTING BITS FROM LSB to MSB i.e. right to left.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentException	Thrown when one or more arguments have unsupported or illegal values.
Exception	Thrown when an exception error condition occurs.

Parameters

data	Byte array to extract bit sequence from.
byteOffset	Byte offset, starting on the LSB side i.e. right side of the byte array.
bitOffset	Bit offset, starting on right side of the byte array plus byteOffset if applicable.
numberOfBits	The number of bits that need to be extracted from the byte array.

Returns

Retuns a 64-bit long value.

6.11.2.20 static byte Subsembly.SmartCard.CardHex.ReverseBitOrder (byte *inputByte*) [inline], [static]

Reverses the bit order within a byte.

Parameters

inputByte	inputByte.

Returns

byte.

6.11.2.21 static byte [ ] Subsembly.SmartCard.CardHex.ShiftBit (byte[ ] *parameter*) [inline], [static]

Rolls the array by one bit.

Parameters

parameter	Byte array to be rolled.

Returns

A byte[].

6.11.2.22 static void Subsembly.SmartCard.CardHex.ShiftBits (ref byte[ ] *vbData,* int *numberOfBitsShifted*) [inline], [static]

Shifts a byte array by numberOfBitsShifted. Currently only rightShifts are supported.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	[in,out].
numberOfBits↩ Shifted	.

6.11.2.23 static byte [ ] Subsembly.SmartCard.CardHex.ToByteArray (string *sHex*) [inline], [static]

Converts a string of hex digits into a byte array.

Exceptions

ArgumentException	Thrown when one or more arguments have unsupported or illegal values.

Parameters

sHex	String of hex digits. This string must not be null and must have an even length. If it is null, then an ArgumentNullException will be thrown. The hex digits may be upper or lower case. If it is not of even length or contains characters that are not valid hex digits, then an ArgumentException will be thrown.

Returns

Returns a byte array with the byte values of every pair of hex digits in the input string. If the given input string is empty, then the returned byte array is an empty array, too. Generally the size of the returned byte array is half the size of the input string.

6.11.2.24 static void Subsembly.SmartCard.CardHex.ToByteArray (string *sHex*, byte[ ] *vb*, int *nOffset*) [inline], [static]

Converts a string of hex digits into bytes into a byte array.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentException	Thrown when one or more arguments have unsupported or illegal values.

Parameters

sHex	String of hex digits. This string must not be null and must have an even length. If it is null, then an ArgumentNullException will be thrown. The hex digits may be upper or lower case. If it is not of even length or contains characters that are not valid hex digits, then an ArgumentException will be thrown.
vb	The destination byte array where the individual bytes of the given string shall be stored.
nOffset	Offset into the byte array where the first character shall be stored.

6.11.2.25 static char Subsembly.SmartCard.CardHex.ToDigit (int *nDigit*) [inline], [static]

Converts an int representing a nibble to a hex digit character to a given value.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nDigit	Digit value in the range from 0 through 15.

Returns

Returns character in the range '0' through '9' or 'A' through 'F'.

6.11.2.26 static byte [ ] Subsembly.SmartCard.CardHex.XORN (byte[ ] *inData*, byte[ ] *xorData*) [inline], [static]

XORes an N-byte block.

Exceptions

ArgumentException	Thrown when one or more arguments have unsupported or illegal values.

Parameters

inData	Information describing the in.
xorData	Information describing the exclusive or.

Returns

A byte[].

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardHex.cs

6.12 Subsembly.SmartCard.CardPcScConfig Class Reference

Collects information about a PC/SC card terminal configuration. This information usually is stored in the CardTerminal↩ Registry.

Public Member Functions

- CardPcScConfig ()

  Construct CardPcScConfig with all default values.
- CardPcScConfig (string sReaderName)

  Constructor.
- void Load (string sXmlConfig)

  Loads the PC/SC configuration XML string.
- string Save ()

  Gets the save.
- override bool Equals (object rhs)

  Tests if this object is considered equal to another.
- override int GetHashCode ()

  Calculates a hash code for this object.

Properties

- string ReaderName `[get, set]`

  Name of the card reader as it is known by the PC/SC Workgroup API.
- bool ReadProperties `[get, set]`

  Quick fix to control access to Reader properties only available via direct access to the smart card reader system. This is to prevent interference with smart card services and minidrivers. Do not change this value unless you know what you are doing.

6.12.1 Detailed Description

Collects information about a PC/SC card terminal configuration. This information usually is stored in the CardTerminal↩ Registry.

Currently the only configuration information required and stored is the card reader name that identifies the card reader to the PC/SC Workgroup API.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Subsembly.SmartCard.CardPcScConfig.CardPcScConfig (string *sReaderName*) [inline]

Constructor.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

sReaderName	Name of the reader.

6.12.3 Member Function Documentation

6.12.3.1 override bool Subsembly.SmartCard.CardPcScConfig.Equals (object *rhs*) [inline]

Tests if this object is considered equal to another.

Parameters

rhs	The object to compare with the current object.

Returns

True if the objects are considered equal, false if they are not.

See also

M:System.Object.Equals(object)

6.12.3.2 override int Subsembly.SmartCard.CardPcScConfig.GetHashCode ( ) [inline]

Calculates a hash code for this object.

Returns

A hash code for this object.

See also

M:System.Object.GetHashCode()

6.12.3.3 void Subsembly.SmartCard.CardPcScConfig.Load (string *sXmlConfig*) [inline]

Loads the PC/SC configuration XML string.

Parameters

sXmlConfig	The XML Configuration to load.

6.12.3.4 string Subsembly.SmartCard.CardPcScConfig.Save ( ) `[inline]`

Gets the save.

Returns

A string.

6.12.4 Property Documentation

6.12.4.1 string Subsembly.SmartCard.CardPcScConfig.ReaderName `[get]`, `[set]`

Name of the card reader as it is known by the PC/SC Workgroup API.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange←Exception	Thrown when one or more arguments are outside the required range.

The name of the reader.

6.12.4.2 bool Subsembly.SmartCard.CardPcScConfig.ReadProperties `[get]`, `[set]`

Quick fix to control access to Reader properties only available via direct access to the smart card reader system. This is to prevent interference with smart card services and minidrivers. Do not change this value unless you know what you are doing.

True if read properties, false if not.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardPcSc←Config.cs

6.13 Subsembly.SmartCard.CardPcscPart3 Class Reference

Provides ATR parsing capabilities for PCSC 2.01 Part 3 compliant card systems.

Public Member Functions

- CardPcscPart3 (byte[ ] atr)
  Parses ATR to extract info about the card standard and card type.

Properties

- bool isValid `[get]`

Returns true if the atr is a valid PCSC Part3 compliant ATR.

- bool isContactless `[get]`

  Returns true unless we can tell that there is a contact card standard involved (based on PC/SC Part 3 ATR analysis)
- int CardIdentifier `[get]`

  PC/SC part 3 compliant card identifier; 0xFFFF if the card identifier is undefined.
- int CardStandard `[get]`

  PC/SC part 3 compliant card standard.
- string CardName `[get]`

  Card name derived from PC/SC part 3 card identifier NN.
- string CardProtocol `[get]`

  Card protocol derived from PC/SC part 3 standard 'SS'.

6.13.1 Detailed Description

Provides ATR parsing capabilities for PCSC 2.01 Part 3 compliant card systems.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 Subsembly.SmartCard.CardPcscPart3.CardPcscPart3 (byte[ ] *atr*) `[inline]`

Parses ATR to extract info about the card standard and card type.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

atr	20-byte ATR.

6.13.3 Property Documentation

6.13.3.1 int Subsembly.SmartCard.CardPcscPart3.CardIdentifier `[get]`

PC/SC part 3 compliant card identifier; 0xFFFF if the card identifier is undefined.

The identifier of the card.

6.13.3.2 string Subsembly.SmartCard.CardPcscPart3.CardName `[get]`

Card name derived from PC/SC part 3 card identifier NN.

The name of the card.

6.13.3.3 string Subsembly.SmartCard.CardPcscPart3.CardProtocol `[get]`

Card protocol derived from PC/SC part 3 standard 'SS'.

The card protocol.

6.13.3.4 int Subsembly.SmartCard.CardPcscPart3.CardStandard [get]

PC/SC part 3 compliant card standard.

The card standard.

6.13.3.5 bool Subsembly.SmartCard.CardPcscPart3.isContactless [get]

Returns true unless we can tell that there is a contact card standard involved (based on PC/SC Part 3 ATR analysis)

True if this object is contactless, false if not.

6.13.3.6 bool Subsembly.SmartCard.CardPcscPart3.isValid [get]

Returns true if the atr is a valid PCSC Part3 compliant ATR.

True if this object is valid, false if not.

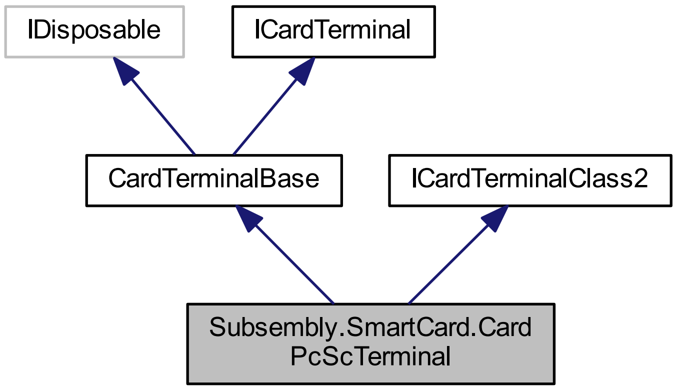
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardPcsc↩ Part3.cs

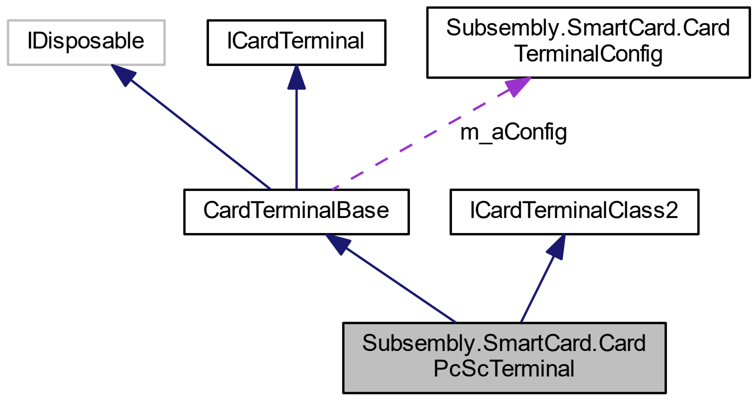
6.14 Subsembly.SmartCard.CardPcScTerminal Class Reference

Implementation of the high-level ICardTerminal abstraction based on the PC/SC Workgroup API.;

Inheritance diagram for Subsembly.SmartCard.CardPcScTerminal:



Collaboration diagram for Subsembly.SmartCard.CardPcScTerminal:


IDisposable
ICardTerminal
Subsembly.SmartCard.Card TerminalConfig
m_aConfig
CardTerminalBase
ICardTerminalClass2
Subsembly.SmartCard.Card PcScTerminal

Public Member Functions

- CardPcScTerminal ()

  Default constructor that does not implicitly initialize this CardPcScTerminal instance for a particular card reader.
- CardPcScTerminal (SCardResourceManager aCardResourceManager, bool fOwnResourceManager, string s←ReaderName)

  Creates a CardPcScTerminal instance for a given reader, using the given PC/SC resource manager.
- CardPcScTerminal (string sReaderName)

  Creates a CardPcScTerminal instance for a card terminal where the PC/SC Workgroup API reader name is given.
- void ReadProperties ()

  Determine the properties of the card terminal by establishing a direct connection. Note that this will fail if any other application already accesses the same reader. We handle this in a benign way, by just ignoring the failed direct connection as it might be prevented by a shared violation due to a card already being powered up during instantiation of the card terminal class.

Protected Member Functions

- override void InternalInit (string sXmlConfig)

  Initializes a CardPcScTerminal instance that was constructed with the default constructor.
- override void Dispose (bool fDisposing)

  Orderly disposes this CardPcScTerminal instance.
- override void InternalResetCardTerminal ()

  Entirely resets the card terminal, disconnecting (powering down) any existing card connection.
- override bool InternalIsCardPresent (int nSlot, ref bool requiresRemoval)

  Quick poll that indicates whether a card is inserted in a card terminal slot. This method also takes care of a connection refresh in case a pending card transaction is detected. This is to prevent PC/SC level transaction timeouts introduced with Windows 8. We are doing this here, because main polling thread is calling this method indirectly whenever it asks for a reader status update. Any PC/SC level errors must be caught here to prevent unhandled exceptions. Unless we detect removed cards, error codes are mapped to a card terminal DriverException which will then trigger a lost terminal event.

- override CardActivationResult InternalConnectCard (int nSlot, out byte[ ] vbATR)

  Attempts to connect to the card.
- override CardActivationResult InternalReconnectCard (int nSlot, out byte[ ] vbATR)

  Attempts to re-connect to the card.
- override CardTransactionResult InternalBeginCardTransaction ()

  Begins a card transaction. 10JUN2014: we set m_isPendingTransaction. This flag can be used to trigger the refresh mechanism necessary on Windows systems and get around 5 second transaction limit.
- override CardTransactionResult InternalEndCardTransaction ()

  Ends a card transaction.
- override CardResponseAPDU InternalSendCommand (int nSlot, CardCommandAPDU aCommandAPDU)

  Abstract method that must be overriden by a concrete card terminal implementation class to sends a given command APDU to the card reader.
- override void InternalDisconnectCard (int nSlot)

  Disconnects from the previously connected card.
- override void InternalDeliverCard (int nSlot, bool fCapture)

  Delivering an inserted card is currently not supported. This method therefore does nothing.

Properties

- SCardResourceManager CardResourceManager `[get]`

  Reference to SCardResourceManager that is used by this class. Don't use this reference unless absolutely necessary.
- SCardComm CardComm `[get]`

  Reference to SCardComm that is used by this class. Don't use this reference unless absolutely necessary.
- SCardAccessMode AccessMode `[get, set]`

  Selects the PC/SC access mode that will be used by the card connection. The default access mode is SHARED unless set otherwise in the registry. It can be changed to EXCLUSIVE by the HKLM key. "1" or no key means SHARED any other value translates to EXCLUSIVE.
- bool IsClass2 `[get]`

  Gets a value indicating whether this object is class 2.

Additional Inherited Members

6.14.1 Detailed Description

Implementation of the high-level ICardTerminal abstraction based on the PC/SC Workgroup API.;

An application that wants to use the convenient high level abstraction of the ICardTerminal interface with a PC/SC Workgroup API compatible reader shall do so by instantiating an instance of this class and then use the services of the ICardTerminal interface. After instantiation, only the ICardTerminal interface should be used and not the class methods directly.

See also

T:Subsembly.SmartCard.CardTerminalBase, T:Subsembly.SmartCard.ICardTerminalClass2

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Subsembly.SmartCard.CardPcScTerminal.CardPcScTerminal (SCardResourceManager *aCardResourceManager,* bool *fOwnResourceManager,* string *sReaderName*) [inline]

Creates a CardPcScTerminal instance for a given reader, using the given PC/SC resource manager.

Use this constructor if you already have an instance of the PC/SC resource manager.

Parameters

aCard↩ Resource↩ Manager	Reference of card resource manager to be used by this new instance. The ownership of this card resource manager may be passed or stay within the calling application, depending on the fOwnResourceManager parameter.
fOwnResource↩ Manager	Indicates whether the ownership of the SCardResourceManager instance shall remain with the calling application (fOwnResourceManager is false) or whether the new CardPcScTerminal instance shall take over ownership of the SCardResourceManager. If the CardPcScTerminal takes over ownership, then it will orderly dispose the SCardResourceManager when it is disposed, too. Otherwise it remains the task of the calling application to finally dispose the SCardResource↩ Manager.
sReaderName	PC/SC Workgroup API reader name of the card reader to use with this class. This may have been selected from the list of card reader names from the SCardResourceManager class.

6.14.2.2 Subsembly.SmartCard.CardPcScTerminal.CardPcScTerminal (string *sReaderName*) [inline]

Creates a CardPcScTerminal instance for a card terminal where the PC/SC Workgroup API reader name is given.

Parameters

sReaderName	PC/SC Workgroup API reader name of the card reader to use with this class.

6.14.3 Member Function Documentation

6.14.3.1 override void Subsembly.SmartCard.CardPcScTerminal.Dispose (bool *fDisposing*) [inline], [protected], [virtual]

Orderly disposes this CardPcScTerminal instance.

This is invoked from the base class CardTerminalBase.Dispose() or finalizer. If this class is derived further, then it shall override this method, too, and finally invoked this Dispose method.

Parameters

fDisposing	If true, then this method is invoked due to an invocation of the public IDispose.Dispose method. If false, then this method is invoked from the destructor (finalizer).

See also

M:Subsembly.SmartCard.CardTerminalBase.Dispose(bool)

Reimplemented from Subsembly.SmartCard.CardTerminalBase.

6.14.3.2 override CardTransactionResult Subsembly.SmartCard.CardPcScTerminal.InternalBeginCardTransaction ( ) [inline], [protected], [virtual]

Begins a card transaction. 10JUN2014: we set m_isPendingTransaction. This flag can be used to trigger the refresh mechanism necessary on Windows systems and get around 5 second transaction limit.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Returns

card transaction result.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalBeginCardTransaction()

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.3 override CardActivationResult Subsembly.SmartCard.CardPcScTerminal.InternalConnectCard (int *nSlot*, out byte[ ] *vbATR*) [inline], [protected], [virtual]

Attempts to connect to the card.

This method implements its own protocol selection strategy which prefers T=1 over T=0. Thus it always tries a T=1 connection first. Only if this attempt fails, then a T=0 connection is tried.

Parameters

nSlot	Slot number of card terminal slot of interest.
vbATR	[out] Returns the complete ATR that was received when the card was successfully powered up. Will be null whenever the card was not successfully powered up.

Returns

Returns the result from the attempt to activate the card. See CardActivationResult for possible return values.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalConnectCard(int,out byte[ ])

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.4 override void Subsembly.SmartCard.CardPcScTerminal.InternalDeliverCard (int *nSlot*, bool *fCapture*) [inline], [protected], [virtual]

Delivering an inserted card is currently not supported. This method therefore does nothing.

Parameters

nSlot	Slot number of card terminal slot of interest.
fCapture	Capture or eject.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalDeliverCard(int,bool)

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.5 override void Subsembly.SmartCard.CardPcScTerminal.InternalDisconnectCard (int *nSlot*) [inline], [protected], [virtual]

Disconnects from the previously connected card.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nSlot	Slot number of card terminal slot of interest.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalDisconnectCard(int)

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.6 override CardTransactionResult Subsembly.SmartCard.CardPcScTerminal.InternalEndCardTransaction ( ) [inline], [protected], [virtual]

Ends a card transaction.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Returns

card transaction result.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalEndCardTransaction()

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.7 override void Subsembly.SmartCard.CardPcScTerminal.InternalInit (string *sXmlConfig*) [inline], [protected], [virtual]

Initializes a CardPcScTerminal instance that was constructed with the default constructor.

Parameters

sXmlConfig	

Provides the specific configuration information.

String that contains XML data with configuration information for initialising this instance of the card terminal. The XML data contains a $<$ Config$>$ root node with card terminal specific child nodes.

If there is no configuration information, then this may be null.

Usually the configuration information of a card terminal is stored in the CardTerminalRegistry.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalInit(string)

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.8 override bool Subsembly.SmartCard.CardPcScTerminal.InternalIsCardPresent (int *nSlot*, ref bool *requiresRemoval*) [inline], [protected], [virtual]

Quick poll that indicates whether a card is inserted in a card terminal slot. This method also takes care of a connection refresh in case a pending card transaction is detected. This is to prevent PC/SC level transaction timeouts introduced with Windows 8. We are doing this here, because main polling thread is calling this method indirectly whenever it asks for a reader status update. Any PC/SC level errors must be caught here to prevent unhandled exceptions. Unless we detect removed cards, error codes are mapped to a card terminal DriverException which will then trigger a lost terminal event.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nSlot	Slot number of card terminal slot of interest.
requiresRemoval	[in,out] Flag to indicate that user must remove card.

Returns

Returns true if a card is present in the indicated slot and false if not.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalIsCardPresent(int,bool@)

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.9 override CardActivationResult Subsembly.SmartCard.CardPcScTerminal.InternalReconnectCard (int *nSlot*, out byte[ ] *vbATR*) [inline], [protected], [virtual]

Attempts to re-connect to the card.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nSlot	Slot number of card terminal slot of interest.
vbATR	Card ATR.

Returns

Returns the result from the attempt to activate the card. See CardActivationResult for possible return values.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalReconnectCard(int)

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.10 override void Subsembly.SmartCard.CardPcScTerminal.InternalResetCardTerminal ( ) [inline], [protected], [virtual]

Entirely resets the card terminal, disconnecting (powering down) any existing card connection.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalResetCardTerminal()

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.11 override CardResponseAPDU Subsembly.SmartCard.CardPcScTerminal.InternalSendCommand (int *nSlot*, CardCommandAPDU *aCommandAPDU*) [inline], [protected], [virtual]

Abstract method that must be overriden by a concrete card terminal implementation class to sends a given command APDU to the card reader.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nSlot	Slot number of card terminal slot of interest.
aCommandAP↩ DU	Complete command APDU to be transmitted.

Returns

Returns the response APDU from the inserted card.

See also

M:Subsembly.SmartCard.CardTerminalBase.InternalSendCommand(int,CardCommandAPDU)

Implements Subsembly.SmartCard.CardTerminalBase.

6.14.3.12 void Subsembly.SmartCard.CardPcScTerminal.ReadProperties ( ) [inline]

Determine the properties of the card terminal by establishing a direct connection. Note that this will fail if any other application already accesses the same reader. We handle this in a benign way, by just ignoring the failed direct connection as it might be prevented by a shared violation due to a card already being powered up during instantiation of the card terminal class.

Accessing a card reader directly can interfere with third-party smart card services. That's why we don't use this approach unless absolutely necessary. ReadProperties() is fully functional. However as of 20NOV2013 we don't call it anymore as part of our efforts to remove this method.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

6.14.4 Property Documentation

6.14.4.1 SCardAccessMode Subsembly.SmartCard.CardPcScTerminal.AccessMode [get], [set]

Selects the PC/SC access mode that will be used by the card connection. The default access mode is SHARED unless set otherwise in the registry. It can be changed to EXCLUSIVE by the HKLM key. "1" or no key means SHARED any other value translates to EXCLUSIVE.

The default value for this property is derived from the REG_SZ registry value `HKLM`. If the registry value is "1", then the default AccessMode will be Subsembly.SmartCard.PcSc.SCardAccessMode.Shared. In any other case the default AccessMode will be Subsembly.SmartCard.PcSc.SCardAccessMode.Exclusive. Changing this property, however, does not change the registry value.

The access mode.

6.14.4.2 SCardComm Subsembly.SmartCard.CardPcScTerminal.CardComm [get]

Reference to SCardComm that is used by this class. Don't use this reference unless absolutely necessary.

The card communications.

6.14.4.3 SCardResourceManager Subsembly.SmartCard.CardPcScTerminal.CardResourceManager [get]

Reference to SCardResourceManager that is used by this class. Don't use this reference unless absolutely necessary.

The card resource manager.

6.14.4.4 bool Subsembly.SmartCard.CardPcScTerminal.IsClass2 [get]

Gets a value indicating whether this object is class 2.

True if this object is class 2, false if not.

See also

P:Subsembly.SmartCard.ICardTerminalClass2.IsClass2

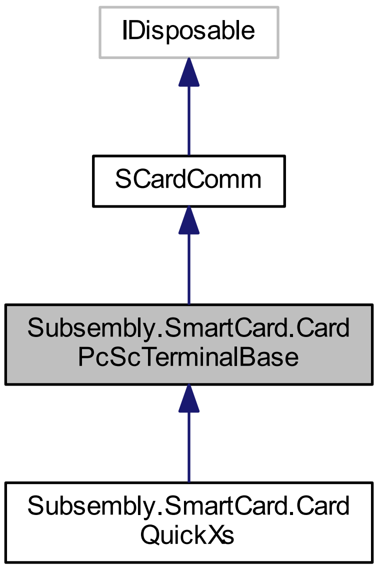
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardPcSc↩ Terminal.cs

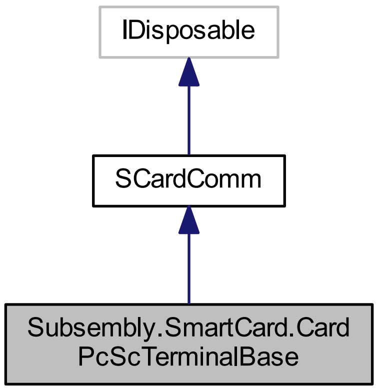
6.15 Subsembly.SmartCard.CardPcScTerminalBase Class Reference

Higher level smart card access class based on the PC/SC access class SCardComm.

Inheritance diagram for Subsembly.SmartCard.CardPcScTerminalBase:


IDisposable
SCardComm
Subsembly.SmartCard.Card PcScTerminalBase
Subsembly.SmartCard.Card QuickXs

Collaboration diagram for Subsembly.SmartCard.CardPcScTerminalBase:



Public Member Functions

- CardPcScTerminalBase (SCardResourceManager aResourceManager)

  Constructs this CardPcScTerminalBase instance.
- CardResponseAPDU SendCommand (CardCommandAPDU aCommandAPDU)

  Sends the command APDU to the card and returns its response APDU.

Properties

- byte[ ] ATR `[get]`

  Provides the ATR of the connected card.
- string TerminalSerialNumber `[get]`

  Provides the vendor supplied terminal serial number.
- bool CanVerifyPinSecure `[get]`

  Indicates whether a secure PIN entry is possible with the connected reader.
- bool CanModifyPinSecure `[get]`

  Indicates whether a secure PIN modification is possible with the connected reader.

Additional Inherited Members

6.15.1 Detailed Description

Higher level smart card access class based on the PC/SC access class SCardComm.

See also

T:Subsembly.SmartCard.PcSc.SCardComm

6.15.2 Constructor & Destructor Documentation

6.15.2.1 Subsembly.SmartCard.CardPcScTerminalBase.CardPcScTerminalBase (SCardResourceManager *aResourceManager*) `[inline]`

Constructs this CardPcScTerminalBase instance.

Parameters

aResource←Manager	Reference to the PC/SC SCardResourceManager instance.

6.15.3 Member Function Documentation

6.15.3.1 CardResponseAPDU Subsembly.SmartCard.CardPcScTerminalBase.SendCommand (CardCommandAPDU *aCommandAPDU*) `[inline]`

Sends the command APDU to the card and returns its response APDU.

The PC/SC Workgroup API is designed to send TPDUs and not APDUs. This means that this method has to and does handle the correct transmission of APDUs over T=0 or T=1 TPDUs using GET RESPONSE and so on.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

aCommandAP↩ DU	The command APDU to be sent. Must not be `null`

Returns

The response APDU from the card.

6.15.4 Property Documentation

6.15.4.1 byte [ ] Subsembly.SmartCard.CardPcScTerminalBase.ATR `[get]`

Provides the ATR of the connected card.

The value is `null` if no card has been successfully connected so far.

6.15.4.2 bool Subsembly.SmartCard.CardPcScTerminalBase.CanModifyPinSecure `[get]`

Indicates whether a secure PIN modification is possible with the connected reader.

True if we can modify pin secure, false if not.

6.15.4.3 bool Subsembly.SmartCard.CardPcScTerminalBase.CanVerifyPinSecure `[get]`

Indicates whether a secure PIN entry is possible with the connected reader.

True if we can verify pin secure, false if not.

6.15.4.4 string Subsembly.SmartCard.CardPcScTerminalBase.TerminalSerialNumber `[get]`

Provides the vendor supplied terminal serial number.

The value is `null` if the serial number is not available.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardPcSc↩ TerminalBase.cs

6.16 Subsembly.SmartCard.CardPinControl Class Reference

Collection of attributes and methods that control the PIN handling and its transformation into a command APDU sent to the smart card.

Public Member Functions

- CardPinControl (CardCommandAPDU aTemplate, CardPinEncoding nEncoding, int nOffset)

Construct a CardPinControl instance from the minimal information required in order to perform a PIN entry and verification.

- CardPinControl (CardCommandAPDU aTemplate, CardPinEncoding nEncoding, int nOffset, int nOffset2)

  Construct a CardPinControl instance from the minimal information required in order to perform entry of two PINs and perform a PIN modification.
- bool IsAcceptablePin (string sPin)

  Checks a PIN whether it is acceptable according to this PIN control.
- byte[ ] EncodePin (string sPin)

  Helper function that encodes the given PIN as required by this CardPinControl.
- CardCommandAPDU BuildCardCommandAPDU (string sPin)

  Builds the VERIFY command APDU to be sent to the card by filling the given PIN into the command APDU template.
- CardCommandAPDU BuildTerminalCommandAPDU (int nSlot, string sLabel, int nTimeout)

  Builds a command APDU to be sent to the terminal in order to perform a secure PIN entry by itself.
- CardCommandAPDU BuildModifyCardCommandAPDU (string sPin1, string sPin2)

  Builds the MODIFY command APDU to be sent to the card by filling the given PINs into the command APDU template.
- CardCommandAPDU BuildTerminalModifyCommandAPDU (int nSlot, string[ ] vsLabels, int nTimeout)

  Builds a command APDU to be sent to the terminal in order to perform a secure PIN modification by itself.

Static Public Member Functions

- static byte[ ] EncodePin (CardPinEncoding nPinEncoding, string sPin)

  Helper function that encodes the given PIN as required by this CardPinControl.

Properties

- CardCommandAPDU Template `[get]`

  The command APDU template that will be used to construct the final VERIFY PIN (or similar) command.
- CardPinEncoding Encoding `[get]`

  The PIN encoding that is used when inserting the PIN into the command APDU template.
- int Offset `[get]`

  Byte offset where the PIN shall be inserted into the command APDU template.
- int Offset2 `[get]`

  Byte offset where the second PIN shall be inserted into the command APDU template.
- int PinBlockLength `[get, set]`

  New in version 4.0: The byte length of a complete PIN block including any padding.
- int MinLength `[get, set]`

  The minimum acceptable length of the PIN.
- int MaxLength `[get, set]`

  The maximum acceptable length of the PIN.
- bool MustConfirm `[get, set]`

  True if the card holder must confirm his PIN entry by pressing the ENTER or OK button.
- bool IsAlphanumeric `[get]`

  Indicates whether the PIN entry may be alphanumeric.

6.16.1 Detailed Description

Collection of attributes and methods that control the PIN handling and its transformation into a command APDU sent to the smart card.

This class supports handling of one or two PINs. Two PINs are required when modifying a PIN. If two PINs are used, they must have the same general restrictions.

With version 4.0 several changes and additions have been made to this class. All changes are marked with a "New in version 4.0" tag.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Subsembly.SmartCard.CardPinControl.CardPinControl (**CardCommandAPDU** *aTemplate,* **CardPinEncoding** *nEncoding,* int *nOffset*) [inline]

Construct a CardPinControl instance from the minimal information required in order to perform a PIN entry and verification.

Exceptions

ArgumentNullException	The parameter aTemplate was `null`.
ArgumentException	The parameter aTemplate is an extended command APDU; or the requested P←IN encoding is not known; or the requested PIN encoding is CardPinEncoding.←Format2PinBlock and there is not enough room for an eight byte PIN block starting at nOffset.
ArgumentOutOfRange←Exception	The offset is negative or outside the available data field.

Parameters

aTemplate	Command APDU template that will be used to construct the final VERIFY PIN (or similar) command. This must not be null and must be a valid ISO 7816-4 command APDU with short length fields. If the parameter nOffset points to the end of the given data field, then the PIN will be appended to the command data and the length Lc is implicitly adjusted to accomodate the actual PIN length.
nEncoding	Selects the encoding of the PIN when it is inserted into the command APDU template.
nOffset	Byte offset where the encoded PIN shall be inserted into the command APDU template. The offset is computed relative to the beginning of the data field of the command APDU. The offset must fall within the data field of the given command APDU template.

6.16.2.2 Subsembly.SmartCard.CardPinControl.CardPinControl (**CardCommandAPDU** *aTemplate,* **CardPinEncoding** *nEncoding,* int *nOffset,* int *nOffset2*) [inline]

Construct a CardPinControl instance from the minimal information required in order to perform entry of two PINs and perform a PIN modification.

Exceptions

ArgumentOutOfRange←Exception	Either offset is negative or outside the available data field.

ArgumentException	The parameter aTemplate is an extended command APDU; or the requested P↩ IN encoding is not known; or the requested PIN encoding is CardPinEncoding.↩ Format2PinBlock and there is not enough room for an eigth byte PIN block starting at nOffset or nOffset2.

Parameters

aTemplate	Command APDU template that will be used to construct the final MODIFY PIN (or similar) command. This must not be null and must be a valid ISO 7816-4 command APDU with short length fields. If the parameters nOffset and nOffset2, both point to the end of the given data field, then the old PIN and the new PIN will be appended to the command data and the length Lc is implicitly adjusted to accomodate the actual PIN lengths.
nEncoding	Selects the encoding of the PINs when they are inserted into the command APDU template.
nOffset	Byte offset where the first encoded PIN shall be inserted into the command APDU template. The offset is computed relative to the data field and must fall within the data field.
nOffset2	Byte offset where the second encoded PIN shall be inserted into the command APDU template. The offset is specified similar to the nOffset parameter.

6.16.3 Member Function Documentation

6.16.3.1 CardCommandAPDU Subsembly.SmartCard.CardPinControl.BuildCardCommandAPDU (string *sPin*) [inline]

Builds the VERIFY command APDU to be sent to the card by filling the given PIN into the command APDU template.

The given PIN will be checked whether it can be encoded and inserted into the command APDU template according to the properties of this class. If successful, then a new command APDU instance that includes the given PIN will be returned. The command APDU template owned by this class will not be changed. Thus this instance can be re-used in order to create multiple command APDUs.

Parameters

sPin	The PIN to be filled into the command APDU template.

Returns

If successful a new instance of the CardCommandAPDU class is returned that contains the command APDU template filled with the encoded PIN. The returned command APDU can then be sent to the card terminal in order to actually perform the PIN verification.
If the given PIN is not acceptable or it cannot be encoded as required, then null will be returned. Use Is↩ AcceptablePin to check a given PIN before attempting to build a card command APDU.

6.16.3.2 CardCommandAPDU Subsembly.SmartCard.CardPinControl.BuildModifyCardCommandAPDU (string *sPin1*, string *sPin2*) [inline]

Builds the MODIFY command APDU to be sent to the card by filling the given PINs into the command APDU template.

Parameters

sPin1	The first pin.

sPin2	The second pin.

Returns

If successful a new instance of the CardCommandAPDU class is returned that contains the command APDU template filled with the encoded PINs. The returned command APDU can then be sent to the card terminal in order to actually perform the PIN modification.
If one of the given PINs is not acceptable or it cannot be encoded as required, then null will be returned. Use IsAcceptablePin to check a PIN before attempting to build a card command APDU.

6.16.3.3 CardCommandAPDU Subsembly.SmartCard.CardPinControl.BuildTerminalCommandAPDU (int *nSlot,* string *sLabel,* int *nTimeout*) [inline]

Builds a command APDU to be sent to the terminal in order to perform a secure PIN entry by itself.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.
NotSupportedException	Thrown when the requested operation is not supported.

Parameters

nSlot	The card terminal slot number of the slot where the card to receive the verify command is inserted.
sLabel	According to the CT-BCS specification only ASCII characters are allowed. The max length of the display string is 16 characters. If this is null, then no label will be set.
nTimeout	

Maximum time to wait for the user to enter the first digit (or letter) of the PIN. This time is given in seconds. If the user does not begin to enter his PIN within this timeout, then the PIN entry is automatically cancelled. Usually there is also an implicit timeout between every digit of the PIN, but this is implementation or device specific and cannot be controlled by this method.

If this is zero, then some card termnal specific default timeout will be applied. Very large timeouts are shortened to the maximum of 255 seconds. Negative values are not allowed.

Returns

If successful a new instance of the CardCommandAPDU class is returned that contains a command APDU according to CT-BCS that can be sent to the card terminal in order to perform a secure PIN entry.

6.16.3.4 CardCommandAPDU Subsembly.SmartCard.CardPinControl.BuildTerminalModifyCommandAPDU (int *nSlot,* string[ ] *vsLabels,* int *nTimeout*) [inline]

Builds a command APDU to be sent to the terminal in order to perform a secure PIN modification by itself.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

NotSupportedException	Thrown when the requested operation is not supported.

Parameters

nSlot	The card terminal slot number of the slot where the card to receive the modify command is inserted.
vsLabels	

Optional array of text labels that replace the default labels. If null, then no special labels will be specified in the created terminal command APDU. If it is provided, then it must contain 6 strings that replace the following labels in the order given.

These default texts are longer than 16 characters !?! Please enter PIN Action successful PIN wrong or blocked Please enter new PIN Repeat input PIN not identical. Abort

According to the CT-BCS specification only ASCII characters are allowed. The max length of the display string is 16 characters. If this is null, then no label will be set.

Parameters

nTimeout	

Maximum time to wait for the user to enter the first digit (or letter) of the PIN. This time is given in seconds. If the user does not begin to enter his PIN within this timeout, then the PIN entry is automatically cancelled. Usually there is also an implicit timeout between every digit of the PIN, but this is implementation or device specific and cannot be controlled by this method.

If this is zero, then some card terminal specific default timeout will be applied. Very large timeouts are shortened to the maximum of 255 seconds. Negative values are not allowed.

Returns

If successful a new instance of the CardCommandAPDU class is returned that contains a command APDU according to CT-BCS that can be sent to the card terminal in order to perform a secure PIN entry.

6.16.3.5 byte [ ] Subsembly.SmartCard.CardPinControl.EncodePin (string *sPin*) `[inline]`

Helper function that encodes the given PIN as required by this CardPinControl.

The given PIN will be checked whether it can be encoded according to the properties of this class. If not, then null will be returned.

Parameters

sPin	The PIN to be encoded.

Returns

Returns a byte array that contains the encoded PIN. The length of the byte array depends on the length of the PIN and the selected encoding. If a BCD encoding is selected, and the PIN has an odd length, then the final nibble will have all bits set.
If the given PIN cannot be encoded, then null is returned.

See also

BuildCardCommandAPDU

6.16.3.6 static byte [ ] Subsembly.SmartCard.CardPinControl.EncodePin (CardPinEncoding *nPinEncoding,* string *sPin*) `[inline]`, `[static]`

Helper function that encodes the given PIN as required by this CardPinControl.

The given PIN will be checked whether it can be encoded according to the properties of this class. If not, then null will be returned.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

nPinEncoding	.
sPin	The PIN to be encoded.

Returns

Returns a byte array that contains the encoded PIN. The length of the byte array depends on the length of the PIN and the selected encoding. If a BCD encoding is selected, and the PIN has an odd length, then the final nibble will have all bits set.
If the given PIN cannot be encoded, then null is returned.

6.16.3.7 bool Subsembly.SmartCard.CardPinControl.IsAcceptablePin (string *sPin*) `[inline]`

Checks a PIN whether it is acceptable according to this PIN control.

It is checked whether the PIN length is acceptable according to MinLength and MaxLength and whether the PIN can be encoded as required (i.e. is numeric or alphanumeric just as required).

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

sPin	The PIN to be checked.

Returns

Returns `true` if the PIN is acceptable and `false` if not.

6.16.4 Property Documentation

6.16.4.1 CardPinEncoding Subsembly.SmartCard.CardPinControl.Encoding `[get]`

The PIN encoding that is used when inserting the PIN into the command APDU template.

This is the value that was passed to the constructor of this class. It cannot be changed thereafter. Note that the PIN encoding defines whether a numeric or an alphanumeric PIN may be used.

See also

IsAlphanumeric

6.16.4.2 bool Subsembly.SmartCard.CardPinControl.IsAlphanumeric `[get]`

Indicates whether the PIN entry may be alphanumeric.

If the selected PIN encoding supports representation of alphanumeric PINs, then this property will be true. If the selected PIN encoding supports numeric PINs only, then this will be false.

6.16.4.3 int Subsembly.SmartCard.CardPinControl.MaxLength `[get], [set]`

The maximum acceptable length of the PIN.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

By default the maximum acceptable length of the PIN is derived from the encoding and the command template. If the maximum length is set to a value smaller than the minimum length, then the minimum length is set equal to the maximum length. The maximum length cannot be set to a negative value, and cannot be set to zero.

If the selected PIN encoding is CardPinEncoding.Format2PinBlock, then the maximum PIN length cannot be less than four or more than twelve. An attempt to set a value outside the required range for a CardPinEncoding.Format2PinBlock will cause an ArgumentOutOfRangeException to be thrown.

Input devices or dialogs may or may not implicitly honor this requirement.

See also

MinLength

6.16.4.4 int Subsembly.SmartCard.CardPinControl.MinLength `[get], [set]`

The minimum acceptable length of the PIN.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

By default the minimum acceptable length of the PIN is determined by the chosen PIN encoding. Most of the time the default will be zero. The minimum length cannot be set to a negative value, but may be set to zero.

If the selected PIN encoding is CardPinEncoding.Format2PinBlock, then the minimum PIN length cannot be less than four or more than twelve. An attempt to set a value outside the required range for a CardPinEncoding.Format2PinBlock will cause an ArgumentOutOfRangeException to be thrown.

Input devices or dialogs may or may not implicitly honor this requirement.

See also

MaxLength

6.16.4.5 bool Subsembly.SmartCard.CardPinControl.MustConfirm `[get], [set]`

True if the card holder must confirm his PIN entry by pressing the ENTER or OK button.

If this property is set to true, then the card holder must confirm every PIN entry by pressing the ENTER or OK button. By default this property is set to `true`.

6.16.4.6 int Subsembly.SmartCard.CardPinControl.Offset `[get]`

Byte offset where the PIN shall be inserted into the command APDU template.

This is the value that was passed to the constructor of this class. It cannot be changed thereafter.

6.16.4.7 int Subsembly.SmartCard.CardPinControl.Offset2 `[get]`

Byte offset where the second PIN shall be inserted into the command APDU template.

This is the value that was passed to the constructor of this class. It is -1 if it was not specified, i.e. second PIN won't be used. It cannot be changed thereafter.

6.16.4.8 int Subsembly.SmartCard.CardPinControl.PinBlockLength `[get]`, `[set]`

New in version 4.0: The byte length of a complete PIN block including any padding.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

This value is computed from the parameters given to the constructor, but may be changed.

This may be zero, if the PIN is actually appended to the command data.

6.16.4.9 CardCommandAPDU Subsembly.SmartCard.CardPinControl.Template `[get]`

The command APDU template that will be used to construct the final VERIFY PIN (or similar) command.

This is the value that was passed to the constructor of this class. It cannot be changed thereafter. Thus it is never null.

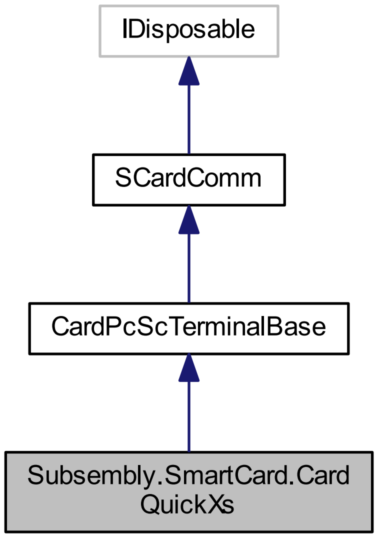
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardPin↩ Control.cs

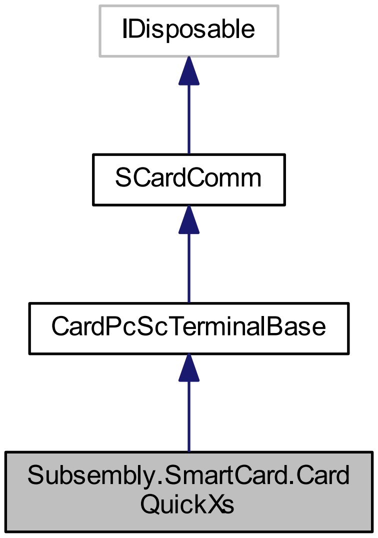
6.17 Subsembly.SmartCard.CardQuickXs Class Reference

Higher level smart card access class based on the PC/SC access classSCardComm. It is meant for quick access to smart card system in a way that is still relatively close to the native API.

Inheritance diagram for Subsembly.SmartCard.CardQuickXs:



Collaboration diagram for Subsembly.SmartCard.CardQuickXs:



Public Member Functions

- CardQuickXs (SCardResourceManager aResourceManager)

  Constructs this CardQuickXs instance.
- CardResponseAPDU SelectApplication (byte[ ] vbApplicationID)

  Simplifies the selection of a card application.
- CardResponseAPDU SelectRoot ()

  Generic ISO 7816-4 command for selecting the master file (MF).
- CardResponseAPDU SelectFile (int nFileID)

  Generic ISO 7816-4 command for selecting a file.
- CardResponseAPDU ReadBinary (int nOffset, byte nLength)

  Generic ISO 7816-4 command for reading data from the currently selected elementary file with transparent structure.
- CardResponseAPDU ReadBinary (byte nSFI, byte nOffset, byte nLength)

  Generic ISO 7816-4 command for reading data from a elementary file (EF) with transparent structure. The EF will be implicitly selected through a short file identifier (SFI).
- CardResponseAPDU UpdateBinary (int nOffset, byte[ ] vbData)

  Generic ISO 7816-4 command for updating data in the currently selected elementary file with transparent structure.
- CardResponseAPDU WriteBinary (int nOffset, byte[ ] vbData)

  Generic ISO 7816-4 command for writing data to the currently selected elementary file with transparent structure.
- CardResponseAPDU ReadRecord (byte nRecNo, byte nLength)

  Generic ISO 7816-4 command for reading a record from the currently selected elementary file with record structure.
- CardResponseAPDU UpdateRecord (byte nRecNo, byte[ ] vbData)

  Generic ISO 7816-4 command for updating data in the currently selected elementary file with record structure.
- CardResponseAPDU SendCommandEx (CardCommandAPDU aCommandAPDU)

  Sends a command exception.

Properties

- byte CLA `[get, set]`

  The class byte to be used when sending generic ISO 7816-4 command APDUs to the card.

Additional Inherited Members

6.17.1 Detailed Description

Higher level smart card access class based on the PC/SC access classSCardComm. It is meant for quick access to smart card system in a way that is still relatively close to the native API.

This class does not provide the comfort of events such as card lost/found or reader lost/found. Therefore it is simple and easy-to-use. Use this class for direct reader access where no real card handle is available and card found/lost events can't be expected. Example: communicating with a reader that exposes proprietary pseudo APDUs CardCommandA↩ PDU and CardResponseAPDU instances.

See also

T:Subsembly.SmartCard.CardPcScTerminalBase

6.17.2 Constructor & Destructor Documentation

6.17.2.1 Subsembly.SmartCard.CardQuickXs.CardQuickXs (**SCardResourceManager** *aResourceManager*) `[inline]`

Constructs this CardQuickXs instance.

Parameters

aResource↩ Manager	Reference to the PC/SC SCardResourceManager instance.

Parameter aResourceManager was null.

6.17.3 Member Function Documentation

6.17.3.1 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.ReadBinary (int *nOffset*, byte *nLength*) [inline]

Generic ISO 7816-4 command for reading data from the currently selected elementary file with transparent structure.

This method uses the following generic command APDU: `CLA 'B0' HI LO LE`, where HI and LO are the high and low byte of the given offset and LE is the given length byte.

Exceptions

ArgumentOutOfRange↩ Exception	The given parameter nOffset was invalid.

Parameters

nOffset	Byte offset where to begin reading. The offset must be in the range from 0 through hex '7FFF'.
nLength	Number of bytes to read. If this is zero, then all available bytes up to the end of the file are read.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the data was read successfully. The read data is available through the CardResponseAPDU.GetData method.

Exception from the base class

SCardComm.Transmit method.

6.17.3.2 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.ReadBinary (byte *nSFI*, byte *nOffset*, byte *nLength*) [inline]

Generic ISO 7816-4 command for reading data from a elementary file (EF) with transparent structure. The EF will be implicitly selected through a short file identifier (SFI).

This method uses the following generic command APDU: `CLA 'B0' '80'+SFI OF LE`, where SFI is the short file identifier, and OF is the given offset, and LE is the given length byte.

Exceptions

ArgumentOutOfRange↩ Exception	The given parameter nSFI is invalid.

Parameters

nSFI	The short file identifier (SFI) of the EF from which the binary data shall be read. A valid SFI must be in the range from zero through 31.

nOffset	Byte offset where to begin reading.
nLength	Number of bytes to read. If this is zero, then all available bytes up to the end of the file are read.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the data was read successfully. The read data is available through the CardResponseAPDU.GetData method.

Exception from the base class

SCardComm.Transmit method.

6.17.3.3 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.ReadRecord (byte *nRecNo,* byte *nLength*) `[inline]`

Generic ISO 7816-4 command for reading a record from the currently selected elementary file with record structure.

This method uses the following generic command APDU: `CLA 'B2' RN '04' LE`, where RN is record number byte and LE is the given length byte.

Note that this method does not work with ACOS1 cards which would need the byte value '00' in P2 instead of the correct ISO 7816-4 value '04' which is used here.

Parameters

nRecNo	Absolute record number of record to be read. The first record usually has the record number 1.
nLength	Number of bytes to read from record. If this is zero, then the entire record is read.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the record was read successfully. The read record is available through the CardResponseAPDU.GetData method.

Exception from the base class

SCardComm.Transmit method.

6.17.3.4 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.SelectApplication (byte[ ] *vbApplicationID*) `[inline]`

Simplifies the selection of a card application.

An attempt is made to select the card application with the given application ID (AID). The Selection is attempted using the most simple ISO 7816-4 SELECT command: 'CLA A4 04 0C'.

The class byte for the SELECT command will be taked from the CLA property.

Exceptions

ArgumentNullException	The parameter vbApplicationID was `null`.
ArgumentException	The vbApplicationID byte array was empty or longer than 256 bytes.

Parameters

vbApplicationID	The application ID of the application to be selected.

Returns

In any case, the response APDU of the selection command will be returned to the caller.

Exception from the base class

SCardComm.Transmit method.

6.17.3.5 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.SelectFile (int *nFileID*) `[inline]`

Generic ISO 7816-4 command for selecting a file.

The file to be selected must be an elemtary file (EF).

This method uses the following generic command APDU: `CLA 'A4' '02' '0C' FILEID`, where FILEID is the given two byte file ID.

Exceptions

ArgumentOutOfRange↩ Exception	The given nFileID was outside the permitted range.

Parameters

nFileID	The file ID (FID) of the file to be selected. The file ID must be in the range from hex '0000' through hex 'FFFF'.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the file was selected successfully.

Exception from the base class

SCardComm.Transmit method.

6.17.3.6 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.SelectRoot ( ) `[inline]`

Generic ISO 7816-4 command for selecting the master file (MF).

This method issues the following generic command APDU: `CLA 'A4' '00' '0C' '02' '3F 00'`.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the file was selected successfully.

6.17.3.7 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.SendCommandEx (CardCommandAPDU *aCommandAPDU*) `[inline]`

Sends a command exception.

If the card returns the status word '61 XX' or '9F XX' and no data was returned, then a GET RESPONSE command is issued in order to retrieve the pending response. If the card returns '6C XX', then the very same command is tried again, but this time with a correct Le field.

Parameters

aCommandAP↩ DU	.

Returns

A CardResponseAPDU.

6.17.3.8 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.UpdateBinary (int *nOffset*, byte[ ] *vbData*) `[inline]`

Generic ISO 7816-4 command for updating data in the currently selected elementary file with transparent structure.

This method uses the following generic command APDU: `CLA 'D6' HI LO LC DATA`, where HI and LO are the high and low byte of the given offset, LC is the length of the given data and DATA is the given data array itself.

With some readers, this method can also be used to update data on a synchronous memory card. The UPDATE BIN↩ ARY APDU sent by this method conforms to the MCT specification part 7 (IC Cards with Synchronouse Transmission, Usage of Interindustry Commands) which can be downloaded from `www.teletrust.de`

Exceptions

ArgumentOutOfRange↩ Exception	The parameter nOffset was outside the permitted range.
ArgumentNullException	The parameter vbData was `null`.
ArgumentException	The parameter vbData was invalid.

Parameters

nOffset	Byte offset where to begin updating data. The offset must be in the range from 0 through hex '7FFF'.
vbData	Data that shall be written to the file at the given offset. Must not be `null` and must not have a zero length or be longer than 256 bytes.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the update was successful.

Exception from the base class

SCardComm.Transmit method.

6.17.3.9 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.UpdateRecord (byte *nRecNo*, byte[ ] *vbData*) `[inline]`

Generic ISO 7816-4 command for updating data in the currently selected elementary file with record structure.

This method uses the following generic command APDU: `CLA 'DC' RN '04' LC DATA`, where RN is record number byte, LC is the length of the given data and DATA is the given data array itself.

Exceptions

ArgumentNullException	The parameter vbData was `null`.
ArgumentException	The array vbData was empty or longer than 256 bytes.

Parameters

nRecNo	Absolute record number of record to be updated. The first record usually has the record number 1.
vbData	Data that shall be written to the record. Must not be `null` and must not have a zero length or be longer than 256 bytes. Usually the entire record must be updated, thus the given data must match the size of the record structure on the smart card.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the update was successful.

Exception from the base class

SCardComm.Transmit method.

6.17.3.10 CardResponseAPDU Subsembly.SmartCard.CardQuickXs.WriteBinary (int *nOffset*, byte[ ] *vbData*) [inline]

Generic ISO 7816-4 command for writing data to the currently selected elementary file with transparent structure.

This method uses the following generic command APDU: `CLA 'D0' HI LO LC DATA`, where HI and LO are the high and low byte of the given offset, LC is the length of the given data and DATA is the given data array itself.

With some readers, this method can also be used to write data to a synchronous memory card. The WRITE BINARY APDU sent by this method conforms to the Kobil B1 smart card reader which is emulated by readers that claim to be B1 compatible.

Exceptions

ArgumentOutOfRange↩ Exception	The parameter nOffset was outside the permitted range.
ArgumentNullException	The parameter vbData was `null`.
ArgumentException	The parameter vbData was invalid.

Parameters

nOffset	Byte offset where to begin writing data. The offset must be in the range from 0 through hex '7FFF'.
vbData	Data that shall be written to the file at the given offset. Must not be `null` and must not have a zero length or be longer than 256 bytes.

Returns

The complete response APDU is returned. The property CardResponseAPDU.IsSuccessful can be used to check whether the update was successful.

Exception from the base class

SCardComm.Transmit method.

6.17.4 Property Documentation

6.17.4.1 byte Subsembly.SmartCard.CardQuickXs.CLA `[get], [set]`

The class byte to be used when sending generic ISO 7816-4 command APDUs to the card.

Depending on the smart card operating system and whether secure messaging or logical channels are used, a different class byte value may be required. For example, GSM SIM cards always use a class byte of 'A0', ACOS1 cards always use a class byte of '80'. Contactless storage cards and pseudo APDUs mostly use CLA='FF'

The default class byte value is '00'.

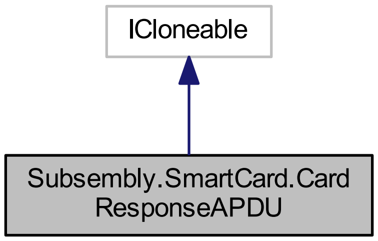
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardQuickXs.cs

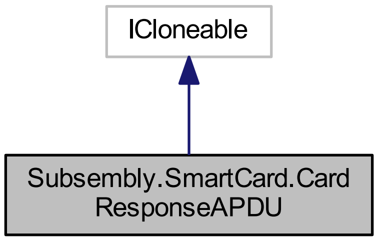
6.18 Subsembly.SmartCard.CardResponseAPDU Class Reference

Generic utility class for evaluating and processing of smart card response APDUs according to ISO 7816-4.

Inheritance diagram for Subsembly.SmartCard.CardResponseAPDU:



Collaboration diagram for Subsembly.SmartCard.CardResponseAPDU:



Public Member Functions

- CardResponseAPDU (int nSW)

  Construct a CardResponseAPDU instance that contains just a status word SW.
- CardResponseAPDU (byte[ ] vbData, int nSW)

  Construct a CardResponseAPDU instance from a the response data and the status word SW.
- CardResponseAPDU (byte[ ] vbData, int nOffset, int nLength, int nSW)

  Construct a CardResponseAPDU instance from a the response data and the status word SW.
- object Clone ()

  Creates a new CardResponseAPDU instance that is a complete and deep copy of this CardResponseAPDU instance.
- byte[ ] GenerateBytes ()

  Generates the complete response APDU into a newly allocated raw byte array.
- byte[ ] GetData ()

  Returns the data field of an ISO 7816-4 response APDU.
- void SetData (byte[ ] vbData)

  Adds or replaces the data field of this response APDU.
- void SetData (byte[ ] vbData, int nOffset, int nLength)

  Adds or replaces the data field of this response APDU.
- void AppendData (byte[ ] vbData)

  Appends the given data block to the data field of this response APDU.
- void TruncateData (int nLr)

  Truncates the data block of this response APDU at the given size.
- override string ToString ()

  Formats this response APDU, including all its data, in a nice human readable fashion.

Static Public Member Functions

- static CardResponseAPDU ParseBytes (byte[ ] vbResponseAPDU)

  Creates a CardResponseAPDU instance by parsing a response APDU given as a byte array.
- static CardResponseAPDU ParseBytes (byte[ ] vbResponseAPDU, int nOffset, int nLength)

  Creates a CardResponseAPDU instance by parsing a response APDU given as a range inside a byte array.

Public Attributes

- const int MAXSHORTLENGTH = 258

  The max length in bytes of a single ISO 7816-4 response APDU when short length fields are used.
- const int MAXEXTENDEDLENGTH = 65538

  The max length in bytes of a single ISO 7816-4 response APDU when extended length fields are used.
- const int MINLENGTH = 2

  The min length of any ISO 7816-4 response APDU.
- const int MAXSHORTLR = 256

  The max size of the data field of an ISO 7816-4 response APDU with short length fields.
- const int MAXEXTENDEDLR = 65536

  The max size of the data field of an ISO 7816-4 response APDU with extended length fields.

Properties

- int SW `[get, set]`

    Access to the status word of an ISO 7816-4 response APDU.
- byte SW1 `[get, set]`

    Access to the byte SW1 of the status word of an ISO 7816-4 response APDU.
- byte SW2 `[get, set]`

    Access to the byte SW2 of the status word of an ISO 7816-4 response APDU.
- bool IsSuccessful `[get]`

    Is true when the status word indicates success according to ISO 7816-4.
- bool IsWarning `[get]`

    Is true when the status word indicates a warning according to ISO 7816-4.
- bool IsError `[get]`

    Is true when the status word indicates an error according to ISO 7816-4.
- bool IsExecutionError `[get]`

    Is true when the status word indicates an execution error according to ISO 7816-4.
- bool IsCheckingError `[get]`

    Is true when the status word indicates a checking error according to ISO 7816-4.
- int Lr `[get]`

    Provides the length Lr of an ISO 7816-4 response APDU. This is the actual length of the data field returned by GetData.
- bool IsExtended `[get]`

    Indicates whether this is an extended ISO 7816-4 response APDU.

6.18.1 Detailed Description

Generic utility class for evaluating and processing of smart card response APDUs according to ISO 7816-4.

See also

T:System.ICloneable

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Subsembly.SmartCard.CardResponseAPDU.CardResponseAPDU (int *nSW*) `[inline]`

Construct a CardResponseAPDU instance that contains just a status word SW.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nSW	The status word SW as an integer in the range from 0x0000 through 0xFFFF.

6.18.2.2 Subsembly.SmartCard.CardResponseAPDU.CardResponseAPDU (byte[ ] *vbData*, int *nSW*) `[inline]`

Construct a CardResponseAPDU instance from a the response data and the status word SW.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	The response data just as it was received from the smart card. The parameter must not be null. Valid response data lengths range from 1 through MAXEXTENDEDLR = 65536.
nSW	The status word SW as an integer in the range from 0x0000 through 0xFFFF.

6.18.2.3 Subsembly.SmartCard.CardResponseAPDU.CardResponseAPDU (byte[ ] *vbData*, int *nOffset*, int *nLength*, int *nSW*) [inline]

Construct a CardResponseAPDU instance from a the response data and the status word SW.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	A byte array that provides the data for the data field for this response APDU. This parameter must not be null. This method allocates a new array and copies the data from the given array. It does not keep any reference to the given array.
nOffset	Offset into byte array where the first byte shall be taken from. This must not be negative and must be within the given byte array.
nLength	Number of bytes to take from the given byte array starting at nOffset. This must not be zero or negative. The entire length must fall inside the given byte array. The length must not be longer than MAXEXTENDEDLR = 65536 bytes.
nSW	The status word SW as an integer in the range from 0x0000 through 0xFFFF.

6.18.3 Member Function Documentation

6.18.3.1 void Subsembly.SmartCard.CardResponseAPDU.AppendData (byte[ ] *vbData*) [inline]

Appends the given data block to the data field of this response APDU.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	

The data to be appended to the data field for this response APDU. The length Lr is incremented by the total length of this byte array. The total length must not exceed MAXEXTENDEDLR = 65536 bytes. This parameter must not be null.

6.18.3.2 object Subsembly.SmartCard.CardResponseAPDU.Clone ( ) [inline]

Creates a new CardResponseAPDU instance that is a complete and deep copy of this CardResponseAPDU instance.

Returns

Clone.

See also

M:System.ICloneable.Clone()

6.18.3.3 byte [ ] Subsembly.SmartCard.CardResponseAPDU.GenerateBytes ( ) [inline]

Generates the complete response APDU into a newly allocated raw byte array.

Returns

Byte array that contains the complete response APDU. This is what was actually received from the smart card. This is never null. The minimum size of the returned array is MINLENGTH = 2 the maximum size of the returned array is MAXEXTENDEDLENGTH = 65538.

6.18.3.4 byte [ ] Subsembly.SmartCard.CardResponseAPDU.GetData ( ) [inline]

Returns the data field of an ISO 7816-4 response APDU.

Returns

If the response APDU does not include a data field, then null is returned. Otherwise a reference to the byte array with the data field of the response APDU is returned.

6.18.3.5 static CardResponseAPDU Subsembly.SmartCard.CardResponseAPDU.ParseBytes (byte[ ] *vbResponseAPDU*) [inline], [static]

Creates a CardResponseAPDU instance by parsing a response APDU given as a byte array.

Exceptions

ArgumentNullException	The parameter vbResponseAPDU was null.

Parameters

vbResponseA↩ PDU	The complete response APDU just as it was received from the smart card. The parameter must not be null.

Returns

A CardResponseAPDU.

6.18.3.6 static CardResponseAPDU Subsembly.SmartCard.CardResponseAPDU.ParseBytes (byte[ ] *vbResponseAPDU*, int *nOffset*, int *nLength*) [inline], [static]

Creates a CardResponseAPDU instance by parsing a response APDU given as a range inside a byte array.

Exceptions

ArgumentNullException	The parameter vbResponseAPDU was null.
ArgumentOutOfRange↩ Exception	The given response APDU is not a valid ISO 7816-4 response APDU or the nOffset or the nLength parameter was invalid.

Parameters

vbResponseA↩ PDU	Byte array that contains the response APDU. The parameter must not be null.
nOffset	Offset into the byte array where the response APDU begins.
nLength	Length of the response APDU inside the byte array.

Returns

A CardResponseAPDU.

6.18.3.7 void Subsembly.SmartCard.CardResponseAPDU.SetData (byte[ ] *vbData*) `[inline]`

Adds or replaces the data field of this response APDU.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	

The data field to be set for this response APDU. The length Lr is derived from the total length of this byte array. If this parameter is null, then any existing data field will be removed. If it is given, it must not be empty and must not be longer than MAXEXTENDEDLR = 65536 bytes.

The CardResponseAPDU will keep a direct reference to the given byte array and will not make a copy of it.

6.18.3.8 void Subsembly.SmartCard.CardResponseAPDU.SetData (byte[ ] *vbData,* int *nOffset,* int *nLength*) `[inline]`

Adds or replaces the data field of this response APDU.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

vbData	A byte array that provides the data for the data field for this response APDU. This parameter must not be null. This method allocates a new array and copies the data from the given array. It does not keep any reference to the given array.

nOffset	Offset into byte array where the first byte shall be taken from. This must not be negative and must be within the given byte array.
nLength	Number of bytes to take from the given byte array starting at nOffset. This must not be zero or negative. The entire length must fall inside the given byte array. The length must not be longer than MAXEXTENDEDLR = 65536 bytes.

6.18.3.9 override string Subsembly.SmartCard.CardResponseAPDU.ToString ( ) [inline]

Formats this response APDU, including all its data, in a nice human readable fashion.

Returns

Returns a nicely formatted string usable for tracing output.

See also

M:System.Object.ToString()

6.18.3.10 void Subsembly.SmartCard.CardResponseAPDU.TruncateData (int *nLr*) [inline]

Truncates the data block of this response APDU at the given size.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nLr	Maximum length of response data to truncate at. This must not be negative and must not be larger than MAXEXTENDEDLR = 65536. If this is zero, then any existing data field will be deleted. If this is larger than the current size of the data field, then this method does nothing. Otherwise the first nLr bytes of the data field are copied into a new array that becomes the new data field of this response APDU.

6.18.4 Member Data Documentation

6.18.4.1 const int Subsembly.SmartCard.CardResponseAPDU.MINLENGTH = 2

The min length of any ISO 7816-4 response APDU.

6.18.5 Property Documentation

6.18.5.1 bool Subsembly.SmartCard.CardResponseAPDU.IsCheckingError [get]

Is true when the status word indicates a checking error according to ISO 7816-4.

True if this object is checking error, false if not.

6.18.5.2 bool Subsembly.SmartCard.CardResponseAPDU.IsError [get]

Is true when the status word indicates an error according to ISO 7816-4.

True if this object is error, false if not.

6.18.5.3 bool Subsembly.SmartCard.CardResponseAPDU.IsExecutionError [get]

Is true when the status word indicates an execution error according to ISO 7816-4.

True if this object is execution error, false if not.

6.18.5.4 bool Subsembly.SmartCard.CardResponseAPDU.IsExtended [get]

Indicates whether this is an extended ISO 7816-4 response APDU.

True if this object is extended, false if not.

6.18.5.5 bool Subsembly.SmartCard.CardResponseAPDU.IsSuccessful [get]

Is true when the status word indicates success according to ISO 7816-4.

This property is `true` whenever the status word is '90 00', or the SW1 of the status word is '61' or '9F'.

True if this object is successful, false if not.

6.18.5.6 bool Subsembly.SmartCard.CardResponseAPDU.IsWarning [get]

Is true when the status word indicates a warning according to ISO 7816-4.

True if this object is warning, false if not.

6.18.5.7 int Subsembly.SmartCard.CardResponseAPDU.Lr [get]

Provides the length Lr of an ISO 7816-4 response APDU. This is the actual length of the data field returned by GetData.

Depending on whether a data field was included in the response APDU, this value ranges from zero up to MAXE↩ XTENDEDLR = 65536 (command with extended length field). A length of zero means that no data field is present.

6.18.5.8 int Subsembly.SmartCard.CardResponseAPDU.SW [get], [set]

Access to the status word of an ISO 7816-4 response APDU.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

The status word is always built from the last two bytes of an ISO 7816-4 response APDU.

6.18.5.9 byte Subsembly.SmartCard.CardResponseAPDU.SW1 [get], [set]

Access to the byte SW1 of the status word of an ISO 7816-4 response APDU.

The byte SW1 is always the most significant byte of the status word.

See also

SW

6.18.5.10 byte Subsembly.SmartCard.CardResponseAPDU.SW2 `[get]`, `[set]`

Access to the byte SW2 of the status word of an ISO 7816-4 response APDU.

The byte SW2 is always the least significant byte of the status word.

See also

SW

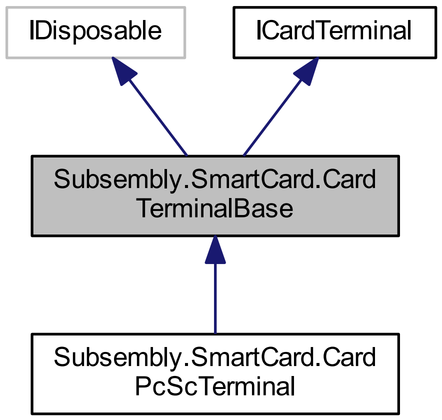
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardResponse↩ APDU.cs

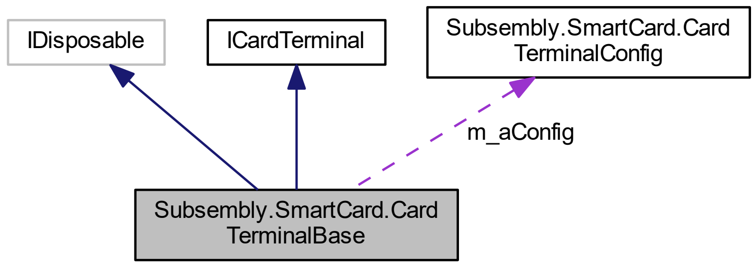
6.19 Subsembly.SmartCard.CardTerminalBase Class Reference

Common base class to simplify the implementation of the ICardTerminal interface. This class is intended for card terminal manufacturers and not for application programmers.

Inheritance diagram for Subsembly.SmartCard.CardTerminalBase:


IDisposable
ICardTerminal
Subsembly.SmartCard.Card TerminalBase
Subsembly.SmartCard.Card PcScTerminal

Collaboration diagram for Subsembly.SmartCard.CardTerminalBase:


IDisposable
ICardTerminal
Subsembly.SmartCard.Card TerminalConfig
m_aConfig
Subsembly.SmartCard.Card TerminalBase

Public Member Functions

- void Dispose ()

  Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.
- void Init (CardTerminalConfig aConfig)

  Boilerplate implementation of ICardTerminal.Init.
- void ResetCardTerminal ()

  Boilerplate implementation of ICardTerminal.ResetCardTerminal.
- bool IsCardPresent (int nSlot, ref bool requiresRemoval)

  Boilerplate implementation of ICardTerminal.IsCardPresent.
- CardActivationResult ConnectCard (int nSlot, CardTypes nCardType, out byte[ ] vbATR)

  Boilerplate implementation of ICardTerminal.ConnectCard.
- CardActivationResult ReconnectCard (int nSlot, out byte[ ] vbATR)

  Boilerplate implementation of ICardTerminal.ReconnectCard.
- CardResponseAPDU SendCommand (int nSlot, CardCommandAPDU aCommand)

  Boilerplate implementation of ICardTerminal.SendCommand.
- CardTransactionResult BeginTransaction (int nSlot)

  Boilerplate implementation of ICardTerminal.BeginTransaction.
- CardTransactionResult EndTransaction (int nSlot)

  Boilerplate implementation of ICardTerminal.EndTransaction.
- void DisconnectCard (int nSlot)

  Boilerplate implementation of ICardTerminal.DisconnectCard.
- void DeliverCard (int nSlot, bool fCapture)

  Boilerplate implementation of ICardTerminal.DeliverCard.

Protected Member Functions

- CardTerminalBase ()

  This protected constructor initializes this base class.
- virtual void Dispose (bool fDisposing)

  Must be overridden by derived classes in order to implement the Dispose pattern.

- abstract void InternalInit (string sXmlConfig)

  Abstract method that must be overriden by a concrete card terminal implementation in order to initialise the card terminal.
- abstract void InternalResetCardTerminal ()

  Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal reset functionality of the ICardTerminal interface.
- abstract bool InternalIsCardPresent (int nSlot, ref bool requiresRemoval)

  Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal card presence detection functionality of the ICardTerminal interface.
- abstract CardActivationResult InternalConnectCard (int nSlot, out byte[ ] vbATR)

  Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal connect functionality of the ICardTerminal interface.
- abstract CardActivationResult InternalReconnectCard (int nSlot, out byte[ ] vbATR)

  Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal reconnect functionality of the ICardTerminal interface.
- abstract CardResponseAPDU InternalSendCommand (int nSlot, CardCommandAPDU aCommand)

  Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal send command functionality of the ICardTerminal interface.
- abstract CardTransactionResult InternalBeginCardTransaction ()

  Abstract method that must be overritten by a concrete card terminal implementation in order to provide this card transaction functionality of the ICardTerminal interface.
- abstract CardTransactionResult InternalEndCardTransaction ()

  Abstract method that must be overritten by a concrete card terminal implementation in order to provide this card transaction functionality of the ICardTerminal interface.
- abstract void InternalDisconnectCard (int nSlot)

  Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal power down functionality of the ICardTerminal interface.
- abstract void InternalDeliverCard (int nSlot, bool fCapture)

  Abstract method that can be overriden by a concrete card terminal implementation in order to provide the card terminal eject functionality of the ICardTerminal interface.

Protected Attributes

- bool m_fDisposed = false

  This flag is checked by all public methods of this base class. If this is true, then an ObjectDisposedException is thrown, without any further action. Any additional public method in a derived class shall check this flag, too, and act similar.
- bool m_fIsInitialised = false

  This flag is set to true when the Init method was successfully called. Specialised constructors in derived classes must set this flag to true in order to indicate that the class can actually be used.
- CardTerminalConfig m_aConfig = null

  Holds the card terminal configuration of this card terminal.
- string m_sInternalName = null

  A derived implementation must set the internal name string in its constructor.
- string m_sManufacturerName = null

  A derived implementation must set the manufacturer name string in its constructor.
- string m_sProductName = null

  A derived implementation must set the product name string in its constructor.
- string m_sProductVersion = null

  A derived implementation must set the product version string in its constructor.
- string m_sProductAdditionalInfo = null

A derived implementation must set the product additional info string in its constructor.

- string m_sPhysicalConnection = null

  A derived implementation must set the physical connection string in its constructor.
- string m_sProductSerialNumber = null

  A derived implementation must set the serial number string in its constructor. 17NOV2011 MJ add m_sProductSerial↩ Number to accommodate terminal serial number.
- CardTerminalFunctionalUnits m_nFunctionalUnits

  If a derived card terminal implementation supports any additional function units, it must set these flags for the supported functional units in its constructor. By default this value is CardTerminalFunctionalUnits.None.
- int m_nSlots = 1

  A derived implementation must initialize the number of slots of the card terminal in its constructor. If it doesn't, the default value 1 will be used.
- int m_nUserSlots = 1

  A derived implementation must initialize the number of user accessible slots of the card terminal in its constructor. If it doesn't, the default value 1 will be used.
- CardTypes m_nCardType

  Card type that this card terminal is currently connected to.

Properties

- CardTerminalConfig Config `[get]`

  Boilerplate implementation of ICardTerminal.Config that simply returns the value of the protected member m_aConfig.
- string InternalName `[get]`

  Boilerplate implementation of ICardTerminal.InternalName that simply returns the value of the protected member m_s↩ InternalName.
- string ManufacturerName `[get]`

  Boilerplate implementation of ICardTerminal.ManufacturerName that simply returns the value of the protected member m_sManufacturerName.
- string ProductName `[get]`

  Boilerplate implementation of ICardTerminal.ProductName that simply returns the value of the protected member m_s↩ ProductName.
- string ProductVersion `[get]`

  Boilerplate implementation of ICardTerminal.ProductVersion that simply returns the value of the protected member m_↩ sProductVersion.
- string ProductAdditionalInfo `[get]`

  Boilerplate implementation of ICardTerminal.ProductAdditionalInfo that simply returns the value of the protected member m_sProductAdditionalInfo.
- string ProductSerialNumber `[get]`

  Boilerplate implementation of ICardTerminal.ProductSerialNumber that simply returns the value of the protected member m_sProductSerialNumber.
- string PhysicalConnection `[get]`

  Boilerplate implementation of ICardTerminal.PhysicalConnection that simply returns the value of the protected member m_sPhysicalConnection.
- CardTerminalFunctionalUnits FunctionalUnits `[get]`

  Boilerplate implementation of ICardTerminal.FunctionalUnits that simply returns the value of the protected member m_↩ nFunctionalUnits.
- int Slots `[get]`

  Boilerplate implementation of ICardTerminal.Slots that simply returns the value of the protected member m_nSlots.
- int UserSlots `[get]`

  Boilerplate implementation of ICardTerminal.UserSlots that simply returns the value of the protected member m_nUser↩ Slots.

6.19.1 Detailed Description

Common base class to simplify the implementation of the ICardTerminal interface. This class is intended for card terminal manufacturers and not for application programmers.

The CardWerk SmartCard framework currently includes two implementations of the ICardTerminal interface. One for CT-API driven card terminals and another for PC/SC Workgroup API driven card terminals. Both are derived from this base class. It is recommended, that third party card terminal suppliers who want to provide their own, optimized ICardTerminal implementation derive from this class, too.

See also

T:System.IDisposable, T:Subsembly.SmartCard.ICardTerminal

6.19.2 Member Function Documentation

6.19.2.1 CardTransactionResult Subsembly.SmartCard.CardTerminalBase.BeginTransaction (int *nSlot*) [inline]

Boilerplate implementation of ICardTerminal.BeginTransaction.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was correctly initialised and throws a InvalidOperationException if not. Then the parameter nSlot is checked against m_nSlots and an ArgumentOutOfRangeException is thrown if the requested slot does not exist. Also the parameter aCommand is checked whether it is null, and a ArgumentNullException is thrown, if so.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nSlot	Slot number of card terminal slot of interest.

Returns

Returns the result of the end transaction request.

See also

M:Subsembly.SmartCard.ICardTerminal.BeginTransaction(int)

Implements Subsembly.SmartCard.ICardTerminal.

6.19.2.2 CardActivationResult Subsembly.SmartCard.CardTerminalBase.ConnectCard (int *nSlot*, CardTypes *nCardType*, out byte[ ] *vbATR*) [inline]

Boilerplate implementation of ICardTerminal.ConnectCard.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was correctly initialised and throws a InvalidOperationException if not. Then the parameter nSlot is checked against m_nSlots and an ArgumentOutOfRangeException is thrown if the requested slot does not exist.

If the preconditions have been satisfied, then the abstract method InternalConnectCard is invoked.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.
ArgumentOutOfRange←↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nSlot	Slot number of card terminal slot of interest.
nCardType	Card type to connect to. If the actual ICardTerminal implementation does not support the requested card type, then CardActivationResult.NotImplemented is returned.
vbATR	[out] Returns the complete ATR that was received when the card was successfully powered up. Will be null whenever the card was not successfully powered up.

Returns

Returns the result from the attempt to activate the card. See CardActivationResult for possible return values.

See also

M:Subsembly.SmartCard.ICardTerminal.ConnectCard(int,CardTypes,out byte[ ])

Implements Subsembly.SmartCard.ICardTerminal.

6.19.2.3 void Subsembly.SmartCard.CardTerminalBase.DeliverCard (int *nSlot*, bool *fCapture*) [inline]

Boilerplate implementation of ICardTerminal.DeliverCard.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was correctly initialized and throws a InvalidOperationException if not. Then the parameter nSlot is checked against m_nSlots and an ArgumentOutOfRangeException is thrown if the requested slot does not exist.

If the preconditions have been satisfied, then the abstract method InternalDeliverCard is invoked.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.
ArgumentOutOfRange←↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nSlot	Slot number of card terminal slot of interest.
fCapture	If true, then the card shall be confiscated, if possible.

See also

M:Subsembly.SmartCard.ICardTerminal.DeliverCard(int,bool)

Implements Subsembly.SmartCard.ICardTerminal.

6.19.2.4 void Subsembly.SmartCard.CardTerminalBase.DisconnectCard (int *nSlot*) [inline]

Boilerplate implementation of ICardTerminal.DisconnectCard.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was correctly initialised and throws a InvalidOperationException if not. Then the parameter nSlot is checked against m_nSlots and an ArgumentOutOfRangeException is thrown if the requested slot does not exist.

If the preconditions have been satisfied, then the abstract method InternalDisconnectCard is invoked.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.
ArgumentOutOfRange←Exception	Thrown when one or more arguments are outside the required range.

Parameters

nSlot	Slot number of card terminal slot of interest.

See also

M:Subsembly.SmartCard.ICardTerminal.DisconnectCard(int)

Implements Subsembly.SmartCard.ICardTerminal.

6.19.2.5 void Subsembly.SmartCard.CardTerminalBase.Dispose ( ) `[inline]`

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

See also

M:System.IDisposable.Dispose()

6.19.2.6 virtual void Subsembly.SmartCard.CardTerminalBase.Dispose (bool *fDisposing*) `[inline]`, `[protected]`, `[virtual]`

Must be overridden by derived classes in order to implement the Dispose pattern.

Parameters

fDisposing	If true, then this method is invoked due to an invocation of the public IDispose.Dispose method. If false, then this method is invoked from the destructor (finalizer). In the latter case only perform cleanup on data stored in your object's own fields, and do not perform any actions on referenced objects or static fields.

A derived implementation should follow the following C# example when implementing its Dispose method.

```
protected override void Dispose(bool fDisposing)
{
    if (!m_fDisposed)
    {
        try
        {
            if (fDisposing)
            {
                // Clean up all unmanaged resources referenced from this class.
                // For example, here is the place to orderly release handles to a
                // native card terminal resource.
            }
        }
        finally
        {
            // This finally clause ensures that this base class is also cleaned up
            // properly
            base.Dispose(fDisposing);
        }
    }
}
```

Reimplemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.7 CardTransactionResult Subsembly.SmartCard.CardTerminalBase.EndTransaction (int *nSlot*) [inline]

Boilerplate implementation of ICardTerminal.EndTransaction.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was correctly initialised and throws a InvalidOperationException if not. Then the parameter nSlot is checked against m_nSlots and an ArgumentOutOfRangeException is thrown if the requested slot does not exist. Also the parameter aCommand is checked whether it is null, and a ArgumentNullException is thrown, if so.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nSlot	Slot number of card terminal slot of interest.

Returns

Returns the result of the end transaction request.

See also

M:Subsembly.SmartCard.ICardTerminal.EndTransaction(int)

Implements Subsembly.SmartCard.ICardTerminal.

6.19.2.8 void Subsembly.SmartCard.CardTerminalBase.Init (CardTerminalConfig *aConfig*) [inline]

Boilerplate implementation of ICardTerminal.Init.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was already initialized and throws a InvalidOperationException if so. Next it invokes the abstract method InternalInit which must be provided by the derived class. If that internal method returns without throwing an exception the internal flag m_fIsInitialised is set to indicate successful initialization.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

aConfig	Provides the complete configuration information. This is never null.

See also

M:Subsembly.SmartCard.ICardTerminal.Init(CardTerminalConfig)

Implements Subsembly.SmartCard.ICardTerminal.

6.19.2.9 abstract CardTransactionResult Subsembly.SmartCard.CardTerminalBase.InternalBeginCardTransaction ( ) `[protected], [pure virtual]`

Abstract method that must be overritten by a concrete card terminal implementation in order to provide this card transaction functionality of the ICardTerminal interface.

This base class already ensured that the object was not already disposed, was correctly initialised, that the slot number is within the range of available slots.

Returns

Returns the card transaction response.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.10 abstract CardActivationResult Subsembly.SmartCard.CardTerminalBase.InternalConnectCard (int *nSlot*, out byte[ ] *vbATR*) `[protected], [pure virtual]`

Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal connect functionality of the ICardTerminal interface.

This base class already ensured that the object was not already disposed, was correctly initialised andthat the slot number is within the range of available slots before invoking InternalConnectCard.

Parameters

nSlot	Slot number of card terminal slot of interest.
vbATR	[out] Returns the complete ATR that was received when the card was successfully powered up. Will be null whenever the card was not successfully powered up.

Returns

Returns the result from the attempt to activate the card. See CardActivationResult for possible return values.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.11 abstract void Subsembly.SmartCard.CardTerminalBase.InternalDeliverCard (int *nSlot*, bool *fCapture*) `[protected], [pure virtual]`

Abstract method that can be overriden by a concrete card terminal implementation in order to provide the card terminal eject functionality of the ICardTerminal interface.

This base class already ensured that the object was not already disposed, was correctly initialised and that the slot number is within the range of available slots before invoking InternalDeliverCard.

Parameters

nSlot	Slot number of card terminal slot of interest.
fCapture	If true, then the card shall be confiscated, if possible.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.12 abstract void Subsembly.SmartCard.CardTerminalBase.InternalDisconnectCard (int *nSlot*) [protected], [pure virtual]

Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal power down functionality of the ICardTerminal interface.

This base class already ensured that the object was not already disposed, was correctly initialised and that the slot number is within the range of available slots before invoking InternalDisconnectCard.

Overriding this method is optional. If it is not overridden then the no-op default method provided by this base class will be used. This is suitable for card terminals that do not support an eject mechanism.

Parameters

nSlot	Slot number of card terminal slot of interest.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.13 abstract CardTransactionResult Subsembly.SmartCard.CardTerminalBase.InternalEndCardTransaction ( ) [protected], [pure virtual]

Abstract method that must be overritten by a concrete card terminal implementation in order to provide this card transaction functionality of the ICardTerminal interface.

This base class already ensured that the object was not already disposed, was correctly initialised, that the slot number is within the range of available slots.

Returns

Returns the card transaction response.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.14 abstract void Subsembly.SmartCard.CardTerminalBase.InternalInit (string *sXmlConfig*) [protected], [pure virtual]

Abstract method that must be overriden by a concrete card terminal implementation in order to initialise the card terminal.

This base class already ensured that the object was not already disposed and was not already initialised before invoking InternalResetCardTerminal.

Parameters

sXmlConfig	

Provides the specific configuration information.

String that contains XML data with configuration information for initialising this instance of the card terminal. The XML data contains a < Config> root node with card terminal specific child nodes.

If there is no configuration information, then this may be null.

Usually the configuration information of a card terminal is stored in the CardTerminalRegistry.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.15 abstract bool Subsembly.SmartCard.CardTerminalBase.InternalIsCardPresent (int *nSlot*, ref bool *requiresRemoval*) [protected], [pure virtual]

Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal card presence detection functionality of the ICardTerminal interface.

This base class already ensured that the object was not already disposed, was correctly initialised and that the slot number is within the range of available slots before invoking InternalIsCardPresent.

Parameters

nSlot	Slot number of card terminal slot of interest.
requiresRemoval	[in,out] Flag to indicate that user must remove card from slot.

Returns

Returns true if a card is present in the indicated slot and false if not.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.16 abstract CardActivationResult Subsembly.SmartCard.CardTerminalBase.InternalReconnectCard (int *nSlot*, out byte[ ] *vbATR*) [protected], [pure virtual]

Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal reconnect functionality of the ICardTerminal interface.

This base class already ensured that the object was not already disposed, was correctly initialised and that the slot number is within the range of available slots before invoking InternalReconnectCard.

Parameters

nSlot	Slot number of card terminal slot of interest.
vbATR	Card ATR after reconnect (warm ATR)

Returns

Returns the result from the attempt to activate the card. See CardActivationResult for possible return values.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.17 abstract void Subsembly.SmartCard.CardTerminalBase.InternalResetCardTerminal ( ) [protected], [pure virtual]

Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal reset functionality of the ICardTerminal interface.

This base class already ensured that the object was not already disposed and was correctly initialised before invoking InternalResetCardTerminal.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.18 abstract CardResponseAPDU Subsembly.SmartCard.CardTerminalBase.InternalSendCommand (int *nSlot*, CardCommandAPDU *aCommand*) [protected], [pure virtual]

Abstract method that must be overriden by a concrete card terminal implementation in order to provide the card terminal send command functionality of the ICardTerminal interface.

This base class already ensured that the object was not already disposed, was correctly initialised, that the slot number is within the range of available slots, and that aCommand is not null before invoking InternalSendCommand.

Parameters

nSlot	Slot number of card terminal slot of interest.
aCommand	Complete command APDU to be transmitted.

Returns

Returns the response APDU from the inserted card.

Implemented in Subsembly.SmartCard.CardPcScTerminal.

6.19.2.19 bool Subsembly.SmartCard.CardTerminalBase.IsCardPresent (int *nSlot*, ref bool *requiresRemoval*) [inline]

Boilerplate implementation of ICardTerminal.IsCardPresent.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was correctly initialised and throws a InvalidOperationException if not. Then the parameter nSlot is checked against m_nSlots and an ArgumentOutOfRangeException is thrown if the requested slot does not exist.

If the preconditions have been satisfied, then the abstract method InternalIsCardPresent is invoked.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nSlot	Slot number of card terminal slot of interest.
requiresRemoval	[in,out] Returns a flag telling the caller if the card must be removed by the user This is required for motorized readers.

Returns

Returns true if a card is present in the indicated slot and false if not.

See also

M:Subsembly.SmartCard.ICardTerminal.IsCardPresent(int,bool@)

Implements Subsembly.SmartCard.ICardTerminal.

6.19.2.20 CardActivationResult Subsembly.SmartCard.CardTerminalBase.ReconnectCard (int *nSlot*, out byte[ ] *vbATR*) [inline]

Boilerplate implementation of ICardTerminal.ReconnectCard.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was correctly initialised and throws a InvalidOperationException if not. Then the parameter nSlot is checked against m_nSlots and an ArgumentOutOfRangeException is thrown if the requested slot does not exist.

If the preconditions have been satisfied, then the abstract method InternalReconnectCard is invoked.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

nSlot	Slot number of card terminal slot of interest.
vbATR	Card ATR.

Returns

Returns the result from the attempt to activate the card. See CardActivationResult for possible return values.

See also

M:Subsembly.SmartCard.ICardTerminal.ReconnectCard(int)

Implements Subsembly.SmartCard.ICardTerminal.

6.19.2.21 void Subsembly.SmartCard.CardTerminalBase.ResetCardTerminal ( ) [inline]

Boilerplate implementation of ICardTerminal.ResetCardTerminal.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was correctly initialized and throws a InvalidOperationException if not. Finally it invokes the abstract method InternalResetCardTerminal which must be provided by the derived class.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.

See also

M:Subsembly.SmartCard.ICardTerminal.ResetCardTerminal()

Implements Subsembly.SmartCard.ICardTerminal.

6.19.2.22 CardResponseAPDU Subsembly.SmartCard.CardTerminalBase.SendCommand (int *nSlot*, CardCommandAPDU *aCommand*) [inline]

Boilerplate implementation of ICardTerminal.SendCommand.

This implementation checks whether this object was disposed and throws an ObjectDisposedException if so. Also it checks whether the object was correctly initialised and throws a InvalidOperationException if not. Then the parameter nSlot is checked against m_nSlots and an ArgumentOutOfRangeException is thrown if the requested slot does not exist. Also the parameter aCommand is checked whether it is null, and a ArgumentNullException is thrown, if so.

If the preconditions have been satisfied, then the command APDU is traced by invoking the trace feature CardTerminal↵ Manager singleton. Then the abstract method InternalSendCommand is invoked. Finally the response APDU is traced by invoking the trace feature CardTerminalManager singleton one more time.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.
ArgumentOutOfRange↵ Exception	Thrown when one or more arguments are outside the required range.
ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

nSlot	Slot number of card terminal slot of interest.
aCommand	Complete command APDU to be transmitted.

Returns

Returns the response APDU from the inserted card.

See also

M:Subsembly.SmartCard.ICardTerminal.SendCommand(int,CardCommandAPDU)

Implements Subsembly.SmartCard.ICardTerminal.

6.19.3 Member Data Documentation

6.19.3.1 CardTerminalConfig Subsembly.SmartCard.CardTerminalBase.m_aConfig = null [protected]

Holds the card terminal configuration of this card terminal.

6.19.3.2 CardTypes Subsembly.SmartCard.CardTerminalBase.m_nCardType [protected]

Card type that this card terminal is currently connected to.

6.19.3.3 CardTerminalFunctionalUnits Subsembly.SmartCard.CardTerminalBase.m_nFunctionalUnits [protected]

Initial value:

```
=
            CardTerminalFunctionalUnits.None
```

If a derived card terminal implementation supports any additional function units, it must set these flags for the supported functional units in its constructor. By default this value is CardTerminalFunctionalUnits.None.

6.19.4 Property Documentation

6.19.4.1 CardTerminalConfig Subsembly.SmartCard.CardTerminalBase.Config [get]

Boilerplate implementation of ICardTerminal.Config that simply returns the value of the protected member m_aConfig.

The configuration.

See also

P:Subsembly.SmartCard.ICardTerminal.Config

6.19.4.2 CardTerminalFunctionalUnits Subsembly.SmartCard.CardTerminalBase.FunctionalUnits [get]

Boilerplate implementation of ICardTerminal.FunctionalUnits that simply returns the value of the protected member m↵ _nFunctionalUnits.

The functional units.

See also

P:Subsembly.SmartCard.ICardTerminal.FunctionalUnits

6.19.4.3 string Subsembly.SmartCard.CardTerminalBase.InternalName [get]

Boilerplate implementation of ICardTerminal.InternalName that simply returns the value of the protected member m_↵ sInternalName.

The name of the internal.

See also

P:Subsembly.SmartCard.ICardTerminal.InternalName

6.19.4.4 string Subsembly.SmartCard.CardTerminalBase.ManufacturerName [get]

Boilerplate implementation of ICardTerminal.ManufacturerName that simply returns the value of the protected member m_sManufacturerName.

The name of the manufacturer.

See also

P:Subsembly.SmartCard.ICardTerminal.ManufacturerName

6.19.4.5 string Subsembly.SmartCard.CardTerminalBase.PhysicalConnection [get]

Boilerplate implementation of ICardTerminal.PhysicalConnection that simply returns the value of the protected member m_sPhysicalConnection.

The physical connection.

See also

P:Subsembly.SmartCard.ICardTerminal.PhysicalConnection

6.19.4.6 string Subsembly.SmartCard.CardTerminalBase.ProductAdditionalInfo [get]

Boilerplate implementation of ICardTerminal.ProductAdditionalInfo that simply returns the value of the protected member m_sProductAdditionalInfo.

Information describing the product additional.

See also

P:Subsembly.SmartCard.ICardTerminal.ProductAdditionalInfo

6.19.4.7 string Subsembly.SmartCard.CardTerminalBase.ProductName [get]

Boilerplate implementation of ICardTerminal.ProductName that simply returns the value of the protected member m_←sProductName.

The name of the product.

See also

P:Subsembly.SmartCard.ICardTerminal.ProductName

6.19.4.8 string Subsembly.SmartCard.CardTerminalBase.ProductSerialNumber [get]

Boilerplate implementation of ICardTerminal.ProductSerialNumber that simply returns the value of the protected member m_sProductSerialNumber.

The product serial number.

See also

P:Subsembly.SmartCard.ICardTerminal.ProductSerialNumber

6.19.4.9 string Subsembly.SmartCard.CardTerminalBase.ProductVersion [get]

Boilerplate implementation of ICardTerminal.ProductVersion that simply returns the value of the protected member m←_sProductVersion.

The product version.

See also

P:Subsembly.SmartCard.ICardTerminal.ProductVersion

6.19.4.10 int Subsembly.SmartCard.CardTerminalBase.Slots [get]

Boilerplate implementation of ICardTerminal.Slots that simply returns the value of the protected member m_nSlots.

The slots.

See also

P:Subsembly.SmartCard.ICardTerminal.Slots

6.19.4.11 int Subsembly.SmartCard.CardTerminalBase.UserSlots `[get]`

Boilerplate implementation of ICardTerminal.UserSlots that simply returns the value of the protected member m_n↩ UserSlots.

The user slots.

See also

P:Subsembly.SmartCard.ICardTerminal.UserSlots

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardTerminal↩ Base.cs

6.20 Subsembly.SmartCard.CardTerminalConfig Class Reference

Collects information about a card terminal configuration that is stored in the CardTerminalRegistry

Public Member Functions

- CardTerminalConfig (string sAssignedName)

    Constructor.
- ICardTerminal Create ()

    Instantiates and initializes a card terminal from this card terminal configuration.
- override bool Equals (object rhs)

    Determines whether the specified object is equal to the current object.
- override int GetHashCode ()

    The hash code includes only the AssignedName.

Public Attributes

- const string CTAPICARDTERMINALCLASSNAME

    Constant that provides the fully qualified name of the default CT-API based ICardTerminal implementation.
- const string PCSCCARDTERMINALCLASSNAME

    Constant that provides the fully qualified name of the default PC/SC based ICardTerminal implementation.
- const string MCARDCARDTERMINALCLASSNAME

    Constant that provides the fully qualified name of the default MCard API based ICardTerminal implementation.

Properties

- string AssignedName `[get]`

    Name that was assigned to this card terminal configuration.
- string AssemblyName `[get, set]`

    The assembly name of the assembly that contains the class that implements the ICardTerminal interface for this card terminal. This assembly will be loaded in order to instantiate ClassName when it is started up.
- string ClassName `[get, set]`

The fully qualified class name of the class that implements the ICardTerminal interface for this card terminal. This class will be instantiated by the CardTerminalManager when it is started up.

- CardTerminalType Type `[get]`

  Indicates which type of standard card terminal this is. If it is no standard type at all, then CardTerminalType.UNKNOWN is returned.
- bool Enabled `[get, set]`

  Controls whether this card terminal shall be actually employed by the CardTerminalManager or not.
- bool UseSecurePin `[get, set]`

  Controls whether the class 2 functionality shall be used or not. Defaults to true, i.e. class 2 functionality will be used if available.
- string Config `[get, set]`

  XML configuration data passed to the ICardTerminal interface when the card terminal is initialized. May be null if there is no configuration data.
- string ErrorInfo `[get, set]`

  Contains human readable error information about this configuration.

6.20.1 Detailed Description

Collects information about a card terminal configuration that is stored in the CardTerminalRegistry

Each card terminal configuration is uniquely identified by its AssignedName. As its name suggests, the AssignedName can be arbitrarily assigned by the user using the card terminal configurator tool. It is not possible to register multiple card terminals having the same AssignedName property value with the CardTerminalRegistry.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 Subsembly.SmartCard.CardTerminalConfig.CardTerminalConfig (string *sAssignedName*) `[inline]`

Constructor.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

sAssignedName	Name of the assigned.

6.20.3 Member Function Documentation

6.20.3.1 ICardTerminal Subsembly.SmartCard.CardTerminalConfig.Create ( ) `[inline]`

Instantiates and initializes a card terminal from this card terminal configuration.

If the configured card terminal failed and null is returned, then the ErrorInfo property contains a descriptive string about the nature of the error. If it succeeded, then the ErrorInfo property is set to null.

Returns

The successfully instantiated and initialized card terminal is returned. If this card terminal configuration is disabled or an error occurs, then null is returned.

6.20.3.2 override bool Subsembly.SmartCard.CardTerminalConfig.Equals (object *rhs*) [inline]

Determines whether the specified object is equal to the current object.

Parameters

rhs	The object to compare with the current object.

Returns

true if the specified object is equal to the current object; otherwise, false.

See also

M:System.Object.Equals(object)

6.20.3.3 override int Subsembly.SmartCard.CardTerminalConfig.GetHashCode ( ) [inline]

The hash code includes only the AssignedName.

Returns

A hash code for this object.

See also

M:System.Object.GetHashCode()

6.20.4 Member Data Documentation

6.20.4.1 const string Subsembly.SmartCard.CardTerminalConfig.CTAPICARDTERMINALCLASSNAME

Initial value:

```
=
            "Subsembly.SmartCard.CardCtApiTerminal"
```

Constant that provides the fully qualified name of the default CT-API based ICardTerminal implementation.

6.20.4.2 const string Subsembly.SmartCard.CardTerminalConfig.MCARDCARDTERMINALCLASSNAME

Initial value:

```
=
            "Subsembly.SmartCard.MCard.MCardCardTerminal"
```

Constant that provides the fully qualified name of the default MCard API based ICardTerminal implementation.

6.20.4.3 const string Subsembly.SmartCard.CardTerminalConfig.PCSCCARDTERMINALCLASSNAME

Initial value:

```
=
        "Subsembly.SmartCard.CardPcScTerminal"
```

Constant that provides the fully qualified name of the default PC/SC based ICardTerminal implementation.

6.20.5 Property Documentation

6.20.5.1 string Subsembly.SmartCard.CardTerminalConfig.AssemblyName [get], [set]

The assembly name of the assembly that contains the class that implements the ICardTerminal interface for this card terminal. This assembly will be loaded in order to instantiate ClassName when it is started up.

If this is null, then the currently executing assembly will be used and no assembly will be loaded. This works for the built-in card terminal implementations, but probably won't work for external card terminal implementations.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

The name of the assembly.

6.20.5.2 string Subsembly.SmartCard.CardTerminalConfig.AssignedName [get]

Name that was assigned to this card terminal configuration.

The name of the assigned.

6.20.5.3 string Subsembly.SmartCard.CardTerminalConfig.ClassName [get], [set]

The fully qualified class name of the class that implements the ICardTerminal interface for this card terminal. This class will be instantiated by the CardTerminalManager when it is started up.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

The name of the class.

6.20.5.4 string Subsembly.SmartCard.CardTerminalConfig.Config [get], [set]

XML configuration data passed to the ICardTerminal interface when the card terminal is initialized. May be null if there is no configuration data.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

The configuration.

6.20.5.5 bool Subsembly.SmartCard.CardTerminalConfig.Enabled `[get], [set]`

Controls whether this card terminal shall be actually employed by the CardTerminalManager or not.

This flag can be used to disable PC/SC Workgroup card terminals that are automatically registered during startup.

True if enabled, false if not.

6.20.5.6 string Subsembly.SmartCard.CardTerminalConfig.ErrorInfo `[get], [set]`

Contains human readable error information about this configuration.

Exceptions

ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

If an attempt to instantiate and initialise the card terminal according to this configuration, then an error description about the failure shall be stored here. This error info will be shown to the user in the card terminal configurator. If this is null, then no error occured or no attempt to use that card terminal was made, yet.

6.20.5.7 CardTerminalType Subsembly.SmartCard.CardTerminalConfig.Type `[get]`

Indicates which type of standard card terminal this is. If it is no standard type at all, then CardTerminalType.UNKNOWN is returned.

The type.

6.20.5.8 bool Subsembly.SmartCard.CardTerminalConfig.UseSecurePin `[get], [set]`

Controls whether the class 2 functionality shall be used or not. Defaults to true, i.e. class 2 functionality will be used if available.

True if use secure pin, false if not.

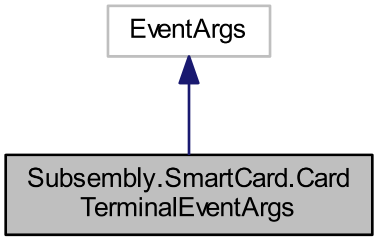
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardTerminal↩ Config.cs

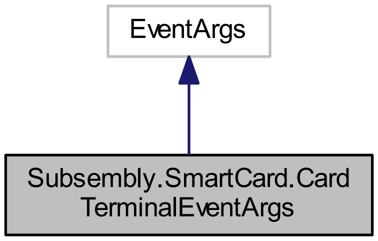
6.21 Subsembly.SmartCard.CardTerminalEventArgs Class Reference

This class provides the event information for the card terminal events that are raised by the CardTerminalManager. Events can be triggered by card insertion, removal as well as a reader's request for card removal (from a the lip of a motorized reader or from an RF field).

Inheritance diagram for Subsembly.SmartCard.CardTerminalEventArgs:



Collaboration diagram for Subsembly.SmartCard.CardTerminalEventArgs:



Properties

- CardTerminalSlot Slot `[get]`

  Reference to the CardTerminalSlot instance where the event came from.

6.21.1 Detailed Description

This class provides the event information for the card terminal events that are raised by the CardTerminalManager. Events can be triggered by card insertion, removal as well as a reader's request for card removal (from a the lip of a motorized reader or from an RF field).

See also

T:System.EventArgs

6.21.2 Property Documentation

6.21.2.1 CardTerminalSlot Subsembly.SmartCard.CardTerminalEventArgs.Slot `[get]`

Reference to the CardTerminalSlot instance where the event came from.

The slot.

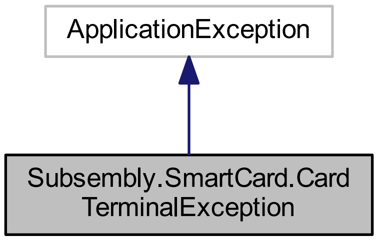
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardTerminal↩ Events.cs

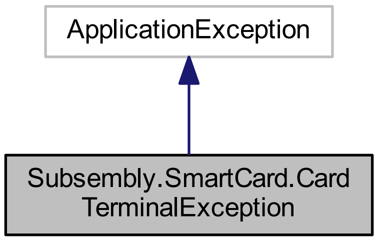
6.22 Subsembly.SmartCard.CardTerminalException Class Reference

Whenever an error occurs an exception of this class is thrown.

Inheritance diagram for Subsembly.SmartCard.CardTerminalException:



Collaboration diagram for Subsembly.SmartCard.CardTerminalException:



Public Member Functions

- CardTerminalException (CardTerminalExceptionCode nCode)

  Constructor.
- CardTerminalException (CardTerminalExceptionCode nCode, string sMessage)

  Constructor.
- CardTerminalException (CardTerminalExceptionCode nCode, Exception aInnerException)

  Constructor.

Properties

- CardTerminalExceptionCode Code `[get]`

  Access the card terminal exception code that further distinguishes the cause of this exception.

6.22.1 Detailed Description

Whenever an error occurs an exception of this class is thrown.

A CardTerminalException may provide an InnerException that references the low level exception thrown from the internals of an implementation of the ICardTerminal interface. Currently there are only two implementations of the ICard←Terminal interface propagating either a SCardException or a CtApiException.

See also

T:System.ApplicationException

6.22.2 Constructor & Destructor Documentation

6.22.2.1 Subsembly.SmartCard.CardTerminalException.CardTerminalException (CardTerminalExceptionCode *nCode*) `[inline]`

Constructor.

Parameters

nCode	.

6.22.2.2 Subsembly.SmartCard.CardTerminalException.CardTerminalException (CardTerminalExceptionCode *nCode*, string *sMessage*) `[inline]`

Constructor.

Parameters

nCode	.
sMessage	.

6.22.2.3 Subsembly.SmartCard.CardTerminalException.CardTerminalException (CardTerminalExceptionCode *nCode*, Exception *aInnerException*) `[inline]`

Constructor.

Parameters

nCode	.
aInnerException	The inner exception.

6.22.3 Property Documentation

6.22.3.1 CardTerminalExceptionCode Subsembly.SmartCard.CardTerminalException.Code `[get]`

Access the card terminal exception code that further distinguishes the cause of this exception.

The code.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardTerminal←
Exception.cs

6.23 Subsembly.SmartCard.CardTerminalManager Class Reference

CardTerminalManager class is a singleton that is the one and only instance that manages card terminals and tracks card insertions and removals.

Public Member Functions

- CardTerminalSlot[ ] GetSlots ()

  Returns an array with references to the monitored card terminal slots.
- string[ ] GetSlotNames ()

  Returns an array with all names of all monitored card terminal slots.
- void EnlistCardTerminal (ICardTerminal aCardTerminal)

  Adds a card terminal to the list of monitored card terminals.
- void DelistCardTerminal (ICardTerminal aCardTerminal)

  Removes a card terminal from the list of monitored card terminals.
- ICardTerminal CreateCardTerminal (CardTerminalConfig aConfig)

  Instantiates and initialises a card terminal from a given card terminal configuration.
- int Startup (bool fAutoRegisterPcSc)

  Starts CardTerminalManager.
- int Startup (bool fAutoRegisterPcSc, bool fLocalUser)

  Completely Starts up the card terminal manager adding all registered and enabled card terminals to the monitored card terminals list.
- void Shutdown ()

  Completely shuts down the card terminal manager, delisting and disposing all monitored card terminals.

Properties

- static CardTerminalManager Singleton `[get]`

  Use this static property to get access to the one and only CardTerminalManager singleton.
- bool StartedUp `[get]`

Indicates whether the card terminal manager was started up by Startup(bool,bool).

- CardTerminalRegistry Registry `[get]`

  The CardTerminalRegistry that is used by this CardTerminalManager.
- int PollIntervall `[get, set]`

  Polling interval in milliseconds. Every PollIntervall milliseconds the card position of the primary slot of the monitored card terminals are polled.
- int SlotCount `[get]`

  Number of monitored card terminal slots.
- int SlotCountMinimum `[get, set]`

  Minimum of card terminal slots.

Events

- CardTerminalEventHandler CardInsertedEvent

  Event that will be raised whenever a card is inserted in any one of the monitored card terminals primary slots.
- CardTerminalEventHandler CardRemovedEvent

  Event that will be raised whenever a card is removed from any one of the monitored card terminals primary slots.
- CardTerminalEventHandler CardRequiresRemovalEvent

  Event that will be raised whenever a card must be removed from any of the monitored card terminal's primary slot.
- CardTerminalEventHandler CardTerminalFoundEvent

  Event that will be raised whenever the a card terminal (slot) gets found. Example: user connects an additional reader after the singleton already started.
- CardTerminalEventHandler CardTerminalLostEvent

  Event that will be raised whenever the card terminal connection gets lost. Example: PC/SC smart card service stopped after terminal startup.

6.23.1 Detailed Description

CardTerminalManager class is a singleton that is the one and only instance that manages card terminals and tracks card insertions and removals.

There is always only a single instance of CardTerminalManager class, hence it is called a singleton. Applications must use the public Singleton property in order to get access to the card terminal manager.

The card terminal manager provides events for catching card terminal events such as card insertion or removal. The card terminal manager runs a background thread that polls the card insertion state of all monitored card terminal slots and raises events whenever a card is inserted or removed.

Initially the CardTerminalManager manages no card terminals at all. An application must call Startup(bool,bool) in order to automatically enlist all registered card terminals.

See also

CardTerminalRegistry

6.23.2 Member Function Documentation

6.23.2.1 ICardTerminal Subsembly.SmartCard.CardTerminalManager.CreateCardTerminal (CardTerminalConfig *aConfig*) `[inline]`

Instantiates and initialises a card terminal from a given card terminal configuration.

If the configured card terminal failed and null is returned, then the CardTerminalConfig.ErrorInfo property contains a descriptive string about the nature of the error. If it succeeded, then the CardTerminalConfig.ErrorInfo property is set to null.

Parameters

aConfig	The card terminal configuration that defines and configures the ICardTerminal implementation to be created.

Returns

The successfully instantiated and initialised card terminal is returned. If the provided card terminal configuration is disabled or an error occurs, then null is returned.

See also

CardTerminalConfig.Create

6.23.2.2 void Subsembly.SmartCard.CardTerminalManager.DelistCardTerminal (ICardTerminal *aCardTerminal*) [inline]

Removes a card terminal from the list of monitored card terminals.

Delisting a card terminal returns the ownership of the ICardTerminal reference to the calling application. This implies that the calling application is now responsible for orderly disposing the ICardTerminal again.

When a card terminal is delisted all its card slots are removed from the list of monitored card terminal slots. No more events will be raised when a card is inserted or removed.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
InvalidOperationException	Thrown when the requested operation is invalid.

Parameters

aCardTerminal	Card terminal to be removed.

6.23.2.3 void Subsembly.SmartCard.CardTerminalManager.EnlistCardTerminal (ICardTerminal *aCardTerminal*) [inline]

Adds a card terminal to the list of monitored card terminals.

Adding a card terminal to the monitor list passes the ownership of the ICardTerminal reference to the CardTerminal↩ Manager. This implies that the card terminal must not be disposed by the application, but is automatically disposed when the CardTerminalManager shuts down. An application can revoke ownership by delisting the card terminal with DelistCardTerminal

When a card terminal is enlisted all its card slots are added to the list of monitored card terminal slots. Once a card terminal slot is monitored, it will be periodically polled by a background thread for changes in the card presence state.

Exceptions

ArgumentNullException	The argument was null.

InvalidOperationException	The given card terminal was already enlisted.

Parameters

aCardTerminal	Card terminal which shall be enlisted at the CardTerminalManager singleton.

See also

Shutdown, DelistCardTerminal

6.23.2.4 string [ ] Subsembly.SmartCard.CardTerminalManager.GetSlotNames ( ) `[inline]`

Returns an array with all names of all monitored card terminal slots.

Returns

An array of string.

6.23.2.5 CardTerminalSlot [ ] Subsembly.SmartCard.CardTerminalManager.GetSlots ( ) `[inline]`

Returns an array with references to the monitored card terminal slots.

Returns

An array of card terminal slot.

6.23.2.6 void Subsembly.SmartCard.CardTerminalManager.Shutdown ( ) `[inline]`

Completely shuts down the card terminal manager, delisting and disposing all monitored card terminals.

This method shall be called by an application as soon as it no longer needs any access to the managed card terminals. This method disposes all enlisted card terminals and stops monitoring card insertion or removal events.

Exceptions

InvalidOperationException	The card terminal manager was not successfully started up before.

See also

StartedUp

6.23.2.7 int Subsembly.SmartCard.CardTerminalManager.Startup (bool *fAutoRegisterPcSc*) `[inline]`

Starts CardTerminalManager.

Simply calls Startup(bool,bool) passing `false` for the fLocalUser parameter.

Parameters

fAutoRegister↩ PcSc	true: registers all readers currently available via PCSC; false: request to monitor only readers listed in registry file

Returns

An int.

6.23.2.8 int Subsembly.SmartCard.CardTerminalManager.Startup (bool *fAutoRegisterPcSc,* bool *fLocalUser*) [inline]

Completely Starts up the card terminal manager adding all registered and enabled card terminals to the monitored card terminals list.

This method only returns after the background polling thread has been started successfully and after it updated all the slot states for the first time around. This means, that if a card is already inserted, a CardInsertedEvent will be fired before this method returns.

Exceptions

InvalidOperationException	The card terminal manager was already started up.

Parameters

fAutoRegister↩ PcSc	If this parameter is true, then all PC/SC Workgroup card readers that are not already registered will be added to the registry. For most applications it is suggested that this parameter is passed as `true` .

Parameters

fLocalUser	If this is true, then the registry data will be stored in the non-roaming local application data. Use this setting for all interactive applications. If this is false the registry data will be stored in the shared common application data folder. Use this for non-interactive services.

Returns

Returns the number of card terminals that have been started up and added to the list of monitored card terminals. If this is zero, then no card terminals are configured, thus an application should ask the user to configure a card terminal first.

See also

Shutdown, StartedUp

6.23.3 Property Documentation

6.23.3.1 int Subsembly.SmartCard.CardTerminalManager.PollIntervall [get], [set]

Polling interval in milliseconds. Every PollIntervall milliseconds the card position of the primary slot of the monitored card terminals are polled.

Setting a new PollInterval will only have an effect on the next interval. The current waiting interval will still be completed.

The poll intervall.

6.23.3.2 CardTerminalRegistry Subsembly.SmartCard.CardTerminalManager.Registry `[get]`

The CardTerminalRegistry that is used by this CardTerminalManager.

The CardTerminalRegistry is instantiated when Startup(bool,bool) is called. Before that, this property will be `null`.

The registry.

6.23.3.3 CardTerminalManager Subsembly.SmartCard.CardTerminalManager.Singleton `[static]`, `[get]`

Use this static property to get access to the one and only CardTerminalManager singleton.

The singleton.

6.23.3.4 int Subsembly.SmartCard.CardTerminalManager.SlotCount `[get]`

Number of monitored card terminal slots.

The number of card terminals that have been started up and added to the list of monitored card terminals. If this is zero, then no card terminals are configured, thus an application should ask the user to configure a card terminal first.

6.23.3.5 int Subsembly.SmartCard.CardTerminalManager.SlotCountMinimum `[get]`, `[set]`

Minimum of card terminal slots.

Minimum number of card terminals our polling thread is looking for. Once that number is reached we stop polling for more slots. The default value is 1 or the number of terminals detected at startup - whichever is higher.

6.23.3.6 bool Subsembly.SmartCard.CardTerminalManager.StartedUp `[get]`

Indicates whether the card terminal manager was started up by Startup(bool,bool).

True if started up, false if not.

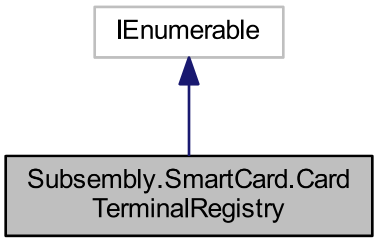
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardTerminal↩
Manager.cs

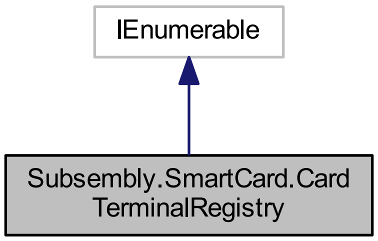
6.24 Subsembly.SmartCard.CardTerminalRegistry Class Reference

The CardTerminalRegistry class is a singleton that is the one and only instance that manages the persistent storage of card terminal configuration values of the SmartCard Subsembly in a registration file.

Inheritance diagram for Subsembly.SmartCard.CardTerminalRegistry:



Collaboration diagram for Subsembly.SmartCard.CardTerminalRegistry:



Public Member Functions

- CardTerminalRegistry (bool fLocalUser)

  Private constructor that prevents that this object be created by somebody else but itself. This guarantees the singleton pattern.
- int LoadRegistry ()

  Reloads the entire card terminal registry from its persistent registration file. .NET 4: This API is potentially dangerous; it asserts which means it cant be transparent. We assume that partial trust code can safely call this API. In fact, it is intended that it is safe for partial trust code to call this method. Therefore, it is SecuritySafeCritical.
- void SaveRegistry ()

  Rewrites the entire card terminal registry into its persistent registration file.
- IEnumerator GetEnumerator ()

  Get enumerator over the registered CardTerminalConfig entries.
- void Register (CardTerminalConfig aCtConfig)

  Add a card terminal configuration to registry (memory and persistent registry).
- void Revoke (CardTerminalConfig aCtConfig)

Removes a card terminal configuration from this registry.

- void Revoke (string sAssignedName)

    Removes a card terminal configuration from this registry.

- bool IsRegistered (CardTerminalConfig aCtConfig)

    Check whether a card terminal configuration is registered.

- bool IsRegistered (string sAssignedName)

    Check whether a card terminal configuration is registered.

- CardTerminalConfig GetRegistered (string sAssignedName)

    Retrieve a card terminal configuration from this registry.

- void AutoRegisterPcSc ()

    Updates PC/SC reader registries (internal and file-based).

- void AutoRegisterPcSc (bool updateRegistryFile)

    Updates PC/SC reader registries (internal and file-based).

- void RemoveDeadPcscEntries (bool updateRegistryFile)

    Removes the missing PC/SC readers listed in registry file.

Properties

- string RegistryFilename `[get]`

    This property provides the filename including the complete path to the SmartCard Subsembly registration file.

- int Count `[get]`

    Provides the number of card terminal configurations stored in this registry.

6.24.1 Detailed Description

The CardTerminalRegistry class is a singleton that is the one and only instance that manages the persistent storage of card terminal configuration values of the SmartCard Subsembly in a registration file.

The CardTerminalRegistry stores information about all card terminals that should be used by the SmartCard assembly. For each card terminal to be used, its name, an identification of the class implementing the ICardTerminal interface and optionally its configuration parameters (see CardTerminalConfig) are stored in the registry.

The entire registry is persistently stored in a registration file. Whenever the CardTerminalRegistry is started up it initializes itself by loading this registration file. The registration file is a simple XML file.

Each time the SmartCard API assembly is started up, it will automatically add all new PC/SC Workgroup API card readers to the the CardTerminalRegistry. An application can add other types of card terminals, such as CT-API card terminals using the Register method.

The CardTerminalRegistry is enumerable as a collection of CardTerminalConfig entries.

If, on a system without registry.xml file, the creation of an appropriate directory file is restricted, this class catches exceptions but stays quiet.

See also

T:System.Collections.IEnumerable

6.24.2 Constructor & Destructor Documentation

6.24.2.1 Subsembly.SmartCard.CardTerminalRegistry.CardTerminalRegistry (bool *fLocalUser*) `[inline]`

Private constructor that prevents that this object be created by somebody else but itself. This guarantees the singleton pattern.

Parameters

fLocalUser	If this is true, then the registry data will be stored in the non- roaming local application data. Use this setting for all interactive applications. If this is false the registry data will be stored in the shared common application data folder. Use this for non-interactive services.

6.24.3 Member Function Documentation

6.24.3.1 void Subsembly.SmartCard.CardTerminalRegistry.AutoRegisterPcSc ( ) [inline]

Updates PC/SC reader registries (internal and file-based).

This method consults the PC/SC SCardResourceManager in order to list all available card readers. Then it compares this list with the registry and adds all PC/SC card readers that are not already registered to this registry.

Any PC/SC card readers that are registered, but that were not returned by the SCardResourceManager are purged from the registry.We save the registry if there is a valid file name. The registry file name is created during instantiation of CardTerminal. This additional check allows registerless CardTerminal (fallback for systems that don't allow access). We intend to support systems such as Wyse S10 ThinOS with RDC sessions into Windows 2008 server this way fixing issues reported in OCT2012 by a customer.

If the registry was changed, then the entire registration file is rewritten. Thus I/O exceptions may occur.

6.24.3.2 void Subsembly.SmartCard.CardTerminalRegistry.AutoRegisterPcSc (bool *updateRegistryFile*) [inline]

Updates PC/SC reader registries (internal and file-based).

This method consults the PC/SC SCardResourceManager in order to list all available card readers. Then it compares this list with the registry and adds all PC/SC card readers that are not already registered to this registry.

Any PC/SC card readers that are registered, but that were not returned by the SCardResourceManager are purged from the registry.We save the registry if there is a valid file name. The registry file name is created during instantiation of CardTerminal. This additional check allows registerless CardTerminal (fallback for systems that don't allow access). We intend to support systems such as Wyse S10 ThinOS with RDC sessions into Windows 2008 server this way fixing issues reported in OCT2012 by a customer.

If the registry was changed, then the entire registration file is rewritten. Thus I/O exceptions may occur.

Parameters

updateRegistry←File	Flag to control file update. This allows us to suppress registry file updates during runtime when the polling thread CardTerminalManager._PollingThread is looking for additional readers.

6.24.3.3 IEnumerator Subsembly.SmartCard.CardTerminalRegistry.GetEnumerator ( ) [inline]

Get enumerator over the registered CardTerminalConfig entries.

Returns

The enumerator.

See also

M:System.Collections.IEnumerable.GetEnumerator()

6.24.3.4 CardTerminalConfig Subsembly.SmartCard.CardTerminalRegistry.GetRegistered (string *sAssignedName*) [inline]

Retrieve a card terminal configuration from this registry.

The returned reference is a direct reference to the CardTerminalConfig instance in the registry. However, any changes made to the CardTerminalConfig properties are not updated to the persistent registration file. Thus, the method Save↩ Registry must be called to make any changes persistent.

Parameters

sAssignedName	The distinguished name to compare to the CardTerminalConfig.AssignedName property of the registered CardTerminalConfig instances in order to look up the card terminal.

Returns

If the sought card terminal configuration is found, then its CardTerminalConfig instance is returned. If not, then null is returned.

6.24.3.5 bool Subsembly.SmartCard.CardTerminalRegistry.IsRegistered (CardTerminalConfig *aCtConfig*) [inline]

Check whether a card terminal configuration is registered.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

aCtConfig	CardTerminalConfig instance that identifies the card terminal configuration to be found in the registry. Only the CardTerminalConfig.AssignedName is used in order to look up the registered card terminal. The other properties of the CardTerminalConfig instance are ignored.

Returns

If the card terminal configuration is registered then true is returned. If not, then false is returned.

6.24.3.6 bool Subsembly.SmartCard.CardTerminalRegistry.IsRegistered (string *sAssignedName*) [inline]

Check whether a card terminal configuration is registered.

Parameters

sAssignedName	The distinguished name to compare to the CardTerminalConfig.AssignedName property of the registered CardTerminalConfig instances in order to look up the card terminal.

Returns

If the card terminal configuration is registered then true is returned. If not, then false is returned.

6.24.3.7 int Subsembly.SmartCard.CardTerminalRegistry.LoadRegistry ( ) [inline]

Reloads the entire card terminal registry from its persistent registration file. .NET 4: This API is potentially dangerous; it asserts which means it cant be transparent. We assume that partial trust code can safely call this API. In fact, it is intended that it is safe for partial trust code to call this method. Therefore, it is SecuritySafeCritical.

Returns

Returns the number of entries that have been loaded. If the persistent registration file was not found, then -1 is returned. Other I/O or XML exceptions may occur. If SmartCard API DLL has insufficent rights.

6.24.3.8 void Subsembly.SmartCard.CardTerminalRegistry.Register (CardTerminalConfig *aCtConfig*) [inline]

Add a card terminal configuration to registry (memory and persistent registry).

The entire registration file is rewritten after successfully adding the new card terminal configuration. Thus I/O exceptions may occur. pseudo reader names are only written to memory representation.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentOutOfRange↩ Exception	Thrown when one or more arguments are outside the required range.

Parameters

aCtConfig	CardTerminalConfig instance that contains the complete card terminal configuration to be added to the registry. This information is not validated, thus it better should be correct.

6.24.3.9 void Subsembly.SmartCard.CardTerminalRegistry.RemoveDeadPcscEntries (bool *updateRegistryFile*) [inline]

Removes the missing PC/SC readers listed in registry file.

Parameters

updateRegistry↩ File	Flag to control registry file update. This allows us to suppress registry file updates during runtime when the polling thread CardTerminalManager._PollingThread is looking for additional readers.

6.24.3.10 void Subsembly.SmartCard.CardTerminalRegistry.Revoke (CardTerminalConfig *aCtConfig*) [inline]

Removes a card terminal configuration from this registry.

If the indicated card terminal is not registered, then this method simply does nothing.

The entire registration file is rewritten after successfully removing the card terminal configuration. Thus I/O exceptions may occur.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

aCtConfig	CardTerminalConfig instance that identifies the card terminal configuration to be removed from the registry. Only the CardTerminalConfig.AssignedName is used in order to look up the registered card terminal. The other properties of the CardTerminalConfig instance are ignored.

6.24.3.11 void Subsembly.SmartCard.CardTerminalRegistry.Revoke (string *sAssignedName*) [inline]

Removes a card terminal configuration from this registry.

If the indicated card terminal is not registered, then this method simply does nothing.

The entire registration file is rewritten after successfully removing the card terminal configuration. Thus I/O exceptions may occur.

Parameters

sAssignedName	The assigned name to compare to the CardTerminalConfig.AssignedName property of the registered CardTerminalConfig instances in order to look up the card terminal.

6.24.4 Property Documentation

6.24.4.1 int Subsembly.SmartCard.CardTerminalRegistry.Count [get]

Provides the number of card terminal configurations stored in this registry.

The count.

6.24.4.2 string Subsembly.SmartCard.CardTerminalRegistry.RegistryFilename [get]

This property provides the filename including the complete path to the SmartCard Subsembly registration file.

This information is provided for diagnostic purposes only. Applications shall not directly read or write the registration file!

The filename of the registry file.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardTerminal↩ Registry.cs

6.25 Subsembly.SmartCard.CardTerminalSlot Class Reference

Internal class that is used by the CardTerminalManager to store the context for tracking the state of a card terminal slot.

Public Member Functions

- CardHandle AcquireCard (CardTypes nCardTypes, out CardActivationResult nActivationResult)

    Acquire exclusive access to a card inserted in this card terminal slot.
- CardTransactionResult BeginTransaction ()

    Begins a card transaction. Important for longer APDU exchanges (example: authentication + data access). This is a way for the host application to ensure that a block of APDU commands is processed completely before releasing a potentially shared smart card session to an external process.
- CardTransactionResult EndTransaction ()

    Ends a card transaction. This makes the card connection available to other processes again.
- void EjectCard ()

    Eject an inserted card.
- CardActivationResult ReconnectCard (out byte[ ] vbAtr)

    Resets a previously established communication connection with the card.
- CardResponseAPDU SendCommand (CardCommandAPDU aCommand)

    Sends a command to this card terminal slot.

- override string ToString ()

  Returns a string that represents the current object.
- byte[ ] GetTerminalSerialNumber ()

  ! ================= work in progress =========================

Properties

- ICardTerminal CardTerminal `[get]`

  The card terminal of this card terminal slot.
- string CardTerminalName `[get]`

  The CardTerminalConfig.AssignedName of the card terminal that hosts this slot.
- int SlotNumber `[get]`

  The number of this slot.
- CardTerminalSlotState State `[get]`

  The current state of this card terminal slot.
- bool IsClass2 `[get]`

  Gets a value indicating whether this object is class 2.
- byte[ ] ATR `[get]`

  Returns the ATR of the card most recently inserted in this card terminal slot.

6.25.1 Detailed Description

Internal class that is used by the CardTerminalManager to store the context for tracking the state of a card terminal slot.

6.25.2 Member Function Documentation

6.25.2.1 **CardHandle** Subsembly.SmartCard.CardTerminalSlot.AcquireCard (**CardTypes** *nCardTypes,* out **CardActivationResult** *nActivationResult*) `[inline]`

Acquire exclusive access to a card inserted in this card terminal slot.

In order to successfully acquire a card handle, this card terminal slot must not be owned by a previously acquired card handle. Thus a previously acquired card handle must have been orderly relinquished (disposed of) before another card handle can be acquired on the same card terminal slot.

Only when a card is actually inserted in the indicated card terminal slot it is possible to acquire it. If no card is inserted, then this method returns CardActivationResult.NoCard and no CardHandle instance.

Usually this method should only be invoked, when the application knows that there is a card inserted into this card reader slot. An application can know this by handling card insertion events.

Parameters

nCardTypes	

Bit field that selects the acceptable card types. In order to accept all asynchronous cards, the value CardTypes.↩ ProcessorCards should be passed. The caller must not mix asynchronous and synchronous card types in one call. Also, the caller must only request one synchronous card type at once.

Parameters

nActivationResult	[out] Returns the result from the attempt to activate the inserted card. Only if this value is returned as CardActivationResult.Success, then a card handle was successfully acquired and will be returned as the function result.

Returns

Returns a successfully acquired card handle. If the inserted card could not be activated successfully, then null is returned instead. Whenever a card handle is returned it must be properly disposed of as soon as it is no longer needed.

6.25.2.2 CardTransactionResult Subsembly.SmartCard.CardTerminalSlot.BeginTransaction ( ) `[inline]`

Begins a card transaction. Important for longer APDU exchanges (example: authentication + data access). This is a way for the host application to ensure that a block of APDU commands is processed completely before releasing a potentially shared smart card session to an external process.

Not all card terminal implementations support protected transaction.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Returns

A CardTransactionResult.

6.25.2.3 void Subsembly.SmartCard.CardTerminalSlot.EjectCard ( ) `[inline]`

Eject an inserted card.

This is a way for the host application to eject a card in case it tries to power up card unsuccessfully leaving it without a handle

6.25.2.4 CardTransactionResult Subsembly.SmartCard.CardTerminalSlot.EndTransaction ( ) `[inline]`

Ends a card transaction. This makes the card connection available to other processes again.

Disconnecting a card also ends a transaction. Windows-based platforms may end a transaction automatically after a 5 second timeout

Returns

A CardTransactionResult.

6.25.2.5 byte [ ] Subsembly.SmartCard.CardTerminalSlot.GetTerminalSerialNumber ( ) `[inline]`

! ================= work in progress =========================

Reads the vendor supplied terminal serial number.

Returns

Reader serial number as byte array.

6.25.2.6 CardActivationResult Subsembly.SmartCard.CardTerminalSlot.ReconnectCard (out byte[ ] *vbAtr*) `[inline]`

Resets a previously established communication connection with the card.

Exceptions

CardTerminalException	This slot is not in the CardTerminalSlotState.Accepted State.

Returns

Returns the result from the attempt to reset and reactivate the card. See CardActivationResult for possible return values.

6.25.2.7 CardResponseAPDU Subsembly.SmartCard.CardTerminalSlot.SendCommand (CardCommandAPDU *aCommand*) `[inline]`

Sends a command to this card terminal slot.

Exceptions

CardTerminalException	This slot is not in the CardTerminalSlotState.Accepted State.

Parameters

aCommand	.

Returns

A CardResponseAPDU.

6.25.2.8 override string Subsembly.SmartCard.CardTerminalSlot.ToString ( ) `[inline]`

Returns a string that represents the current object.

Returns

A string that represents the current object.

See also

M:System.Object.ToString()

6.25.3 Property Documentation

6.25.3.1 byte [ ] Subsembly.SmartCard.CardTerminalSlot.ATR `[get]`

Returns the ATR of the card most recently inserted in this card terminal slot.

Byte array that contains the complete ATR of the card, or `null` if none is available.

6.25.3.2 ICardTerminal Subsembly.SmartCard.CardTerminalSlot.CardTerminal `[get]`

The card terminal of this card terminal slot.

The card terminal.

6.25.3.3 string Subsembly.SmartCard.CardTerminalSlot.CardTerminalName [get]

The CardTerminalConfig.AssignedName of the card terminal that hosts this slot.

The name of the card terminal.

6.25.3.4 bool Subsembly.SmartCard.CardTerminalSlot.IsClass2 [get]

Gets a value indicating whether this object is class 2.

True if this object is class 2, false if not.

6.25.3.5 int Subsembly.SmartCard.CardTerminalSlot.SlotNumber [get]

The number of this slot.

The first, and usually only, slot of a card terminal has the number zero. If a single card terminal has multiple card slots, then they can be addressed by their slot number.

The slot number.

6.25.3.6 CardTerminalSlotState Subsembly.SmartCard.CardTerminalSlot.State [get]

The current state of this card terminal slot.

The state.

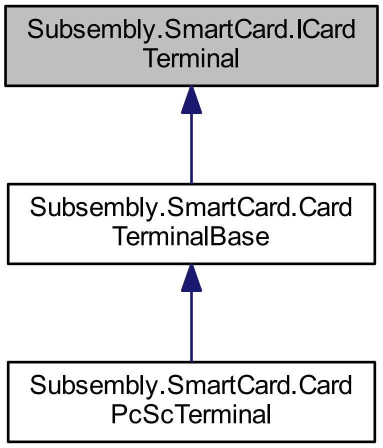
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/CardTerminal↩
  Slot.cs

6.26 Subsembly.SmartCard.ICardTerminal Interface Reference

This interface supports abstract access to any smart card terminal, regardless of the actual driver technology. An instance of this interface is always bound to a particular card terminal unit installed in the system.

Inheritance diagram for Subsembly.SmartCard.ICardTerminal:


Subsembly.SmartCard.ICard Terminal
Subsembly.SmartCard.Card TerminalBase
Subsembly.SmartCard.Card PcScTerminal

Public Member Functions

- void Init (CardTerminalConfig aConfig)

  Initialises this card terminal with the given configuration information.
- void ResetCardTerminal ()

  Entirely resets the card terminal, disconnecting (powering down) any existing card connection.
- bool IsCardPresent (int nSlot, ref bool fRequiresRemoval)

  Quick poll that indicates whether a card is present in a card terminal slot. This method can be called from any thread. It is the only multi-threading safe method of this interface.
- CardTransactionResult BeginTransaction (int nSlot)

  Begins a card transaction.
- CardTransactionResult EndTransaction (int nSlot)

  Ends a card transaction.
- CardActivationResult ConnectCard (int nSlot, CardTypes nCardType, out byte[ ] vbATR)

  Powers up the inserted card and attempts to establish a communication connection with it. Once the connection is established the card is exclusively locked and cannot be accessed by another application or thread.
- CardActivationResult ReconnectCard (int nSlot, out byte[ ] vbATR)

  Resets a previously established communication connection with the card.
- CardResponseAPDU SendCommand (int nSlot, CardCommandAPDU aCommand)

  Sends the command APDU to the given card terminal slot and returns the response APDU from the inserted card.
- void DisconnectCard (int nSlot)

  Powers down the inserted card, and terminates the logical connection with it.
- void DeliverCard (int nSlot, bool fCapture)

  Explicitly requests that an inserted card is delivered or captured. This is only applicable if the card terminal has such mechanism, in most cases it won't.

Properties

- CardTerminalConfig Config `[get]`

  Returns the configuration information with which this card terminal was initialized.
- string InternalName `[get]`

  This is the internal name of this card terminal. This name may be more useful to the programmer than to the user.
- string ManufacturerName `[get]`

  Name of the manufacturer of this card terminal.
- string ProductName `[get]`

  Name of the product itself. Short and clear.
- string ProductVersion `[get]`

  Version of the product. It is not defined whether this is some firmware version or some driver version. Thus it is suggested that the returned string supplies as much human readable version information as possible.
- string ProductAdditionalInfo `[get]`

  Additional information about the product that is neither its name, nor some version information. This can be anything that the card terminal manufacturer wants to tell.
- string ProductSerialNumber `[get]`

  Serial number of the product if available.
- string PhysicalConnection `[get]`

  Returns descriptive information about the physical connector that this card terminal is attached to. This should use some common notation such as COM1, LPT1 or USB.
- CardTerminalFunctionalUnits FunctionalUnits `[get]`

  Indicates which functional units are available with this card terminal. The availability of certain functional units classifies the card terminal according to the ZKA. Other uniits indicate support for card ejection and positioning.
- int Slots `[get]`

  Indicates the total number of slots of this card terminal.
- int UserSlots `[get]`

  Indicates the number of user accessible card slots of this card terminal.

6.26.1 Detailed Description

This interface supports abstract access to any smart card terminal, regardless of the actual driver technology. An instance of this interface is always bound to a particular card terminal unit installed in the system.

As an implementation of this interface very likely holds onto some unmanaged resources, it is required that all objects implementing this interface, also implement the IDisposable interface, releasing its unmanaged resources in its Dispose method.

The SCard Class Library currently provides two implementations of this Interface, one is based on its PC/SC wrapper classes and another is based on its CT-API wrapper classes. However, future versions may include additional implementations. Also, it is possible for third parties to supply additional implementations of this interface.

Being an abstraction, this Interface only supports features that are common among different driver technologies. Hence, more sophisticated applications might need to use the low level wrappers directly.

The methods of the ICardTerminal interface are only to be called from the owning thread. There is a single exception: The method IsCardPresent must be callable from any thread.

In order to simplify implementations of this interface the common base class CardTerminalBase is supplied.

Card terminals that include a PIN pad and support secure PIN entry shall implement the interface ICardTerminalClass2 alongside this interface.

6.26.2 Member Function Documentation

6.26.2.1 CardTransactionResult Subsembly.SmartCard.ICardTerminal.BeginTransaction (int *nSlot*)

Begins a card transaction.

Parameters

nSlot	Slot number of card terminal slot of interest. The primary slot is identified by the slot number zero. The slot number must be in the range from zero through Slots-1.

Returns

Returns the result from the attempt to begin a card transaction. See CardTransactionResult for possible return values.

Implemented in Subsembly.SmartCard.CardTerminalBase.

6.26.2.2 CardActivationResult Subsembly.SmartCard.ICardTerminal.ConnectCard (int *nSlot*, CardTypes *nCardType*, out byte[ ] *vbATR*)

Powers up the inserted card and attempts to establish a communication connection with it. Once the connection is established the card is exclusively locked and cannot be accessed by another application or thread.

Parameters

nSlot	Slot number of card terminal slot of interest. The primary slot is identified by the slot number zero. The slot number must be in the range from zero through Slots-1.
nCardType	Card type to connect to. If the actual ICardTerminal implementation does not support the requested card type, then CardActivationResult.NotImplemented is returned.
vbATR	[out] Returns the complete ATR that was received when the card was successfully powered up. Will be null whenever the card was not successfully powered up.

Returns

Returns the result from the attempt to activate the card. See CardActivationResult for possible return values.

Implemented in Subsembly.SmartCard.CardTerminalBase.

6.26.2.3 void Subsembly.SmartCard.ICardTerminal.DeliverCard (int *nSlot*, bool *fCapture*)

Explicitly requests that an inserted card is delivered or captured. This is only applicable if the card terminal has such mechanism, in most cases it won't.

This method may be invoked regardless of whether a card is inserted, the card is powered up, or no card is inserted. It shall succeed in any of these cases.

If the card in the indicated slot is still connected (powered up), then it shall be automatically disconnected (powered down) when this method is invoked. Even if a card terminal does not have an eject or capture mechanism, then it still has to disconnect (power down) when this method is invoked.

Parameters

nSlot	Slot number of card terminal slot of interest. The primary slot is identified by the slot number zero. The slot number must be in the range from zero through Slots-1.
fCapture	If true, then the card shall be confiscated, if possible. If false, then the card shall simply be eject to the card holder. If a card terminal supports ejecting, but not confiscating cards, then it shall ignore this parameter and always just eject the card.

Implemented in Subsembly.SmartCard.CardTerminalBase.

6.26.2.4 void Subsembly.SmartCard.ICardTerminal.DisconnectCard (int *nSlot*)

Powers down the inserted card, and terminates the logical connection with it.

Parameters

nSlot	Slot number of card terminal slot of interest. The primary slot is identified by the slot number zero. The slot number must be in the range from zero through Slots-1.

Implemented in Subsembly.SmartCard.CardTerminalBase.

6.26.2.5 CardTransactionResult Subsembly.SmartCard.ICardTerminal.EndTransaction (int *nSlot*)

Ends a card transaction.

Parameters

nSlot	Slot number of card terminal slot of interest. The primary slot is identified by the slot number zero. The slot number must be in the range from zero through Slots-1.

Returns

Returns the result from the attempt to end a card transaction. See CardTransactionResult for possible return values.

Implemented in Subsembly.SmartCard.CardTerminalBase.

6.26.2.6 void Subsembly.SmartCard.ICardTerminal.Init (CardTerminalConfig *aConfig*)

Initialises this card terminal with the given configuration information.

During CardTerminalManager.Startup(bool,bool) the CardTerminalManager scans the CardTerminalRegistry and instantiates all registered and enabled card terminals using their default constructor. Immediately after instantiation the Card←TerminalManager calls this Init method, passing the stored configuration information from the CardTerminalRegistry.

Whenever a card terminal is instantiated via its default constructor this Init method must be called in order to initialise it. Additionally an ICardTerminal implementation may provide non-default constructors that implicitly initialise the card terminal and therefore there is no need to call Init. Using specialised non-default constructors, however, makes applications dependent on a specific ICardTerminal implementation.

Parameters

aConfig	Provides the complete configuration information. This is never null.

See also

CardTerminalRegistry, CardTerminalManager, CardTerminalConfig, CardTerminalManager.Startup(bool,bool)

Implemented in Subsembly.SmartCard.CardTerminalBase.

6.26.2.7 bool Subsembly.SmartCard.ICardTerminal.IsCardPresent (int *nSlot*, ref bool *fRequiresRemoval*)

Quick poll that indicates whether a card is present in a card terminal slot. This method can be called from any thread. It is the only multi-threading safe method of this interface.

Parameters

nSlot	Slot number of card terminal slot of interest. The primary slot is identified by the slot number zero. The slot number must be in the range from zero through Slots-1.
fRequires↩ Removal	[in,out] Indicates whether user interaction is required to remove the card. This flag only matters for motorized readers and contactless readers.

Returns

Returns true if a card is present in the indicated slot and false if not.

Implemented in Subsembly.SmartCard.CardTerminalBase.

6.26.2.8 CardActivationResult Subsembly.SmartCard.ICardTerminal.ReconnectCard (int *nSlot*, out byte[ ] *vbATR*)

Resets a previously established communication connection with the card.

Parameters

nSlot	Slot number of card terminal slot of interest. The primary slot is identified by the slot number zero. The slot number must be in the range from zero through Slots-1.
vbATR	Card ATR.

Returns

Returns the result from the attempt to reset and reactivate the card. See CardActivationResult for possible return values.

Implemented in Subsembly.SmartCard.CardTerminalBase.

6.26.2.9 CardResponseAPDU Subsembly.SmartCard.ICardTerminal.SendCommand (int *nSlot*, CardCommandAPDU *aCommand*)

Sends the command APDU to the given card terminal slot and returns the response APDU from the inserted card.

If the connected card type is a memory card, then an application uses this method in order to send command APDUs according to the MCT specification part 7: *IC Cards with Synchronous Transmission; Usage of Interindustry Commands*.

Parameters

nSlot	Slot number of card terminal slot of interest. The primary slot is identified by the slot number zero. The slot number must be in the range from zero through Slots-1.
aCommand	Complete command APDU to be transmitted. This must not be null.

Returns

Returns the response APDU from the inserted card. This is never null. If no response APDU was received for whatever reason, then a CardTerminalException is thrown.

Implemented in Subsembly.SmartCard.CardTerminalBase.

6.26.3 Property Documentation

6.26.3.1 CardTerminalConfig Subsembly.SmartCard.ICardTerminal.Config [get]

Returns the configuration information with which this card terminal was initialized.

The configuration.

6.26.3.2 CardTerminalFunctionalUnits Subsembly.SmartCard.ICardTerminal.FunctionalUnits [get]

Indicates which functional units are available with this card terminal. The availability of certain functional units classifies the card terminal according to the ZKA. Other uniits indicate support for card ejection and positioning.

The functional units.

6.26.3.3 string Subsembly.SmartCard.ICardTerminal.InternalName [get]

This is the internal name of this card terminal. This name may be more useful to the programmer than to the user.

The name of the internal.

6.26.3.4 string Subsembly.SmartCard.ICardTerminal.ManufacturerName [get]

Name of the manufacturer of this card terminal.

The name of the manufacturer.

6.26.3.5 string Subsembly.SmartCard.ICardTerminal.PhysicalConnection [get]

Returns descriptive information about the physical connector that this card terminal is attached to. This should use some common notation such as COM1, LPT1 or USB.

The physical connection.

6.26.3.6 string Subsembly.SmartCard.ICardTerminal.ProductAdditionalInfo [get]

Additional information about the product that is neither its name, nor some version information. This can be anything that the card terminal manufacturer wants to tell.

Information describing the product additional.

6.26.3.7 string Subsembly.SmartCard.ICardTerminal.ProductName [get]

Name of the product itself. Short and clear.

The name of the product.

6.26.3.8 string Subsembly.SmartCard.ICardTerminal.ProductSerialNumber [get]

Serial number of the product if available.

The product serial number.

6.26.3.9 string Subsembly.SmartCard.ICardTerminal.ProductVersion [get]

Version of the product. It is not defined whether this is some firmware version or some driver version. Thus it is suggested that the returned string supplies as much human readable version information as possible.

The product version.

6.26.3.10 int Subsembly.SmartCard.ICardTerminal.Slots [get]

Indicates the total number of slots of this card terminal.

A card terminal may feature zero or more user accessible card slots and zero or more internal secure application module (SAM) sockets. This property shall provide the total number of all of these. All card terminals are required to have at least one card slot or socket, thus this property must always be at least one.

Card slots and sockets are numbered starting at zero. The very first card slot has the slot number zero. This card slot is also known as the primary card slot.

This value is always at least one.

See also

UserSlots

6.26.3.11 int Subsembly.SmartCard.ICardTerminal.UserSlots [get]

Indicates the number of user accessible card slots of this card terminal.

This property provides the number of user accessible card slots that are featured by this reader. For a card reader that features only secure application module (SAM) sockets, this property has the value zero.

The user accessible slots are numbered from zero through UserSlots-1.

Basically this property controls how many slots will be monitored for card insertion or removal. Only user accessible slots are monitored and will fire card inserted and card removed events.

This value may be zero or more.

See also

Slots

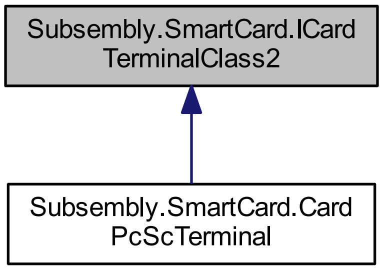
The documentation for this interface was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/ICardTerminal.↩
cs

6.27 Subsembly.SmartCard.ICardTerminalClass2 Interface Reference

This interface provides methods that are only applicable to card terminals that comply at least with ZKA class 2, i.e. that have a PIN pad and support for secure PIN entry. Card terminals supporting class 2 functionality implement this interface alongside the ICardTerminal interface.

Inheritance diagram for Subsembly.SmartCard.ICardTerminalClass2:


Subsembly.SmartCard.ICard TerminalClass2
Subsembly.SmartCard.Card PcScTerminal

Properties

- bool IsClass2 `[get]`

  Indicates whether the connected card terminal really is a class 2 terminal.

6.27.1 Detailed Description

This interface provides methods that are only applicable to card terminals that comply at least with ZKA class 2, i.e. that have a PIN pad and support for secure PIN entry. Card terminals supporting class 2 functionality implement this interface alongside the ICardTerminal interface.

Whether a card terminal is a class 2 card terminal can be determined through the ICardTerminal.FunctionalUnits property of the ICardTerminal interface.

6.27.2 Property Documentation

6.27.2.1 bool Subsembly.SmartCard.ICardTerminalClass2.IsClass2 `[get]`

Indicates whether the connected card terminal really is a class 2 terminal.

If this property is `false`, then none of the other methods of this interface should be called. If this is `true`, then the physically connected card terminal really is a class 2 card termnal that supports secure PIN entry.

This property is required because in most cases the implementation of this interface will only know at runtime whether the connected card terminal is really a class 2 card terminal. Hence, the presence of the ICardTerminalClass2 interface does only mean that the implementation supports class 2. It still needs a class 2 terminal, to actually be a class 2 solution.

The documentation for this interface was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/ICardTerminal↩
  Class2.cs

6.28 Subsembly.SmartCard.LogHelper Class Reference

Easy-to-use helper class to log (debug or trace) info

Static Public Member Functions

- static void SetTrace (bool onOff)

  Sets a trace flag.
- static void **Reset** ()
- static void WriteLine (string msg)

  Writes a line to the log output

6.28.1 Detailed Description

Easy-to-use helper class to log (debug or trace) info

6.28.2 Member Function Documentation

6.28.2.1 static void Subsembly.SmartCard.LogHelper.SetTrace (bool *onOff*) [inline], [static]

Sets a trace flag.

Parameters

onOff	True to disable, false to enable the on.

6.28.2.2 static void Subsembly.SmartCard.LogHelper.WriteLine (string *msg*) [inline], [static]

Writes a line to the log output

Parameters

msg	The message.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/LogHelper.cs

6.29 SmartCardAPI.CardModule.PIV.PivAddonAuthenticator Class Reference

A piv card crypto pkcs #1.

Public Member Functions

- PivAddonAuthenticator (PivCardEdge pivCard)

  PIV card helper class for Constructor. Checks for valid parameters to prevent surprises down the road. Any RSA 2048 bit key is allowed.
- bool ExternalAuthenticate (byte algorithmIdentifier, byte keyReference, byte[ ] keyValue)

Convenience method for external authentication

- bool SendResponseToChallenge (byte algorithmReference, byte keyReference, byte[ ] response)

  Sends a response to challenge. This method should probably go private. 128 byte response smells like RSA 1024 and requires a review to accommodate RSA2048. Handling challenge/response can probably all be included in Crypt() method as the central GENERAL AUTHENTICATE processor.
- bool InternalAuthenticate (int keyIdentifier, PivCardCertificate x509Certificate)

  INTERNAL AUTHENTICATION convenience method that does the full round trip of an internal authentication. It can be used for PIV authentication Depending on the key used for this operation prior successful PIN VERIFY is required.

6.29.1 Detailed Description

A piv card crypto pkcs #1.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 SmartCardAPI.CardModule.PIV.PivAddonAuthenticator.PivAddonAuthenticator (PivCardEdge *pivCard*) [inline]

PIV card helper class for Constructor. Checks for valid parameters to prevent surprises down the road. Any RSA 2048 bit key is allowed.

Parameters

pivCard	PIV card edge handle
keyReference	(Optional) The key reference. Default is 0x9C, the digital signing key
algoType	(Optional) The digital signature algorithm. Default is RSA 2048.

6.29.3 Member Function Documentation

6.29.3.1 bool SmartCardAPI.CardModule.PIV.PivAddonAuthenticator.ExternalAuthenticate (byte *algorithmIdentifier,* byte *keyReference,* byte[ ] *keyValue*) [inline]

Convenience method for external authentication

23FEB2020 moved code from PivCardEdge.cs to include Crypt() in next review 16NOV2016 call extended APDUs for OCS card.

Parameters

algorithm↵ Identifier	Type of the algo.
keyReference	key references other than '80' and '81', if they are used for card management operations.
keyValue	The key value.

Returns

True if it succeeds, false if it fails.

6.29.3.2 bool SmartCardAPI.CardModule.PIV.PivAddonAuthenticator.InternalAuthenticate (int *keyIdentifier,* PivCardCertificate *x509Certificate*) [inline]

INTERNAL AUTHENTICATION convenience method that does the full round trip of an internal authentication. It can be used for PIV authentication Depending on the key used for this operation prior successful PIN VERIFY is required.

TODO: move to addon class. Internal Authenticate is not part of NIST middleware API and therefore a great candidate for a library that adds additional value.

Parameters

keyIdentifier	intended for 9A, 9B, 9E
x509Certificate	X509 certificate

Returns

True if it the card responds with expected data, false otherwise.

6.29.3.3 bool SmartCardAPI.CardModule.PIV.PivAddonAuthenticator.SendResponseToChallenge (byte *algorithmReference,* byte *keyReference,* byte[ ] *response*) [inline]

Sends a response to challenge. This method should probably go private. 128 byte response smells like RSA 1024 and requires a review to accommodate RSA2048. Handling challenge/response can probably all be included in Crypt() method as the central GENERAL AUTHENTICATE processor.

23FEB2020 moved code from PivCardEdge.cs to include Crypt() in next review

Exceptions

ArgumentException	Thrown when one or more arguments have unsupported or illegal values.

Parameters

algorithm←Reference	The algorithm reference.
keyReference	key references other than '80' and '81', if they are used for card management operations.
response	The response.

Returns

True if it succeeds, false if it fails.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv←AddonAuthenticator.cs

6.30 SmartCardAPI.CardModule.PIV.PivCardAddonPkcs1 Class Reference

A piv card crypto pkcs #1.

Public Member Functions

- PivCardAddonPkcs1 (PivCardEdge pivCard, byte keyReference=PIVC.KEYTYPE_DIGITAL_SIGNATURE, byte algoType=PIVC.ALGOID_RSA2048, byte hashType=PIVC.HASH_ALGOID_SHA256)

    PIV card helper class for Constructor. Checks for valid parameters to prevent surprises down the road. Any RSA 2048 bit key is allowed.
- byte[ ] SignFile (string fileName)

Generate RSA 2048, SHA1 under the the signing key requested with this classes constructor.

- byte[ ] Sign (byte[ ] data)

    Generate RSA 2048, SHA1 under the the signing key requested with this classes constructor.

- byte[ ] Sign (string text)

    Generate RSA 2048, SHA1 under the the signing key requested with this classes constructor.

- bool Verify (byte[ ] signature, string text)

    Verifies signature. Note that this is a work around using the card to sign data i.e. it also requires previous PIN verification to allow signature verification. Verification should use the public key and client crypto instead.

6.30.1 Detailed Description

A piv card crypto pkcs #1.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 SmartCardAPI.CardModule.PIV.PivCardAddonPkcs1.PivCardAddonPkcs1 (PivCardEdge *pivCard*, byte *keyReference* = `PIVC.KEYTYPE_DIGITAL_SIGNATURE`, byte *algoType* = `PIVC.ALGOID_RSA2048`, byte *hashType* = `PIVC.HASH_ALGOID_SHA256`) `[inline]`

PIV card helper class for Constructor. Checks for valid parameters to prevent surprises down the road. Any RSA 2048 bit key is allowed.

Parameters

pivCard	PIV card edge handle
keyReference	(Optional) The key reference. Default is 0x9C, the digital signing key
algoType	(Optional) The digital signature algorithm. Default is RSA 2048.
hashType	(Optional) The hash used for digital signature. Default is SHA256.

6.30.3 Member Function Documentation

6.30.3.1 byte [ ] SmartCardAPI.CardModule.PIV.PivCardAddonPkcs1.Sign (byte[ ] *data*) `[inline]`

Generate RSA 2048, SHA1 under the the signing key requested with this classes constructor.

Parameters

data	byte array to be signed.

Returns

256 byte signatur.

6.30.3.2 byte [ ] SmartCardAPI.CardModule.PIV.PivCardAddonPkcs1.Sign (string *text*) `[inline]`

Generate RSA 2048, SHA1 under the the signing key requested with this classes constructor.

Parameters

text	string to be signed.

Returns

256 byte signature.

6.30.3.3 byte [ ] SmartCardAPI.CardModule.PIV.PivCardAddonPkcs1.SignFile (string *fileName*) `[inline]`

Generate RSA 2048, SHA1 under the the signing key requested with this classes constructor.

Parameters

aCard	card handle to access physical card
fileName	name of the file that need to get signed

Returns

256 byte signature.

6.30.3.4 bool SmartCardAPI.CardModule.PIV.PivCardAddonPkcs1.Verify (byte[ ] *signature,* string *text*) `[inline]`

Verifies signature. Note that this is a work around using the card to sign data i.e. it also requires previous PIN verification to allow signature verification. Verification should use the public key and client crypto instead.

- decrypt sigranture under public key
- decode
- calculate hash of known original data
- decode padded datagram
- compare hash to decoded hash value -> if all the above matches, we are ok and the signature is correct

Parameters

signature	256 byte signature.
text	string to be signed.

Returns

true if signature matches.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ AddonSigner.cs

6.31 SmartCardAPI.CardModule.PIV.PivCardCapabilites Class Reference

Helper class to parse, analyze and display BER encoded PIV card capability data. This class allows easy access to any element within the PIV card capability container. Note that this class does not provide card access methods. Data must be read external to this class and then passed during instantiation. Ref SP 800-73-3 Part 1, page 20, , SP800-73-3.

Public Member Functions

- PivCardCapabilites (byte[ ] cardCapabilityData)

  Constructor.
- override string ToString ()

  Converts the card capability container to a nice human readable string. Using one line per non-zero data item. This is for quick and dirty implementation.

Properties

- string CardIdentifier `[get]`

  Allows public access to Card Identifier.
- string CapabilityContainerVersionNumber `[get]`

  Allows public access to Capability Container Version Number.
- string CapabilityGrammarVersionNumber `[get]`

  Allows public access to Capability Grammar VersionNumber.
- string ApplicationsCardURL `[get]`

  Allows public access to Applications Card URL.
- string PKCS15 `[get]`

  Allows public access to PKCS15 flag.
- string RegisteredDataModelNumber `[get]`

  Allows public access to Registered DataModel Number.
- string AccessControlRuleTable `[get]`

  Allows public access to Access Control Rule Table.
- string CardAPDUs `[get]`

  Allows public access to Card APDUs.
- string RedirectionTag `[get]`

  Allows public access to RedirectionTag.
- string CapabilityTuples `[get]`

  Allows public access to CapabilityTuples.
- string StatusTuples `[get]`

  Allows public access to Status Tuples.
- string NextCCC `[get]`

  Allows public access to Next Card Capability Container (CCC)
- string ExtendedApplicationCardURL `[get]`

  Allows public access to Extended Application Card URL.
- string SecurityObjectBuffer `[get]`

  Allows public access to Security Object Buffer.
- string ErrorDetectionCode `[get]`

  Allows public access to Error Detection Code.

6.31.1 Detailed Description

Helper class to parse, analyze and display BER encoded PIV card capability data. This class allows easy access to any element within the PIV card capability container. Note that this class does not provide card access methods. Data must be read external to this class and then passed during instantiation. Ref SP 800-73-3 Part 1, page 20, , SP800-73-3.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 SmartCardAPI.CardModule.PIV.PivCardCapabilites.PivCardCapabilites (byte[ ] *cardCapabilityData*) `[inline]`

Constructor.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

cardCapability↩ Data	Information describing the card capability.

6.31.3 Member Function Documentation

6.31.3.1 override string SmartCardAPI.CardModule.PIV.PivCardCapabilites.ToString ( ) `[inline]`

Converts the card capability container to a nice human readable string. Using one line per non-zero data item. This is for quick and dirty implementation.

Returns

Returns a nicely formatted string usable for tracing output.

See also

M:System.Object.ToString()

6.31.4 Property Documentation

6.31.4.1 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.AccessControlRuleTable `[get]`

Allows public access to Access Control Rule Table.

The access control rule table.

6.31.4.2 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.ApplicationsCardURL `[get]`

Allows public access to Applications Card URL.

The applications card URL.

6.31.4.3 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.CapabilityContainerVersionNumber `[get]`

Allows public access to Capability Container Version Number.

The capability container version number.

6.31.4.4 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.CapabilityGrammarVersionNumber `[get]`

Allows public access to Capability Grammar VersionNumber.

The capability grammar version number.

6.31.4.5 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.CapabilityTuples [get]

Allows public access to CapabilityTuples.

The capability tuples.

6.31.4.6 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.CardAPDUs [get]

Allows public access to Card APDUs.

The card a pd us.

6.31.4.7 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.CardIdentifier [get]

Allows public access to Card Identifier.

The identifier of the card.

6.31.4.8 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.ErrorDetectionCode [get]

Allows public access to Error Detection Code.

The error detection code.

6.31.4.9 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.ExtendedApplicationCardURL [get]

Allows public access to Extended Application Card URL.

The extended application card URL.

6.31.4.10 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.NextCCC [get]

Allows public access to Next Card Capability Container (CCC)

The next ccc.

6.31.4.11 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.PKCS15 [get]

Allows public access to PKCS15 flag.

The pkcs 15.

6.31.4.12 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.RedirectionTag [get]

Allows public access to RedirectionTag.

The redirection tag.

6.31.4.13 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.RegisteredDataModelNumber [get]

Allows public access to Registered DataModel Number.

The registered data model number.

6.31.4.14 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.SecurityObjectBuffer `[get]`

Allows public access to Security Object Buffer.

A buffer for security object data.

6.31.4.15 string SmartCardAPI.CardModule.PIV.PivCardCapabilites.StatusTuples `[get]`

Allows public access to Status Tuples.

The status tuples.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardCapabilities.cs

6.32 SmartCardAPI.CardModule.PIV.PivCardCertificate Class Reference

Helper class to parse X509 data coming from a card data container.

Public Member Functions

- PivCardCertificate (byte[ ] certificateData, string mode="CONTAINER")

  Convenience class to work with X509 certificates stored on PIV cards.
- override string ToString ()

  Convenience method that formats X509 cert in a nice human readable fashion. We are using .NET X509 certificate object wherever we can. Note that this object can be retrieved completely via member getter function.

Properties

- X509Certificate2 X509Certificate `[get]`

  Public access to Microsoft X509Certificate2 class defined in System.Security.Cryptography.X509Certificates. This allows easy usage of X509 certificates on Microsoft Windows systems with .NET framework 2.0 and up.
- byte[ ] CertificateData `[get]`

  Allows public access to raw cert data.
- byte[ ] RawData `[get]`

  Allows public access to the raw data of the X509 certificate container. This contains all tags including the leading '53' container tag.
- byte[ ] CertInfo `[get]`

  Allows public access to raw cert info. Data does not include TLV values related to PIV card container. The returned value can contain up to 1 bytes.
- byte[ ] MSCUID `[get]`

  Allows public access to raw MSCUID. Data does not include TLV values related to PIV card container. The returned value can contain up to 38 bytes.

6.32.1 Detailed Description

Helper class to parse X509 data coming from a card data container.

6.32.2 Constructor & Destructor Documentation

6.32.2.1 SmartCardAPI.CardModule.PIV.PivCardCertificate.PivCardCertificate (byte[ ] *certificateData,* string *mode =* "CONTAINER") [inline]

Convenience class to work with X509 certificates stored on PIV cards.

Exceptions

Exception	Thrown when an exception error condition occurs.

Parameters

certificateData	certificate data; may contain additional PIV related tags.
mode	(Optional) CONTAINER: parses data as PIV data container; X509: parses data as X509 data.

6.32.3 Member Function Documentation

6.32.3.1 override string SmartCardAPI.CardModule.PIV.PivCardCertificate.ToString ( ) [inline]

Convenience method that formats X509 cert in a nice human readable fashion. We are using .NET X509 certificate object wherever we can. Note that this object can be retrieved completely via member getter function.

Returns

Returns a nicely formatted string usable for quick tests.

See also

M:System.Object.ToString()

6.32.4 Property Documentation

6.32.4.1 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCertificate.CertificateData [get]

Allows public access to raw cert data.

Information describing the certificate.

6.32.4.2 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCertificate.CertInfo [get]

Allows public access to raw cert info. Data does not include TLV values related to PIV card container. The returned value can contain up to 1 bytes.

Information describing the cert.

6.32.4.3 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCertificate.MSCUID [get]

Allows public access to raw MSCUID. Data does not include TLV values related to PIV card container. The returned value can contain up to 38 bytes.

The mscuid.

6.32.4.4 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCertificate.RawData [get]

Allows public access to the raw data of the X509 certificate container. This contains all tags including the leading '53' container tag.

Information describing the raw.

6.32.4.5 X509Certificate2 SmartCardAPI.CardModule.PIV.PivCardCertificate.X509Certificate [get]

Public access to Microsoft X509Certificate2 class defined in System.Security.Cryptography.X509Certificates. This allows easy usage of X509 certificates on Microsoft Windows systems with .NET framework 2.0 and up.

The X509 certificate.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardCertificate.cs

6.33 SmartCardAPI.CardModule.PIV.PivCardChuid Class Reference

This is a helper class to parse, analyze and display PIV CHUID data. It processes data that typically resides in the CHUID data containter. However, it does not require a physical card connection.

Public Member Functions

- PivCardChuid (byte[ ] chuidData)

  Constructor.
- string **GetFascnUuid** ()
- override string ToString ()

  Formats this response APDU, including all its data, in a nice human readable fashion.

Properties

- byte[ ] FASCN [get]

  Allows public access to FASCN.
- byte[ ] GUID [get]

  Allows public access to GUID.
- byte[ ] UUID [get]

  Allows public access to UUID.
- byte[ ] EXP_DATE [get]

  Allows public access to CHUID expiration date (raw byte data). Raw data is typically ASCII encoded 4-byte date yyyymmdd.
- byte[ ] RawData [get]

  Allows public access to the raw data of the CHUID container.

6.33.1 Detailed Description

This is a helper class to parse, analyze and display PIV CHUID data. It processes data that typically resides in the CHUID data containter. However, it does not require a physical card connection.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 SmartCardAPI.CardModule.PIV.PivCardChuid.PivCardChuid (byte[ ] *chuidData*) `[inline]`

Constructor.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

chuidData	Information describing the chuid.

6.33.3 Member Function Documentation

6.33.3.1 override string SmartCardAPI.CardModule.PIV.PivCardChuid.ToString ( ) `[inline]`

Formats this response APDU, including all its data, in a nice human readable fashion.

Returns

Returns a nicely formatted string usable for tracing output.

See also

M:System.Object.ToString()

6.33.4 Property Documentation

6.33.4.1 byte [ ] SmartCardAPI.CardModule.PIV.PivCardChuid.EXP_DATE `[get]`

Allows public access to CHUID expiration date (raw byte data). Raw data is typically ASCII encoded 4-byte date yyyymmdd.

The exponent date.

6.33.4.2 byte [ ] SmartCardAPI.CardModule.PIV.PivCardChuid.FASCN `[get]`

Allows public access to FASCN.

The fascn.

6.33.4.3 byte [ ] SmartCardAPI.CardModule.PIV.PivCardChuid.GUID `[get]`

Allows public access to GUID.

The identifier of the unique.

6.33.4.4 byte [ ] SmartCardAPI.CardModule.PIV.PivCardChuid.RawData `[get]`

Allows public access to the raw data of the CHUID container.

Information describing the raw.

6.33.4.5 byte [ ] SmartCardAPI.CardModule.PIV.PivCardChuid.UUID [get]

Allows public access to UUID.

The uuid.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardChuid.cs

6.34 SmartCardAPI.CardModule.PIV.PivCardCryptoAes Class Reference

Easy-to-use entry point for AES crypto processed by cryptographic service provider.

Public Member Functions

- byte[ ] Encrypt (byte[ ] data, byte[ ] aesKey)

  PIV CRYPTO SUPPORT ===================.

6.34.1 Detailed Description

Easy-to-use entry point for AES crypto processed by cryptographic service provider.

6.34.2 Member Function Documentation

6.34.2.1 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCryptoAes.Encrypt (byte[ ] *data*, byte[ ] *aesKey*) [inline]

PIV CRYPTO SUPPORT ===================.

Parameters

data	.
aesKey	.

Returns

A byte[].

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardCryptoAes.cs

6.35 SmartCardAPI.CardModule.PIV.PivCardCryptoHash Class Reference

Helper class for cryptographic operations requiring hash

Public Member Functions

- **PivCardCryptoHash** (byte algoId=PIVC.HASH_ALGOID_SHA256)
- byte **GetAlgoId** ()
- void **SetAlgoId** (byte hashAlgorithmIdentifier)
- byte[ ] GetDigestInfo ()

  Get ASN1 prefix required for padding using hash. ref: `https://niclabs.cl/tchsm/doxygen/algorithms__pkcs1__encoding_8c_source.html` *this digest info is typically used for PKCS #1 padding schemes*
- byte[ ] ComputeFileHash (string fileName)

  Calculates hash of a file using managed .NET crypto.
- byte[ ] ComputeHash (string text)

  Calculates hash of text. This can only be done off-card. The hash type is set during initialization of this class. Currently SHA1 and SHA256 are supported;
- byte[ ] ComputeHash (byte[ ] data)

  Calculates hash of a byte array using a FIPS approved implementation.

6.35.1 Detailed Description

Helper class for cryptographic operations requiring hash

6.35.2 Member Function Documentation

6.35.2.1 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCryptoHash.ComputeFileHash (string *fileName*) [inline]

Calculates hash of a file using managed .NET crypto.

Exceptions

Exception	Thrown when an exception error condition occurs.

Parameters

fileName	open file stream.

Returns

Hash. Langth depends oh hash type

6.35.2.2 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCryptoHash.ComputeHash (string *text*) [inline]

Calculates hash of text. This can only be done off-card. The hash type is set during initialization of this class. Currently SHA1 and SHA256 are supported;

Exceptions

Exception	Thrown when an exception error condition occurs.

Parameters

text	.

Returns

Hash.

6.35.2.3 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCryptoHash.ComputeHash (byte[ ] *data*) [inline]

Calculates hash of a byte array using a FIPS approved implementation.

Exceptions

Exception	Thrown when an exception error condition occurs.

Parameters

data	byte array.

Returns

Hash, length depends on hash type

6.35.2.4 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCryptoHash.GetDigestInfo ( ) [inline]

Get ASN1 prefix required for padding using hash. ref: `https://niclabs.cl/tchsm/doxygen/algorithms↩ __pkcs1__encoding_8c_source.html` this digest info is typically used for PKCS #1 padding schemes

Exceptions

Exception	Thrown when an exception error condition occurs.

Returns

An array of byte.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardCryptoHash.cs

6.36 SmartCardAPI.CardModule.PIV.PivCardCryptoRsa Class Reference

Easy-to-use entry point for RSA crypto processed by cryptographic service provider.

Public Member Functions

- byte[ ] Encrypt (byte[ ] message, byte[ ] publicKey)

  Encrypt byte array using PIV card public key residing in host-side certificate container accessed through cryptographic service provider.

- byte[ ] RSAEncrypt (byte[ ] DataToEncrypt, string ContainerName, byte[ ] publicKey, bool DoOAEPPadding)

  The RSACryptoServiceProvider supports key lengths from 384 bits to 16384 bits in increments of 8 bits if you have // the Microsoft Enhanced Cryptographic Provider installed. It supports key lengths from 384 bits to 512 bits in // increments of 8 bits if you have the Microsoft Base Cryptographic Provider installed.
- void RSAPersistKeyInCSP (string ContainerName)

  Create a persistent RSA key in CSP.
- void RSADeleteKeyInCSP (string ContainerName)

  Delete RSA key in CSP.

6.36.1 Detailed Description

Easy-to-use entry point for RSA crypto processed by cryptographic service provider.

6.36.2 Member Function Documentation

6.36.2.1 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCryptoRsa.Encrypt (byte[ ] *message,* byte[ ] *publicKey*) [inline]

Encrypt byte array using PIV card public key residing in host-side certificate container accessed through cryptographic service provider.

Parameters

message	The message.
publicKey	.

Returns

A byte[].

6.36.2.2 void SmartCardAPI.CardModule.PIV.PivCardCryptoRsa.RSADeleteKeyInCSP (string *ContainerName*) [inline]

Delete RSA key in CSP.

Parameters

ContainerName	.

6.36.2.3 byte [ ] SmartCardAPI.CardModule.PIV.PivCardCryptoRsa.RSAEncrypt (byte[ ] *DataToEncrypt,* string *ContainerName,* byte[ ] *publicKey,* bool *DoOAEPPadding*) [inline]

The RSACryptoServiceProvider supports key lengths from 384 bits to 16384 bits in increments of 8 bits if you have // the Microsoft Enhanced Cryptographic Provider installed. It supports key lengths from 384 bits to 512 bits in // increments of 8 bits if you have the Microsoft Base Cryptographic Provider installed.

Parameters

DataToEncrypt	.
ContainerName	.
publicKey	.
DoOAEPPadding	.

Returns

A byte[].

6.36.2.4 void SmartCardAPI.CardModule.PIV.PivCardCryptoRsa.RSAPersistKeyInCSP (string *ContainerName*) [inline]

Create a persistent RSA key in CSP.

Parameters

ContainerName	.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardCryptoRsa.cs

6.37 SmartCardAPI.CardModule.PIV.PivCardEdge Class Reference

Exposes high level PIV card edge in line with End-Point client application programming interface defined in SP 800-73-4. ref: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-73-4.pdf` Method names are conform with 800-73-4 as much as possible to indicate the exact PIV API middleware function targeted with this implementation.

Public Member Functions

- PivCardEdge (CardHandle hCard)

  Constructor for high-level PIV card access.
- void Connect ()

  (NOT IMPLEMENTED) pivConnect() entry point as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3. Use SmartCard API instead.
- bool ChangeReferenceData (int keyReference, byte[ ] oldKey, byte[ ] newKey)

  Allow use of CHANGE REFERENCE DATA command.
- void Disconnect ()

  (NOT IMPLEMENTED) pivDisconnect() entry point as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3. Use SmartCard API instead.
- byte[ ] GenerateKeyPair (byte keyReference, byte algorithmIdentifier)

  Entry point for credential initialization and administration as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3. Generates a public/private key pair.
- void LogIntoCardApplication ()

  pivLogIntoCardApplication() entry point for data access as defined in PIV End-Point Client- Application Programming Interface SP 800-73-3 not implemented.
- void LogOutOfCardApplication ()

  pivLogOutOfCardApplication() entry point for data access as defined in PIV End-Point Client- Application Programming Interface SP 800-73-3 not implemented.
- bool VerifyPin (string pin, ref int remainingTries)

  PIN verification method. If string is shorter than 8 digits, this method applies 0xFF padding on byte-level. This method is only meant for user PIN. It does not support global PINs, PUKs SO PINs or transport PINs Internal PIN reference is always 0x80 Exammple: "1234" becomes 0x31323334FFFFFFFF.
- int Verify (int keyReference, byte[ ] pin)

PIN verification method. If string is shorter than 8 digits, this method applies 0xFF padding on byte-level. This method is onl;y meant for user PIN. It does not support global PINs, PUKs SO PINs or transport PINs Note, that VERIFY command can use key references other than '00', '80', '96', '97', and '98', if they are used for card management operations. Internal PIN reference is always 0x80 Exammple: "1234" becomes 0x31323334FFFFFFFF.

- string GetVersion ()

  For SP 800-73-3 Part 3 conformant PIV Middleware this method would just return “800-73-3 Client API”. We add Card↩ Werk to the string to indicate a proprietary API version. Our goal though is to implement a PIV API close to SP 800-73-3 specs.

- bool SelectApplet (byte[ ] aid)

  Selects application on a multi-application card. The AID can be an RID or RID + PIX. This method can be used to explore the card and find out what applets are loaded in case the host application wants to support multiple card configurations. This method can also be used as main entry point for data access as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3 Note that cards might select a default applet at power-up. Selecting a different applet might change scope, exposing a different card edge.

- byte[ ] SelectCardApplication ()

  ∗∗∗∗∗∗∗∗∗∗∗ DEPRECIATED ∗∗∗∗∗∗∗∗∗∗∗∗∗ pivSelectCardApplication() entry point for data access as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3 Selects PIV application on a multi-application card. This is only needed on cards that don't select the PIV application at power-up. ∗∗∗∗∗∗∗∗∗∗∗ this method will be removed in future releases ∗∗∗∗∗∗∗∗∗∗∗∗.

- byte[ ] Crypt (byte algorithmIdentifier, byte keyReference, byte[ ] algorithmInput)

  Crypt() is the entry point for GENERAL AUTHENTICATE on-card cryptographic operations as defined in PIV End-Point Client-API SP 800-73-4. This includes building and parsing the necessary challenge/response authentication templates for encryption/signing.

- byte[ ] GetData (byte[ ] vbObjectId)

  PIV GetData() entry point for data access as defined in PIV End-Point Client-Application Programming Interface SP 800-73-4

- bool PutData (byte[ ] vbObjectId, byte[ ] data)

  Entry point for credential initialization and administration as defined in PIV End-Point Client-Application Programming Interface of SP 800-73-3. This method adds header and trailer data necessary for PIV data containers. Make sure to have proper authentication in place to store new certificates.

Properties

- CardHandle cardHandle `[get]`

  Gets the handle of the card.

- string LastError `[get]`

  Gets the last error.

6.37.1 Detailed Description

Exposes high level PIV card edge in line with End-Point client application programming interface defined in SP 800-73-4. ref: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-73-4.pdf` Method names are conform with 800-73-4 as much as possible to indicate the exact PIV API middleware function targeted with this implementation.

6.37.2 Constructor & Destructor Documentation

6.37.2.1 SmartCardAPI.CardModule.PIV.PivCardEdge.PivCardEdge (**CardHandle** ***hCard***) `[inline]`

Constructor for high-level PIV card access.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

hCard	The card.

6.37.3 Member Function Documentation

6.37.3.1 bool SmartCardAPI.CardModule.PIV.PivCardEdge.ChangeReferenceData (int *keyReference,* byte[ ] *oldKey,* byte[ ] *newKey*) [inline]

Allow use of CHANGE REFERENCE DATA command.

Parameters

keyReference	key references other than '80' and '81', if they are used for card management operations.
oldKey	.
newKey	.

Returns

True if it succeeds, false if it fails.

6.37.3.2 void SmartCardAPI.CardModule.PIV.PivCardEdge.Connect ( ) [inline]

(NOT IMPLEMENTED) pivConnect() entry point as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3. Use SmartCard API instead.

Exceptions

PivCardException	Thrown when a Piv Card error condition occurs.

6.37.3.3 byte [ ] SmartCardAPI.CardModule.PIV.PivCardEdge.Crypt (byte *algorithmIdentifier,* byte *keyReference,* byte[ ] *algorithmInput*) [inline]

Crypt() is the entry point for GENERAL AUTHENTICATE on-card cryptographic operations as defined in PIV End-Point Client-API SP 800-73-4. This includes building and parsing the necessary challenge/response authentication templates for encryption/signing.

SP 800-73-4 Part 1, Appendix C describes recommended procedures for PIV algorithm identifier discovery.

All cryptographic operations, except SM on the client side, are performed outside the PIV Middleware i.e this method is THE entry point to all ON-CARD cryptographic operations..

Parameters

algorithm↩ Identifier	Identifier of the cryptographic algorithm to be used for the cryptographic operation. [SP800 - 78, Tables 6 - 2 and 6 - 3]
keyReference	Identifier of the on-card key to be used for the cryptographic operation. See [SP800-78, Table 6-1] and Part 1, Table 4b.
algorithmInput	Sequence of bytes used as the input to the cryptographic operation.The algorithmInput for RSA algorithms shall be restricted to the range 0 to n-1, where n is the RSA modulus.

Exceptions

PivCardException	Thrown when a PIV Card error condition occurs.

Returns

byte[] - Sequence of bytes output by the cryptographic operation. Null if error occurs

6.37.3.4 void SmartCardAPI.CardModule.PIV.PivCardEdge.Disconnect ( ) [inline]

(NOT IMPLEMENTED) pivDisconnect() entry point as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3. Use SmartCard API instead.

Exceptions

PivCardException	Thrown when a Piv Card error condition occurs.

6.37.3.5 byte [ ] SmartCardAPI.CardModule.PIV.PivCardEdge.GenerateKeyPair (byte *keyReference,* byte *algorithmIdentifier*) [inline]

Entry point for credential initialization and administration as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3. Generates a public/private key pair.

Parameters

keyReference	key references other than '80' and '81', if they are used for card management operations.
algorithm↩ Identifier	Identifier for the algorithm.

Returns

An array of byte.

6.37.3.6 byte [ ] SmartCardAPI.CardModule.PIV.PivCardEdge.GetData (byte[ ] *vbObjectId*) [inline]

PIV GetData() entry point for data access as defined in PIV End-Point Client-Application Programming Interface SP 800-73-4

Reads a data object from a PIV card (data) container according to Special Publication 800-73: Data such as Card Holder Unique IDentifier (CHUID) or X509 certificates can be read without prior authentication whereas containers with biometric data require user confirmation (PIN)

GetData returns the complete raw container data including TLV headers and check bytes if applicable.

Expiration check and CHUID signature check (optional) RID 'A0 00 00 00 01 16', container ID = 0x3000 <remark> We do not use ASN.1 object identifiers (OIDs) in the personal verification subset of the OIDs managed by NIST but rather the ObjectID as defined on APDU level. </remark>

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
Exception	Thrown when an exception error condition occurs.

Parameters

vbObjectId	Object ID as defined in PIV standard.

Returns

Byte array with raw container data including TLV header and check byte.

6.37.3.7 string SmartCardAPI.CardModule.PIV.PivCardEdge.GetVersion ( ) [inline]

For SP 800-73-3 Part 3 conformant PIV Middleware this method would just return “800-73-3 Client API”. We add Card↩ Werk to the string to indicate a proprietary API version. Our goal though is to implement a PIV API close to SP 800-73-3 specs.

Returns

The version.

6.37.3.8 void SmartCardAPI.CardModule.PIV.PivCardEdge.LogIntoCardApplication ( ) [inline]

pivLogIntoCardApplication() entry point for data access as defined in PIV End-Point Client- Application Programming Interface SP 800-73-3 not implemented.

Exceptions

PivCardException	Thrown when a Piv Card error condition occurs.

6.37.3.9 void SmartCardAPI.CardModule.PIV.PivCardEdge.LogOutOfCardApplication ( ) [inline]

pivLogOutOfCardApplication() entry point for data access as defined in PIV End-Point Client- Application Programming Interface SP 800-73-3 not implemented.

Exceptions

PivCardException	Thrown when a Piv Card error condition occurs.

6.37.3.10 bool SmartCardAPI.CardModule.PIV.PivCardEdge.PutData (byte[ ] *vbObjectId,* byte[ ] *data*) [inline]

Entry point for credential initialization and administration as defined in PIV End-Point Client-Application Programming Interface of SP 800-73-3. This method adds header and trailer data necessary for PIV data containers. Make sure to have proper authentication in place to store new certificates.

Parameters

vbObjectId	Object ID as defined in PIV standard.
data	raw data without header and footer info.

Returns

true if data was written successfully to card.

6.37.3.11 bool SmartCardAPI.CardModule.PIV.PivCardEdge.SelectApplet (byte[ ] *aid*) [inline]

Selects application on a multi-application card. The AID can be an RID or RID + PIX. This method can be used to explore the card and find out what applets are loaded in case the host application wants to support multiple card configurations. This method can also be used as main entry point for data access as defined in PIV End-Point Client-↩ Application Programming Interface SP 800-73-3 Note that cards might select a default applet at power-up. Selecting a different applet might change scope, exposing a different card edge.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

aid	The aid.

Returns

True if it succeeds, false if it fails.

6.37.3.12 byte [ ] SmartCardAPI.CardModule.PIV.PivCardEdge.SelectCardApplication ( ) [inline]

∗∗∗∗∗∗∗∗∗∗∗ DEPRECIATED ∗∗∗∗∗∗∗∗∗∗∗∗∗ pivSelectCardApplication() entry point for data access as defined in PIV End-Point Client-Application Programming Interface SP 800-73-3 Selects PIV application on a multi-application card. This is only needed on cards that don't select the PIV application at power-up. ∗∗∗∗∗∗∗∗∗∗∗ this method will be removed in future releases ∗∗∗∗∗∗∗∗∗∗∗∗.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
PivCardException	Thrown when a Piv Card error condition occurs.

Returns

A byte[].

6.37.3.13 int SmartCardAPI.CardModule.PIV.PivCardEdge.Verify (int *keyReference,* byte[ ] *pin*) [inline]

PIN verification method. If string is shorter than 8 digits, this method applies 0xFF padding on byte-level. This method is onl;y meant for user PIN. It does not support global PINs, PUKs SO PINs or transport PINs Note, that VERIFY command can use key references other than '00', '80', '96', '97', and '98', if they are used for card management operations. Internal PIN reference is always 0x80 Exammple: "1234" becomes 0x31323334FFFFFFFF.

Parameters

keyReference	key references other than '80' and '81', if they are used for card management operations.
pin	PIN value; up to 8 numeric digits {0,1,2,3,4,5,6,7,8,9}

Returns

An int.

6.37.3.14 bool SmartCardAPI.CardModule.PIV.PivCardEdge.VerifyPin (string *pin,* ref int *remainingTries*) [inline]

PIN verification method. If string is shorter than 8 digits, this method applies 0xFF padding on byte-level. This method is only meant for user PIN. It does not support global PINs, PUKs SO PINs or transport PINs Internal PIN reference is always 0x80 Exammple: "1234" becomes 0x31323334FFFFFFFF.

Exceptions

Exception	Thrown when an exception error condition occurs.

Parameters

pin	PIN value; up to 8 numeric digits {0,1,2,3,4,5,6,7,8,9}
remainingTries	[in,out] remaining tries - only updated if PIN verification fails.

Returns

True if it succeeds, false if it fails.

6.37.4 Property Documentation

6.37.4.1 CardHandle SmartCardAPI.CardModule.PIV.PivCardEdge.cardHandle [get]

Gets the handle of the card.

The card handle.

6.37.4.2 string SmartCardAPI.CardModule.PIV.PivCardEdge.LastError [get]

Gets the last error.

The last error.

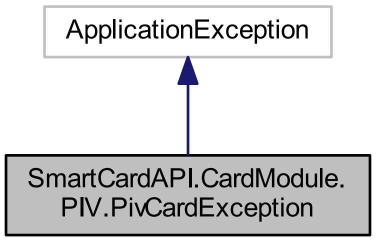
The documentation for this class was generated from the following files:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardEdge.cs
- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardEdge_Crypt.cs
- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardEdge_GetData.cs
- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardEdge_PutData.cs

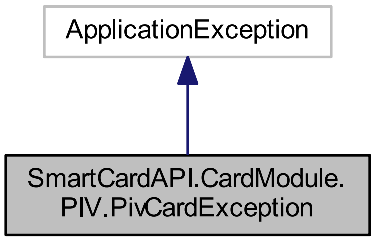
6.38 SmartCardAPI.CardModule.PIV.PivCardException Class Reference

Whenever a SmartCardAPI.CardModule.PIV API error is returned an exception of this class is thrown.

Inheritance diagram for SmartCardAPI.CardModule.PIV.PivCardException:



Collaboration diagram for SmartCardAPI.CardModule.PIV.PivCardException:



Public Member Functions

- PivCardException (PivCardErrorCode nErrorCode)

  Creates a new PivCardException instance from a SmartCardAPI.CardModule.PIV API error code.
- override string ToString ()

  Builds a string that describes the PIV API error that caused this exception.

Properties

- PivCardErrorCode ErrorCode `[get]`

  The error code that was returned by the failing SmartCardAPI.CardModule.PIV API call.

6.38.1 Detailed Description

Whenever a SmartCardAPI.CardModule.PIV API error is returned an exception of this class is thrown.

See also

T:System.ApplicationException

6.38.2 Constructor & Destructor Documentation

6.38.2.1 SmartCardAPI.CardModule.PIV.PivCardException.PivCardException (PivCardErrorCode *nErrorCode*) [inline]

Creates a new PivCardException instance from a SmartCardAPI.CardModule.PIV API error code.

Parameters

nErrorCode	The error indicating integer value returned by a SmartCardAPI.CardModule.PIV API call. Use a type cast to coerce the integer value into a PivCardErrorCode type.

6.38.3 Member Function Documentation

6.38.3.1 override string SmartCardAPI.CardModule.PIV.PivCardException.ToString ( ) [inline]

Builds a string that describes the PIV API error that caused this exception.

Returns

Friendly description of an error on PIV API level.

See also

M:System.Exception.ToString()

6.38.4 Property Documentation

6.38.4.1 PivCardErrorCode SmartCardAPI.CardModule.PIV.PivCardException.ErrorCode [get]

The error code that was returned by the failing SmartCardAPI.CardModule.PIV API call.

The error code.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardException.cs

6.39 SmartCardAPI.CardModule.PIV.PivCardGzip Class Reference

This class exposes method required to compress/decompress PIV card data.

Static Public Member Functions

- static int ReadAllBytesFromStream (Stream stream, byte[ ] buffer)

  Reads all bytes from stream.
- static byte[ ] Compress (String fileName)

Compresses data residing in a file using GZIP algorhithm.

- static byte[ ] Decompress (byte[ ] compressedData)

  Converts compressed byte array to an uncompressed array. Note that with GZIP you might end up with a smaller file size after decompression.

6.39.1 Detailed Description

This class exposes method required to compress/decompress PIV card data.

6.39.2 Member Function Documentation

6.39.2.1 static byte [ ] SmartCardAPI.CardModule.PIV.PivCardGzip.Compress (String *fileName*) `[inline]`, `[static]`

Compresses data residing in a file using GZIP algorhithm.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentException	Thrown when one or more arguments have unsupported or illegal values.

Parameters

fileName	name of the file containing data.

Returns

A byte[].

6.39.2.2 static byte [ ] SmartCardAPI.CardModule.PIV.PivCardGzip.Decompress (byte[ ] *compressedData*) `[inline]`, `[static]`

Converts compressed byte array to an uncompressed array. Note that with GZIP you might end up with a smaller file size after decompression.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.
ArgumentException	Thrown when one or more arguments have unsupported or illegal values.

Parameters

compressedData	byte array to be decompressed.

Returns

A byte[].

6.39.2.3 static int SmartCardAPI.CardModule.PIV.PivCardGzip.ReadAllBytesFromStream (Stream *stream*, byte[ ] *buffer*) `[inline]`, `[static]`

Reads all bytes from stream.

Parameters

stream	The stream.
buffer	The buffer.

Returns

all bytes from stream.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardGzip.cs

6.40 SmartCardAPI.CardModule.PIV.PivCardPrintedInformation Class Reference

This is a helper class to parse, analyze and display PIV PrintedInformation data. It processes data that typically resides in the Printed Information data containter. This class allows easy access to any element within the PIV Printed Information container. Ref SP 800-73- 3 However, it does not require a physical card connection.

Public Member Functions

- PivCardPrintedInformation (byte[ ] printedInformationData)

  Constructor.
- override string ToString ()

  Converts the printed information to a nice human readable string. Using one line per non-zero data item. This is for quick and dirty implementations.

Properties

- string Name `[get]`

  Allows public access to card holder name.
- string EmployeeAffiliation `[get]`

  Allows public access to printed Employee Affiliation.
- string EmployeeAffiliation_Line2 `[get]`

  Allows public access to printed Employee Affiliation Line 2.
- string ExpirationDate `[get]`

  Allows public access to printed Expiration Date.
- string AgencyCardSerialNumber `[get]`

  Allows public access to printed Agency Card Serial Number.
- string IssuerIdentification `[get]`

  Allows public access to printed Issuer Identification.
- string OrganizationAffiliation_Line1 `[get]`

  Allows public access to printed OrganizationAffiliation (line1)
- string OrganizationAffiliation_Line2 `[get]`

  Allows public access to printed OrganizationAffiliation (line2)

6.40.1 Detailed Description

This is a helper class to parse, analyze and display PIV PrintedInformation data. It processes data that typically resides in the Printed Information data containter. This class allows easy access to any element within the PIV Printed Information container. Ref SP 800-73- 3 However, it does not require a physical card connection.

Testes with following PIV cards:

- OCS PIV card NIST cert #668
- Athena PIV NIST cert#991
- Keycorp PIV NIST cert #1055
- OCS PIV End Point v1.08, #23 SHA-1
- OCS PIV End Point v1.08, #35 SHA-256
- OCS PIV End Point v1.08, #36 SHA-256.

6.40.2 Constructor & Destructor Documentation

6.40.2.1 SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.PivCardPrintedInformation (byte[ ] *printedInformationData*) [inline]

Constructor.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

printed↩ InformationData	Information describing the printed information.

6.40.3 Member Function Documentation

6.40.3.1 override string SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.ToString ( ) [inline]

Converts the printed information to a nice human readable string. Using one line per non-zero data item. This is for quick and dirty implementations.

Returns

Returns a nicely formatted string usable for tracing output.

See also

M:System.Object.ToString()

6.40.4 Property Documentation

6.40.4.1 string SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.AgencyCardSerialNumber `[get]`

Allows public access to printed Agency Card Serial Number.

The agency card serial number.

6.40.4.2 string SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.EmployeeAffiliation `[get]`

Allows public access to printed Employee Affiliation.

The employee affiliation.

6.40.4.3 string SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.EmployeeAffiliation_Line2 `[get]`

Allows public access to printed Employee Affiliation Line 2.

The employee affiliation line 2.

6.40.4.4 string SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.ExpirationDate `[get]`

Allows public access to printed Expiration Date.

The expiration date.

6.40.4.5 string SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.IssuerIdentification `[get]`

Allows public access to printed Issuer Identification.

The issuer identification.

6.40.4.6 string SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.Name `[get]`

Allows public access to card holder name.

The name.

6.40.4.7 string SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.OrganizationAffiliation_Line1 `[get]`

Allows public access to printed OrganizationAffiliation (line1)

The organization affiliation line 1.

6.40.4.8 string SmartCardAPI.CardModule.PIV.PivCardPrintedInformation.OrganizationAffiliation_Line2 `[get]`

Allows public access to printed OrganizationAffiliation (line2)

The organization affiliation line 2.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩
CardPrintedInformation.cs

6.41 SmartCardAPI.CardModule.PIV.PivCardSecurityObject Class Reference

Security Object ASN.1 OID=2.16.840.1.101.3.7.2.144.0 This is a helper class to parse, analyze and display PIV security object data. This method does not require a physical card connection. The Security Object is in accordance with Appendix C of PKI for Machine Readable Travel Documents Offering ICC Read-Only Access Version 1.1. [5] Tag “0x←BA” is used to map the ContainerIDs in the PIV data model to the 16 Data Groups specified in the Machine Readable Travel Document (MRTD). The mapping enables the Security Object to be fully compliant for future activities with identity documents.

Public Member Functions

- PivCardSecurityObject (byte[ ] containerData)

  Constructor.
- override string ToString ()

  Stringifies data, in a nice human readable fashion.

Properties

- byte[ ] MappingOfDGtoContainerID `[get]`

  Allows public access to Mapping of DG to ContainerID.
- byte[ ] SecurityObject `[get]`

  Allows public access to Security Object.
- byte[ ] RawData `[get]`

  Allows public access to the raw data of the Security Object container.

6.41.1 Detailed Description

Security Object ASN.1 OID=2.16.840.1.101.3.7.2.144.0 This is a helper class to parse, analyze and display PIV security object data. This method does not require a physical card connection. The Security Object is in accordance with Appendix C of PKI for Machine Readable Travel Documents Offering ICC Read-Only Access Version 1.1. [5] Tag “0x←BA” is used to map the ContainerIDs in the PIV data model to the 16 Data Groups specified in the Machine Readable Travel Document (MRTD). The mapping enables the Security Object to be fully compliant for future activities with identity documents.

The “DG-number-to-Container-ID” mapping object TLV in tag “0xBA”encapsulates a series of three byte triples - one for each PIV data object included in the Security Object. The first byte is the Data Group (DG) number, and the second and third bytes are the most and least significant bytes (respectively) of the Container ID value. The DG number assignment is arbitrary; however, the same number assignment applies to the DataGroupNumber(s) in the DataGroupHash(es). This will ensure that the ContainerIDs in the mapping object refers to the correct hash value in the Security Object (0xBB).

The 0xBB Security Object is formatted according to the MRTD document's Appendix C. The LDS Security Object itself must be in ASN.1 DER format, formatted as specified in Appendix C.2. This structure is then inserted into the encap←ContentInfo field of the CMS object specified in Appendix C.1

The card issuer's digital signature key used to sign the CHUID shall also be used to sign the Security Object. The signature field of the Security Object, Tag “0xBB” shall omit the issuer’s certificate, since it is included in the CHUID. The three optional unsigned data elements 1) Printed Information data object, 2) Unsigned CHUID, and 3) Facial Image data object shall be included in the Security Object3 if present. ref: SP800-73-2.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 SmartCardAPI.CardModule.PIV.PivCardSecurityObject.PivCardSecurityObject (byte[ ] *containerData*) [inline]

Constructor.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

containerData	Information describing the container.

6.41.3 Member Function Documentation

6.41.3.1 override string SmartCardAPI.CardModule.PIV.PivCardSecurityObject.ToString ( ) `[inline]`

Stringifies data, in a nice human readable fashion.

Returns

Returns a nicely formatted string usable for tracing output.

See also

M:System.Object.ToString()

6.41.4 Property Documentation

6.41.4.1 byte [ ] SmartCardAPI.CardModule.PIV.PivCardSecurityObject.MappingOfDGtoContainerID `[get]`

Allows public access to Mapping of DG to ContainerID.

The identifier of the mapping of d gto container.

6.41.4.2 byte [ ] SmartCardAPI.CardModule.PIV.PivCardSecurityObject.RawData `[get]`

Allows public access to the raw data of the Security Object container.

Information describing the raw.

6.41.4.3 byte [ ] SmartCardAPI.CardModule.PIV.PivCardSecurityObject.SecurityObject `[get]`

Allows public access to Security Object.

The security object.

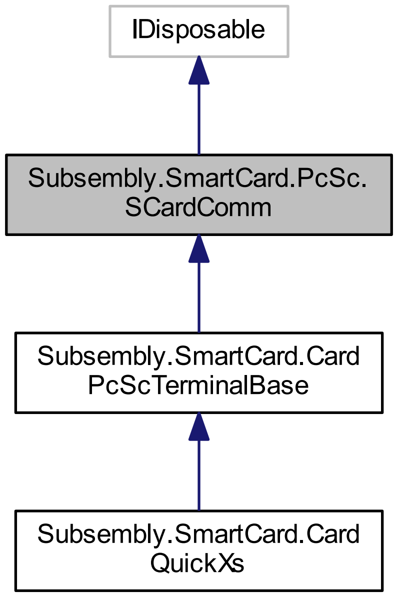
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.CardModule/PIV/Piv↩ CardSecurityObject.cs

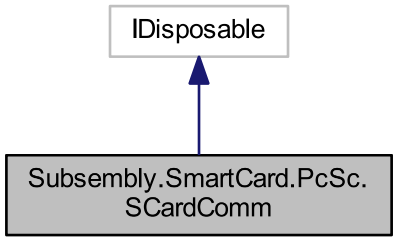
6.42 Subsembly.SmartCard.PcSc.SCardComm Class Reference

This object encapsulates a communication interface to a specific card or reader. It provides methods for managing the connections, controlling transactions, sending and receiving commands, and determining card state information.

Inheritance diagram for Subsembly.SmartCard.PcSc.SCardComm:



Collaboration diagram for Subsembly.SmartCard.PcSc.SCardComm:



Public Member Functions

- SCardComm (SCardResourceManager aResourceManager)

  Constructs this Smart Card Communication instance.
- void Dispose ()

Disposes this Smart Card Communication context.

- bool Connect (string sReader, SCardAccessMode nAccessMode, SCardProtocolIdentifiers nPreferredProtocols)

  Opens a connection to the card located in the reader identified by the sReader parameter.
- bool Reconnect (SCardAccessMode nAccessMode, SCardProtocolIdentifiers nPreferredProtocols, SCard↩ Disposition nInitialization)

  Reconnect to card already associated with this context by a previous Connect. Reconnect possibly resets the card.
- void Disconnect (SCardDisposition nDisposition)

  Disconnect from card. This releases all card resources, sets internal handles to null.
- SCardReaderState Status ()

  Returns the current state of the reader. Can be for PC/SC connection refresh.
- SCardReaderState Status (out byte[ ] vbATR)

  Returns the current state of the reader and the ATR of the card. This method calls SCardStatus() internally - a function recommended to refresh a pending card transaction that exceeds 5 seconds.
- void BeginTransaction ()

  This method initiates a logical transaction against the card associated with this context. If successful, this blocks other applications from accessing the card, allowing the calling application to perform a sequence of operations against the card with assurance that any intermediate state information will remain valid.
- void EndTransaction (SCardDisposition nDisposition)

  This method ends a logical transaction sequence against the card associated with this context. If successful, this frees the transaction and the next pending transaction, if any, is serviced.
- void Transmit (byte[ ] vbSendBuffer, out byte[ ] vbRecvBuffer)

  Transmits the given command APDU to the card and returns the response APDU from the card.
- void Control (int nControlCode, byte[ ] vbInBuffer, out byte[ ] vbOutBuffer)

  This method supports direct communication with the reader device. Its primary intent is to provide a mechanism to communicate with vendor-defined features. It is the responsibility of the vendor to define Control Code values and input/output data associated with these features.
- bool GetReaderCapabilities (SCardReaderCapability nTag, out byte[ ] vbValue)

  This method returns a reader attribute value associated with the supplied nTag parameter (see SCardReaderCapability enumeration for defined values).
- string GetReaderCapabilitiesString (SCardReaderCapability nTag)

  This method returns a reader attribute value associated with the supplied nTag parameter converted into a string.
- int GetReaderCapabilitiesInteger (SCardReaderCapability nTag, int nDefaultValue)

  This method returns a reader attribute value associated with the supplied nTag parameter converted into a convenient integer value.
- int GetReaderCapabilitiesByte (SCardReaderCapability nTag)

  This method returns a reader attribute byte value associated with the supplied nTag parameter converted into a convenient integer value.
- bool SetReaderCapabilities (SCardReaderCapability nTag, byte[ ] vbValue)

  This method sets a reader attribute value. The attribute to be set is identified by the nTag parameter (see SCardReader↩ Capability enumeration for defined values). (see Part 3 for defined values) and the value to set is passed in Buffer. If the attribute does not exist or cannot be set, or the provided value is illegal an error is returned.
- void VerifyPinStart (SCardPinBlock aPinBlock, byte bTimeOut, byte bTimeOut2, byte bEntryValidationCondition, byte[ ] vbData)

  Sends a FEATURE_VERIFY_PIN_START control command to the reader.
- void ModifyPinStart (SCardPinBlock aPinBlock, byte bTimeOut, byte bTimeOut2, byte bInsertionOffsetOld, byte bInsertionOffsetNew, byte bConfirmPIN, byte bEntryValidationCondition, byte[ ] vbData)

  Sends a FEATURE_MODIFY_PIN_START control command to the reader.
- int VerifyPinFinish ()

  Sends a FEATURE_VERIFY_PIN_FINISH control command to the reader.
- int ModifyPinFinish ()

Sends a FEATURE_MODIFY_PIN_FINISH control command to the reader.

- byte GetKeyPressed ()

  Sends a FEATURE_GET_KEY_PRESSED control command to the reader.
- void Abort ()

  Sends a FEATURE_ABORT control command to the reader.
- int VerifyPinDirect (SCardPinBlock aPinBlock, byte bTimeOut, byte bTimeOut2, byte bEntryValidationCondition, byte[ ] vbData)

  Sends a FEATURE_VERIFY_PIN_DIRECT control command to the reader.
- int ModifyPinDirect (SCardPinBlock aPinBlock, byte bTimeOut, byte bTimeOut2, byte bInsertionOffsetOld, byte bInsertionOffsetNew, byte bConfirmPIN, byte bEntryValidationCondition, byte[ ] vbData)

  Sends a FEATURE_MODIFY_PIN_DIRECT control command to the reader.
- byte[ ] MctReaderDirect (byte[ ] vbMctIn)

  Sends a FEATURE_MCT_READERDIRECT control command to the reader.
- byte[ ] MctUniversal (ref byte nSAD, ref byte nDAD, byte[ ] vbCmdAPDU)

  Sends a FEATURE_MCT_UNIVERSAL control command to the reader.

Static Public Member Functions

- static int SCardCtlCode (int nFunctionCode)

  Static helper that computes the control code for Control just like the C++ macro SCARD_CTL_CODE.

Properties

- bool InTransaction `[get]`

  Indicates whether there is a current transaction that was opened by BeginTransaction.
- bool IsConnected `[get]`

  Indicates whether there is a current connection to a smart card that was created by Connect.
- string ConnectedReader `[get]`

  Provides the name of the currently connected reader or `null` if there is no open connection.
- SCardProtocolIdentifiers ActiveProtocol `[get]`

  Tells the currently active protocol.
- string VendorName `[get]`

  The name of the card reader manufacturer.
- string VendorIFDType `[get]`

  The name of the card reader type according to the manufacturer.
- Version VendorIFDVersion `[get]`

  Returns the version number according to the manufacturer.
- string VendorIFDSerialNo `[get]`

  The serial number of the card reader according to the manufacturer.
- SCardChannelType ChannelType `[get]`

  The type of channel that the card reader is connected to.
- int ChannelNumber `[get]`

  The channel number that the card reader is connected to.
- SCardMechanicalCharacteristics MechanicalCharacteristics `[get]`

  Indicates which mechanical characteristics are supported by the connected reader.
- bool IsReinerSCT `[get]`

  Indicates whether the connected reader is from the manufacturer REINER SCT.

- bool IsOmnikey `[get]`

  Indicates whether the connected reader is from the manufacturer HID Global.
- SCardAuthDevices UserToCardAuthDevice `[get]`

  User to Card Authentication Devices. Unsigned four byte integer that is the result of a bitwise OR operation performed on the applicable values from the enumeration SCardAuthDevices.
- SCardAuthDevices UserAuthInputDevice `[get]`

  User Authentication Input Device. Unsigned four byte integer that is the result of a bitwise OR operation performed on the applicable values from the enumeration SCardAuthDevices.
- bool VerifyModifyStarted `[get]`

  Indicates whether there is an active VerifyPinStart or ModifyPinStart operation.
- int FeatureVerifyPinStart `[get]`

  Provides the FEATURE_VERIFY_PIN_START control code.
- int FeatureVerifyPinFinish `[get]`

  Provides the FEATURE_VERIFY_PIN_FINISH control code.
- int FeatureModifyPinStart `[get]`

  Provides the FEATURE_MODIFY_PIN_START control code.
- int FeatureModifyPinFinish `[get]`

  Provides the FEATURE_MODIFY_PIN_FINISH control code.
- int FeatureGetKeyPressed `[get]`

  Provides the FEATURE_GET_KEY_PRESSED control code.
- int FeatureVerifyPinDirect `[get]`

  Provides the FEATURE_VERIFY_PIN_DIRECT control code.
- int FeatureModifyPinDirect `[get]`

  Provides the FEATURE_MODIFY_PIN_DIRECT control code.
- int FeatureMctReaderDirect `[get]`

  Provides the FEATURE_MCT_READERDIRECT control code.
- int FeatureMctUniversal `[get]`

  Provides the FEATURE_MCT_UNIVERSAL control code.
- int FeatureIfdPinProp `[get]`

  Provides the FEATURE_IFD_PIN_PROP control code.
- int FeatureAbort `[get]`

  Provides the FEATURE_ABORT control code.
- int FeatureSetSpeMessage `[get]`

  Provides the FEATURE_SET_SPE_MESSAGE control code.
- int FeatureVerifyPinDirectAppId `[get]`

  Provides the FEATURE_PIN_DIRECT_CONTROL control code.
- int FeatureModifyPinDirectAppId `[get]`

  Provides the FEATURE_MODIFY_PIN_DIRECT_APP_ID control code.
- int FeatureWriteDisplay `[get]`

  Provides the FEATURE_WRITE_DISPLAY control code.
- int FeatureGetKey `[get]`

  Provides the FEATURE_GET_KEY control code.
- int FeatureIfdDisplayProperties `[get]`

  Provides the FEATURE_DISPLAY_PROPERTIES control code.
- int FeatureGetTlvProperties `[get]`

  Provides the FEATURE_GET_TLV_PROPERTIES control code.
- int FeatureCcidEscCommand `[get]`

Provides the FEATURE_CCID_ESC_COMMAND control code.

- IntPtr NativePcscCardHandle `[get]`

  Access to PC/SC card handle necessary to use some third-party API extensions built directly on top of PC/SC. Example: OMNIKEY scardsyn.dll WARNING: DO NOT USE THIS ENTRYPOINT UNLESS YOU HAVE TO. 09DEC2013 MJ WORK IN PROGRESS/ EXPERIMENTAL.

- int PinPropLcdLayoutLines `[get]`

  Indicates the number of lines of the card reader LCD.

- int PinPropLcdLayoutChars `[get]`

  Indicates the number of characters on a line of the card reader LCD.

- byte PinPropEntryValidationCondition `[get]`

  Indicates which EntryValidationCondition flags are supported by the reader.

- bool PinPropTimeOut2 `[get]`

  Indicates whether the TimeOut2 is supported.

6.42.1 Detailed Description

This object encapsulates a communication interface to a specific card or reader. It provides methods for managing the connections, controlling transactions, sending and receiving commands, and determining card state information.

The command interfaces are designed to simplify interaction with a card using the protocols required by the PC/SC Specification. In particular, the usage of the protocols ISO/IEC T=0 and T=1. In addition, a raw mode is supported, which may be used to support arbitrary data exchange protocols (such as T=14) for special-purpose requirements.

This class implements the SCARDCOMM class of the PC/SC Workgroups architecture. See part 5 "ICC Resource Manager Definition" section 3.2.5 of the PC/SC Workgroup Specification. The PC/SC Workgroup Specification is available at the `PC/SC Workgroups homepage`.

Starting with version 4.0, this class also implements part 10 "IFDs with Secure Pin Entry Capabilities" of the PC/SC Workgroup Specification version 2.0. Methods related to secure PIN entry are not supported in the Windows CE version of this implementation.

See also

T:System.IDisposable

6.42.2 Constructor & Destructor Documentation

6.42.2.1 Subsembly.SmartCard.PcSc.SCardComm.SCardComm (SCardResourceManager *aResourceManager*) `[inline]`

Constructs this Smart Card Communication instance.

Exceptions

ArgumentNullException	Parameter aResourceManager was null.

Parameters

aResource←Manager	Reference to the SCardResourceManager instance.

6.42.3 Member Function Documentation

6.42.3.1 void Subsembly.SmartCard.PcSc.SCardComm.Abort ( ) [inline]

Sends a FEATURE_ABORT control command to the reader.

Exceptions

NotSupportedException	Thrown when the requested operation is not supported.

6.42.3.2 void Subsembly.SmartCard.PcSc.SCardComm.BeginTransaction ( ) [inline]

This method initiates a logical transaction against the card associated with this context. If successful, this blocks other applications from accessing the card, allowing the calling application to perform a sequence of operations against the card with assurance that any intermediate state information will remain valid.

At present, only a single outstanding transaction may exist for a given card. If a transaction is underway when an application calls this service, then this method will block pending completion of the prior transaction(s). Application requests for transactions are serviced on a first-in, first-out basis.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	There is no existing connection.

6.42.3.3 bool Subsembly.SmartCard.PcSc.SCardComm.Connect (string *sReader*, SCardAccessMode *nAccessMode*, SCardProtocolIdentifiers *nPreferredProtocols*) [inline]

Opens a connection to the card located in the reader identified by the sReader parameter.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown if there is already an existing connection.
InvalidProgramException	Thrown when an Invalid Program error condition occurs.

Parameters

sReader	Name of card reader to check for a card. This must be a known reader (for example listed on PCSC level). This name must be obtained from one of the overloaded versions of the SCard↵ResourceManager.ListReaders method.
nAccessMode	

The nAccessMode parameter is used to indicate two things. First, it indicates whether the connection is opened for SCardAccessMode.Shared or SCardAccessMode.Exclusive access. If the requested mode is unavailable, an exception is raised.

Second, it indicates whether the caller desires a SCardAccessMode.Direct connection to the reader. Direct mode implies that a connection will be established, even if a card is not present. Direct mode also implies exclusive access to the reader.

The only Access Mode supported by Windows CE is SCardAccessMode.Exclusive access!

Parameters

nPreferred↵Protocols	

Indicates the preferred smart card protocol to use when communicating with the smart card. Currently the only supported protocols are T=0 and T=1.

This parameter may be SCardProtocolIdentifiers.Undefined only if nAccessMode is set to SCardAccessMode.Direct. In this case, no protocol negotiation will be performed until an IOCTL_SMARTCARD_SET_PROTOCOL control directive is sent with Control.

Communication with the card will be initialized only for one of the valid protocols identified by the nPreferredProtocols bit mask. If none of the requested protocols is available, then an exception is raised. The actual protocol in use is available through ActiveProtocol property.

Returns

Returns true if there was a card presentation and we could successfully connect to this card. Returns false if the user did not insert a card or otherwise manually aborted this operation. If communication with the card fails for some other reason, then an exception is thrown.

6.42.3.4 void Subsembly.SmartCard.PcSc.SCardComm.Control (int *nControlCode,* byte[ ] *vbInBuffer,* out byte[ ] *vbOutBuffer*) [inline]

This method supports direct communication with the reader device. Its primary intent is to provide a mechanism to communicate with vendor-defined features. It is the responsibility of the vendor to define Control Code values and input/output data associated with these features.

Marc, 1/19/2020.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	There is no existing connection.
InvalidProgramException	Thrown when an Invalid Program error condition occurs.

Parameters

nControlCode	The control code of the desired operation. Use the static helper SCardCtlCode in order to build a control code from a function code similar to the SCARD_CTL_CODE macro.
vbInBuffer	Vendor defined input data for the given Control Code. This is directly sent to the driver. This parameter can be null if the nControlCode specifies an operation that does not need input data.
vbOutBuffer	[out] Vendor defined response data for the given Control Code. This is the data returned directly from the driver. This may be returned as null if no response data is expected by te given n↩ ControlCode.

6.42.3.5 void Subsembly.SmartCard.PcSc.SCardComm.Disconnect (**SCardDisposition** *nDisposition*) [inline]

Disconnect from card. This releases all card resources, sets internal handles to null.

Marc, 1/19/2020.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	There is no existing connection to be disconnected.

Parameters

nDisposition	

Indicates what to do with the card. See the SCardDisposition enumeration for possible disposition actions.

6.42.3.6 void Subsembly.SmartCard.PcSc.SCardComm.Dispose ( ) [inline]

Disposes this Smart Card Communication context.

This method may be called regardless of a card was previously connected or not. If no card was connected, or the card was already disconnected, then this method does nothing.

See also

M:System.IDisposable.Dispose()

6.42.3.7 void Subsembly.SmartCard.PcSc.SCardComm.EndTransaction (SCardDisposition *nDisposition*) [inline]

This method ends a logical transaction sequence against the card associated with this context. If successful, this frees the transaction and the next pending transaction, if any, is serviced.

Marc, 1/19/2020.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	There is no existing transaction.

Parameters

nDisposition	Upon termination, the action indicated by nDisposition is performed if possible. The allowed disposition actions are SCardDisposition.LeaveCard, SCardDisposition.ResetCard, SCard↩ Disposition.UnpowerCard, and SCardDisposition.EjectCard.

6.42.3.8 byte Subsembly.SmartCard.PcSc.SCardComm.GetKeyPressed ( ) [inline]

Sends a FEATURE_GET_KEY_PRESSED control command to the reader.

No key was pressed at all.

The correction/backspace key was pressed.

The CLEAR button was pressed, the complete PIN entry so far has been cleared.

The OK or ENTER button was pressed.

A timeout occured and the EntryValidationCondition flags did request that the PIN is accepted by a timeout.

The CANCEL button was pressed. PIN entry shall be aborted.

One of the digit buttons was pressed.

A timeout has led to abortion of the PIN entry or the parameters given to VerifyPinStart or ModifyPinStart have been rejected by the card reader.

Exceptions

NotSupportedException	Thrown when the requested operation is not supported.

Returns

Returns a byte that indicates the type of key (if any) that was pressed at the card reader. The individual values are defined by the following table:
<list type="table"> <item> 0x00

6.42.3.9 bool Subsembly.SmartCard.PcSc.SCardComm.GetReaderCapabilities (**SCardReaderCapability** *nTag,* out byte[ ] *vbValue*) [inline]

This method returns a reader attribute value associated with the supplied nTag parameter (see SCardReaderCapability enumeration for defined values).

The Win32 Smart Card API names this method SCardGetAttrib. Here the correct name from the PC/SC Workgroup Specification is used.

See part 3 "Requirements for PC-Connected Interface Devices" of the PC/SC Workgroup Specification. The PC/SC Workgroup Specification is available at the `PC/SC Workgroups homepage`.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	There is no existing connection.

Parameters

nTag	Reader capabilities tag. If the tag is unknown, then false is returned.
vbValue	[out] Receives the Reader Capabilities value or null is the given Tag is not known.

Returns

If the given Reader Capabilities Tag is known, then `true` is returned. If the given Tag is unknown, then `false` is returned.

6.42.3.10 int Subsembly.SmartCard.PcSc.SCardComm.GetReaderCapabilitiesByte (**SCardReaderCapability** *nTag*) [inline]

This method returns a reader attribute byte value associated with the supplied nTag parameter converted into a convenient integer value.

See part 3 "Requirements for PC-Connected Interface Devices" of the PC/SC Workgroup Specification. The PC/SC Workgroup Specification is available at the `PC/SC Workgroups homepage`.

Parameters

nTag	Reader Capabilities Tag to query. See SCardReaderCapability enumeration for defined values.

Returns

Returns the Reader Capability byte value as a convenient integer value in the range from 0 through 255. It is assumed that the returned Reader Capability byte array contains nothing but a single byte. If the requested nTag is not supported by this Reader, then -1 is returned instead.

6.42.3.11 int Subsembly.SmartCard.PcSc.SCardComm.GetReaderCapabilitiesInteger (**SCardReaderCapability** *nTag,* int *nDefaultValue*) [inline]

This method returns a reader attribute value associated with the supplied nTag parameter converted into a convenient integer value.

See part 3 "Requirements for PC-Connected Interface Devices" of the PC/SC Workgroup Specification. The PC/SC Workgroup Specification is available at the `PC/SC Workgroups homepage`.

Parameters

nTag	Reader Capabilities Tag to query. See SCardReaderCapability enumeration for defined values.
nDefaultValue	Provides the default value that will be returned if the requested nTag is not supported by this Reader.

Returns

Returns the Reader Capability value as a convenient integer value. It is assumed that the returned Reader Capability byte array contains exactly four bytes in little-endian encoding. If the requested nTag is not supported by this Reader, then the value of the nDefaultValue parameter is returned instead.

6.42.3.12 string Subsembly.SmartCard.PcSc.SCardComm.GetReaderCapabilitiesString (SCardReaderCapability *nTag*) [inline]

This method returns a reader attribute value associated with the supplied nTag parameter converted into a string.

See part 3 "Requirements for PC-Connected Interface Devices" of the PC/SC Workgroup Specification. The PC/SC Workgroup Specification is available at the `PC/SC Workgroups homepage`

02JUN2020 MJ fix: reader without serial number returns "?" (changed to empty string "") change: return "" if capability is not available

Parameters

nTag	Reader Capabilities Tag to query. See SCardReaderCapability enumeration for defined values.

Returns

Returns the Reader Capability value as a convenient string. If the given nTag is not supported by the Reader an empty string is returned. The returned string is constructed by interpreting the returned byte array as an ISO Latin 1 encoded string (Windows codepage 1252).

6.42.3.13 byte [ ] Subsembly.SmartCard.PcSc.SCardComm.MctReaderDirect (byte[ ] *vbMctIn*) [inline]

Sends a FEATURE_MCT_READERDIRECT control command to the reader.

Marc, 1/19/2020.

Exceptions

NotSupportedException	Thrown when the requested operation is not supported.

Parameters

vbMctIn	Data to send directly to the reader. Usually this is a CT-BCS conforming command APDU.

Returns

Returns a response APDU generated by the card reader.

6.42.3.14 byte [ ] Subsembly.SmartCard.PcSc.SCardComm.MctUniversal (ref byte *nSAD,* ref byte *nDAD,* byte[ ] *vbCmdAPDU*) [inline]

Sends a FEATURE_MCT_UNIVERSAL control command to the reader.

This method sends a CT-API or CT-BCS command to the connected reader and/or card. Please refer to the CT-API and CT-BCS specification for a description of the commands that can be sent through this method.

Exceptions

NotSupportedException	The FEATURE_MCT_UNIVERSAL control code is not supported. Read the FeatureMctUniversal property in order to check whether the FEATURE_MCT_U↩ NIVERSAL control code is supported before calling this method.
ApplicationException	The data returned by the FEATURE_MCT_UNIVERSAL control is malformed.

Parameters

nSAD	[in,out] Source Address according to the CT-API specification.
nDAD	[in,out] Destination Address according to the CT-API specification.
vbCmdAPDU	The command APDU to send. This can contain a command for the smart card, or a CT-BCS command interpreted by the card reader.

Returns

Returns the response APDU either from the smart card or from the card reader.

6.42.3.15 int Subsembly.SmartCard.PcSc.SCardComm.ModifyPinDirect (SCardPinBlock *aPinBlock*, byte *bTimeOut*, byte *bTimeOut2*, byte *bInsertionOffsetOld*, byte *bInsertionOffsetNew*, byte *bConfirmPIN*, byte *bEntryValidationCondition*, byte[ ] *vbData*) `[inline]`

Sends a FEATURE_MODIFY_PIN_DIRECT control command to the reader.

Marc, 1/19/2020.

Exceptions

ArgumentNullException	The parameter aPinBlock or vbData was `null`.
NotSupportedException	The FEATURE_MODIFY_PIN_DIRECT control block is not supported. Read the FeatureModifyPinDirect property in order to check whether the FEATURE_MODI↩ FY_PIN_DIRECT control block is supported before calling this method.

Parameters

aPinBlock	Instance of SCardPinBlock that supplies various parameters that control how the PIN(s) entered by the user will be inserted into the command template provided by the vbData parameter.
bTimeOut	Timeout in seconds. If zero, then the default timeout is used.
bTimeOut2	Timeout in seconds after first key stroke. The property PinPropTimeOut2 indicates whether this parameter is supported.
bInsertion↩ OffsetOld	Byte offset where the PIN block for the old PIN shall be inserted into the command APD↩ U template.
bInsertion↩ OffsetNew	Byte offset where the PIN block for the new PIN shall be inserted into the command APDU template.
bConfirmPIN	Bit field that indicates which PIN entries are requested for confirmation of the PIN modification. The bit 0x01 indicates that the new PIN must be entered twice for confirmation. If this bit is not set, then the new PIN must be entered only once. The bit 0x02 indicates that the old PIN must be entered first. If this bit is not set, then the user only has to enter the new PIN.

bEntry↵ Validation↵ Condition	Flags that select the conditions that complete the PIN entry. See the property PinPropEntry↵ ValidationCondition for details.
vbData	The APDU command template that shall be used when constructing the MODIFY command. Must not be `null`.

Returns

Returns the Status Word (SW) that was returned in the two bytes of response data from the FEATURE_MODIF↵ Y_PIN_DIRECT control command.

6.42.3.16 int Subsembly.SmartCard.PcSc.SCardComm.ModifyPinFinish ( ) `[inline]`

Sends a FEATURE_MODIFY_PIN_FINISH control command to the reader.

Marc, 1/19/2020.

Exceptions

NotSupportedException	Thrown when the requested operation is not supported.

Returns

Returns the Status Word (SW) that was returned in the two bytes of response data from the FEATURE_MODIF↵ Y_PIN_FINISH control command.

6.42.3.17 void Subsembly.SmartCard.PcSc.SCardComm.ModifyPinStart (SCardPinBlock *aPinBlock*, byte *bTimeOut*, byte *bTimeOut2*, byte *bInsertionOffsetOld*, byte *bInsertionOffsetNew*, byte *bConfirmPIN*, byte *bEntryValidationCondition*, byte[ ] *vbData*) `[inline]`

Sends a FEATURE_MODIFY_PIN_START control command to the reader.

Marc, 1/19/2020.

Exceptions

ArgumentNullException	The parameter aPinBlock or vbData was `null`.
NotSupportedException	The FEATURE_MODIFY_PIN_START control block is not supported. Read the FeatureModifyPinStart property in order to check whether the FEATURE_MODI↵ FY_PIN_START control block is supported before calling this method.

Parameters

aPinBlock	Instance of SCardPinBlock that supplies various parameters that control how the PIN(s) entered by the user will be inserted into the command template provided by the vbData parameter.
bTimeOut	Timeout in seconds. If zero, then the default timeout is used.
bTimeOut2	Timeout in seconds after first key stroke. The property PinPropTimeOut2 indicates whether this parameter is supported.

bInsertion↩ OffsetOld	Byte offset where the PIN block for the old PIN shall be inserted into the command APD↩ U template.
bInsertion↩ OffsetNew	Byte offset where the PIN block for the new PIN shall be inserted into the command APDU template.
bConfirmPIN	Bit field that indicates which PIN entries are requested for confirmation of the PIN modification. The bit 0x01 indicates that the new PIN must be entered twice for confirmation. If this bit is not set, then the new PIN must be entered only once. The bit 0x02 indicates that the old PIN must be entered first. If this bit is not set, then the user only has to enter the new PIN.
bEntry↩ Validation↩ Condition	Flags that select the conditions that complete the PIN entry. See the property PinPropEntry↩ ValidationCondition for details.
vbData	The APDU command template that shall be used when constructing the MODIFY command. Must not be `null`.

6.42.3.18 bool Subsembly.SmartCard.PcSc.SCardComm.Reconnect (SCardAccessMode *nAccessMode,* SCardProtocolIdentifiers *nPreferredProtocols,* SCardDisposition *nInitialization*) `[inline]`

Reconnect to card already associated with this context by a previous Connect. Reconnect possibly resets the card.

Note that if Reconnect fails then this object instance is still in the the connected state. Hence the condition can only be cleared by disposing the object or by successfully reconnecting to a card.

This method may be used to clear an error condition preventing access to the card. For example, if the card is reset by another application, then further attempts to execute methods against that card will fail with a warning being returned. This method allows the application to acknowledge the reset notification and continue operation.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown if there is no existing connection to reconnect.
ArgumentException	Thrown when one or more arguments have unsupported or illegal values.
InvalidProgramException	Thrown when an Invalid Program error condition occurs.

Parameters

nAccessMode	

Indicates the new Access Mode that is desired. This method can be used to change the Access Mode on an existing connection. The new Access Mode can be SCardAccessMode.Shared or SCardAccessMode.Exclusive access. If the requested mode is unavailable, an exception is raised. Note that the Access Mode SCardAccessMode.Direct cannot be specified here.

Parameters

nPreferred↩ Protocols	

Indicates the preferred smart card protocol to use when communicating with the smart card. Only the protocols T=0 and T=1 may be specified here!

Parameters

nInitialization	

The nInitialization parameter allows to specify a desired action on the card as a side effect of the Reconnect. This may include performing a warm or cold reset by specifying either the SCardDisposition.ResetCard or SCardDisposition.↩ UnpowerCard standard disposition codes.

Use SCardDisposition.LeaveCard to leave the card as is.

This parameter is ignored if the Reconnect is applied to a connection that would normally return one of the return codes SCARD_W_RESET_CARD, SCARD_W_REMOVED_CARD and may be others dependent of the target platform. This must be done to prevent race conditions in applications.

Do not use SCardDisposition.EjectCard or other standard disposition codes not explicitly allowed above with this method!

Returns

Returns true if we are successfully connected to the card now. Returns false if the user did not insert a card or otherwise manually aborted this operation. If communication with the card fails for some other reason, then an exception is thrown.

6.42.3.19 static int Subsembly.SmartCard.PcSc.SCardComm.SCardCtlCode (int *nFunctionCode*) [inline], [static]

Static helper that computes the control code for Control just like the C++ macro SCARD_CTL_CODE.

Parameters

nFunctionCode	Code value that would otherwise be given to the SCARD_CTL_CODE macro.

Returns

Computed code value to be passed to Control.

6.42.3.20 bool Subsembly.SmartCard.PcSc.SCardComm.SetReaderCapabilities (SCardReaderCapability *nTag*, byte[ ] *vbValue*) [inline]

This method sets a reader attribute value. The attribute to be set is identified by the nTag parameter (see SCard↩ReaderCapability enumeration for defined values). (see Part 3 for defined values) and the value to set is passed in Buffer. If the attribute does not exist or cannot be set, or the provided value is illegal an error is returned.

Marc, 1/19/2020.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	There is no existing connection.
InvalidProgramException	Thrown when an Invalid Program error condition occurs.

Parameters

nTag	Reader capabilities tag. If the tag is unknown, then false is returned.
vbValue	The value to be set is passed in this buffer.

Returns

If the given Reader Capabilities Tag is known, then true is returned. If the given Tag is unknown, then false is returned.

6.42.3.21 SCardReaderState Subsembly.SmartCard.PcSc.SCardComm.Status ( ) [inline]

Returns the current state of the reader. Can be for PC/SC connection refresh.

Marc, 1/19/2020.

Returns

A SCardReaderState.

6.42.3.22 SCardReaderState Subsembly.SmartCard.PcSc.SCardComm.Status (out byte[ ] *vbATR*) [inline]

Returns the current state of the reader and the ATR of the card. This method calls SCardStatus() internally - a function recommended to refresh a pending card transaction that exceeds 5 seconds.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.

Parameters

vbATR	[out] If a card is inserted and the native function call returned ATR data, then it is returned to the caller through this parameter. If no ATR is available then the parameter is returned as `null`. The Status method of the PC/SC Workgroup Specification returns much more information than just the state. However, this additional information is available through various properties of this class.

Returns

Returns a value from the SCardReaderState enumeration that indicates the current state of the reader.

6.42.3.23 void Subsembly.SmartCard.PcSc.SCardComm.Transmit (byte[ ] *vbSendBuffer*, out byte[ ] *vbRecvBuffer*) [inline]

Transmits the given command APDU to the card and returns the response APDU from the card.

Marc, 1/19/2020.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	There is no existing connection.
InvalidProgramException	Thrown when an Invalid Program error condition occurs.

Parameters

vbSendBuffer	Provides the complete command APDU. The entire array will be sent to the card.
vbRecvBuffer	[out] Receives the complete response APDU. The size of the returned array equals the number of data bytes received from the card.

6.42.3.24 int Subsembly.SmartCard.PcSc.SCardComm.VerifyPinDirect (SCardPinBlock *aPinBlock*, byte *bTimeOut*, byte *bTimeOut2*, byte *bEntryValidationCondition*, byte[ ] *vbData*) [inline]

Sends a FEATURE_VERIFY_PIN_DIRECT control command to the reader.

Marc, 1/19/2020.

Exceptions

ArgumentNullException	The parameter aPinBlock or vbData was `null`.
NotSupportedException	The FEATURE_VERIFY_PIN_DIRECT control block is not supported. Read the FeatureVerifyPinDirect property in order to check whether the FEATURE_VERI↩ FY_PIN_DIRECT control block is supported before calling this method.

Parameters

aPinBlock	Instance of SCardPinBlock that supplies various parameters that control how the PIN entered by the user will be inserted into the command template provided by the vbData parameter.
bTimeOut	Timeout in seconds. If zero, then the default timeout is used.
bTimeOut2	Timeout in seconds after first key stroke. The property PinPropTimeOut2 indicates whether this parameter is supported.
bEntry↩ Validation↩ Condition	Flags that select the conditions that complete the PIN entry. See the property PinPropEntry↩ ValidationCondition for details.
vbData	The APDU command template that shall be used when constructing the VERIFY command. Must not be `null`.

Returns

Returns the Status Word (SW) that was returned in the two bytes of response data from the FEATURE_VERIF↩ Y_PIN_DIRECT control command.

6.42.3.25 int Subsembly.SmartCard.PcSc.SCardComm.VerifyPinFinish ( ) `[inline]`

Sends a FEATURE_VERIFY_PIN_FINISH control command to the reader.

Marc, 1/19/2020.

Exceptions

NotSupportedException	Thrown when the requested operation is not supported.

Returns

Returns the Status Word (SW) that was returned in the two bytes of response data from the FEATURE_VERIF↩ Y_PIN_FINISH control command.

6.42.3.26 void Subsembly.SmartCard.PcSc.SCardComm.VerifyPinStart (SCardPinBlock *aPinBlock*, byte *bTimeOut*, byte *bTimeOut2*, byte *bEntryValidationCondition*, byte[ ] *vbData*) `[inline]`

Sends a FEATURE_VERIFY_PIN_START control command to the reader.

Marc, 1/19/2020.

Exceptions

ArgumentNullException	The parameter aPinBlock or vbData was `null`.
NotSupportedException	The FEATURE_VERIFY_PIN_START control block is not supported. Read the FeatureVerifyPinStart property in order to check whether the FEATURE_VERIF↩ Y_PIN_START control block is supported before calling this method.

Parameters

aPinBlock	Instance of SCardPinBlock that supplies various parameters that control how the PIN entered by the user will be inserted into the command template provided by the vbData parameter.
bTimeOut	Timeout in seconds. If zero, then the default timeout is used.
bTimeOut2	Timeout in seconds after first key stroke. The property PinPropTimeOut2 indicates whether this parameter is supported.
bEntry↩ Validation↩ Condition	Flags that select the conditions that complete the PIN entry. See the property PinPropEntry↩ ValidationCondition for details.
vbData	The APDU command template that shall be used when constructing the VERIFY command. Must not be `null`.

6.42.4 Property Documentation

6.42.4.1 SCardProtocolIdentifiers Subsembly.SmartCard.PcSc.SCardComm.ActiveProtocol `[get]`

Tells the currently active protocol.

This property value remains unchanged when the card was removed. It is updated only inside Connect and Reconnect. Before the first card was successfully connected to, this property contains SCardProtocolIdentifiers.Undefined.

The active protocol.

6.42.4.2 int Subsembly.SmartCard.PcSc.SCardComm.ChannelNumber `[get]`

The channel number that the card reader is connected to.

The interpretation of the channel number depends on the type of channel that the card reader is connected to. The channel type can be determined through the SCardComm.ChannelType property.

There must have been a successful Connect before this property may be read. Otherwise an InvalidOperationException will be thrown.

The channel number.

See also

ChannelType

6.42.4.3 SCardChannelType Subsembly.SmartCard.PcSc.SCardComm.ChannelType `[get]`

The type of channel that the card reader is connected to.

There must have been a successful Connect before this property may be read. Otherwise an InvalidOperationException will be thrown.

The type of the channel.

6.42.4.4 string Subsembly.SmartCard.PcSc.SCardComm.ConnectedReader `[get]`

Provides the name of the currently connected reader or `null` if there is no open connection.

The connected reader.

6.42.4.5 int Subsembly.SmartCard.PcSc.SCardComm.FeatureAbort [get]

Provides the FEATURE_ABORT control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.6 int Subsembly.SmartCard.PcSc.SCardComm.FeatureCcidEscCommand [get]

Provides the FEATURE_CCID_ESC_COMMAND control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.7 int Subsembly.SmartCard.PcSc.SCardComm.FeatureGetKey [get]

Provides the FEATURE_GET_KEY control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.8 int Subsembly.SmartCard.PcSc.SCardComm.FeatureGetKeyPressed [get]

Provides the FEATURE_GET_KEY_PRESSED control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.9 int Subsembly.SmartCard.PcSc.SCardComm.FeatureGetTlvProperties [get]

Provides the FEATURE_GET_TLV_PROPERTIES control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.10 int Subsembly.SmartCard.PcSc.SCardComm.FeatureIfdDisplayProperties [get]

Provides the FEATURE_DISPLAY_PROPERTIES control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.11 int Subsembly.SmartCard.PcSc.SCardComm.FeatureIfdPinProp [get]

Provides the FEATURE_IFD_PIN_PROP control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.12 int Subsembly.SmartCard.PcSc.SCardComm.FeatureMctReaderDirect [get]

Provides the FEATURE_MCT_READERDIRECT control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.13 int Subsembly.SmartCard.PcSc.SCardComm.FeatureMctUniversal [get]

Provides the FEATURE_MCT_UNIVERSAL control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.14 int Subsembly.SmartCard.PcSc.SCardComm.FeatureModifyPinDirect `[get]`

Provides the FEATURE_MODIFY_PIN_DIRECT control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.15 int Subsembly.SmartCard.PcSc.SCardComm.FeatureModifyPinDirectAppId `[get]`

Provides the FEATURE_MODIFY_PIN_DIRECT_APP_ID control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.16 int Subsembly.SmartCard.PcSc.SCardComm.FeatureModifyPinFinish `[get]`

Provides the FEATURE_MODIFY_PIN_FINISH control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.17 int Subsembly.SmartCard.PcSc.SCardComm.FeatureModifyPinStart `[get]`

Provides the FEATURE_MODIFY_PIN_START control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.18 int Subsembly.SmartCard.PcSc.SCardComm.FeatureSetSpeMessage `[get]`

Provides the FEATURE_SET_SPE_MESSAGE control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.19 int Subsembly.SmartCard.PcSc.SCardComm.FeatureVerifyPinDirect `[get]`

Provides the FEATURE_VERIFY_PIN_DIRECT control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.20 int Subsembly.SmartCard.PcSc.SCardComm.FeatureVerifyPinDirectAppId `[get]`

Provides the FEATURE_PIN_DIRECT_CONTROL control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.21 int Subsembly.SmartCard.PcSc.SCardComm.FeatureVerifyPinFinish `[get]`

Provides the FEATURE_VERIFY_PIN_FINISH control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.22 int Subsembly.SmartCard.PcSc.SCardComm.FeatureVerifyPinStart `[get]`

Provides the FEATURE_VERIFY_PIN_START control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.23 int Subsembly.SmartCard.PcSc.SCardComm.FeatureWriteDisplay `[get]`

Provides the FEATURE_WRITE_DISPLAY control code.

If the feature is not supported, then the default value zero will be returned.

6.42.4.24 bool Subsembly.SmartCard.PcSc.SCardComm.InTransaction `[get]`

Indicates whether there is a current transaction that was opened by BeginTransaction.

True if in transaction, false if not.

6.42.4.25 bool Subsembly.SmartCard.PcSc.SCardComm.IsConnected `[get]`

Indicates whether there is a current connection to a smart card that was created by Connect.

True if this object is connected, false if not.

6.42.4.26 bool Subsembly.SmartCard.PcSc.SCardComm.IsOmnikey `[get]`

Indicates whether the connected reader is from the manufacturer HID Global.

This check is used internally in order to activate some special handling for RF field and reader control with HID OMNI↵KEY smart card readers.

True if this object is omnikey, false if not.

6.42.4.27 bool Subsembly.SmartCard.PcSc.SCardComm.IsReinerSCT `[get]`

Indicates whether the connected reader is from the manufacturer REINER SCT.

This check is used internally in order to activate some special handling for secure PIN entry with REINER SCT smart card readers.

True if this object is reiner sct, false if not.

6.42.4.28 SCardMechanicalCharacteristics Subsembly.SmartCard.PcSc.SCardComm.MechanicalCharacteristics `[get]`

Indicates which mechanical characteristics are supported by the connected reader.

Note that multiple bits can be set in the returned flags.

There must have been a successfull Connect before this property may be read. Otherwise an InvalidOperationException will be thrown.

The mechanical characteristics.

See also

SCardMechanicalCharacteristics

6.42.4.29 IntPtr Subsembly.SmartCard.PcSc.SCardComm.NativePcscCardHandle [get]

Access to PC/SC card handle necessary to use some third-party API extensions built directly on top of PC/SC. Example: OMNIKEY scardsyn.dll WARNING: DO NOT USE THIS ENTRYPOINT UNLESS YOU HAVE TO. 09DE↩C2013 MJ WORK IN PROGRESS/ EXPERIMENTAL.

The native pcsc card handle.

6.42.4.30 byte Subsembly.SmartCard.PcSc.SCardComm.PinPropEntryValidationCondition [get]

Indicates which EntryValidationCondition flags are supported by the reader.

PIN entry is implicitly finished when the maximum number of digits has been entered.

PIN entry is finished when the OK button on the card reader is pressed.

PIN entry is finished when the timeout occurs. Note that this means that the PIN is accepted when the timeout occurs instead of aborting the PIN entry.

This value is a bit field containing the following flags:

<list type="table"> <item> 0x01

6.42.4.31 int Subsembly.SmartCard.PcSc.SCardComm.PinPropLcdLayoutChars [get]

Indicates the number of characters on a line of the card reader LCD.

The value is zero if no LCD is present. Otherwise it is in the range from 1 through 255.

6.42.4.32 int Subsembly.SmartCard.PcSc.SCardComm.PinPropLcdLayoutLines [get]

Indicates the number of lines of the card reader LCD.

The value is zero if no LCD is present. Otherwise it is in the range from 1 through 255.

6.42.4.33 bool Subsembly.SmartCard.PcSc.SCardComm.PinPropTimeOut2 [get]

Indicates whether the TimeOut2 is supported.

True if pin property time out 2, false if not.

6.42.4.34 SCardAuthDevices Subsembly.SmartCard.PcSc.SCardComm.UserAuthInputDevice [get]

User Authentication Input Device. Unsigned four byte integer that is the result of a bitwise OR operation performed on the applicable values from the enumeration SCardAuthDevices.

The user authentication input device.

6.42.4.35 SCardAuthDevices Subsembly.SmartCard.PcSc.SCardComm.UserToCardAuthDevice [get]

User to Card Authentication Devices. Unsigned four byte integer that is the result of a bitwise OR operation performed on the applicable values from the enumeration SCardAuthDevices.

The user to card authentication device.

6.42.4.36 string Subsembly.SmartCard.PcSc.SCardComm.VendorIFDSerialNo [get]

The serial number of the card reader according to the manufacturer.

There must have been a successful Connect before this property may be read. Otherwise an InvalidOperationException will be thrown.

According to the PC/SC Workgroup specification the returned name is an ASCII string with has at most 32 characters.

The vendor ifd serial no.

6.42.4.37 string Subsembly.SmartCard.PcSc.SCardComm.VendorIFDType [get]

The name of the card reader type according to the manufacturer.

There must have been a successful Connect before this property may be read. Otherwise an InvalidOperationException will be thrown.

According to the PC/SC Workgroup specification the returned name is an ASCII string which has at most 32 characters.

The type of the vendor ifd.

6.42.4.38 Version Subsembly.SmartCard.PcSc.SCardComm.VendorIFDVersion [get]

Returns the version number according to the manufacturer.

There must have been a successful Connect before this property may be read. Otherwise an InvalidOperationException will be thrown.

Only the Major, Minor, and Build property of the Version instance are initialised from the information returned by the PC/SC API.

6.42.4.39 string Subsembly.SmartCard.PcSc.SCardComm.VendorName [get]

The name of the card reader manufacturer.

There must have been a successful Connect before this property may be read. Otherwise an InvalidOperationException will be thrown.

According to the PC/SC Workgroup specification the returned name is an ASCII string which has at most 32 characters.

The name of the vendor.

6.42.4.40 bool Subsembly.SmartCard.PcSc.SCardComm.VerifyModifyStarted [get]

Indicates whether there is an active VerifyPinStart or ModifyPinStart operation.

This flag is set by a successful call to VerifyPinStart or ModifyPinStart. It is reset when either of VerifyPinFinish, Modify↩ PinFinish, or Abort is called.

True if verify modify started, false if not.

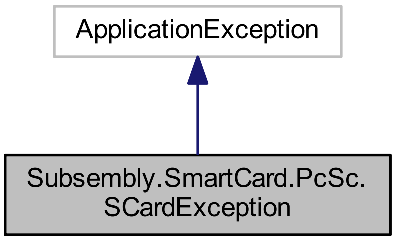
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/PcSc/SCard↩ Comm.cs

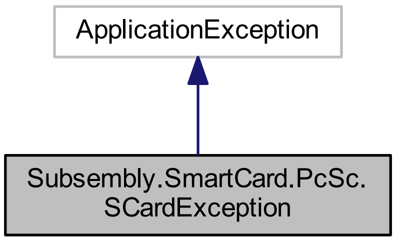
6.43 Subsembly.SmartCard.PcSc.SCardException Class Reference

Exception for signalling smart card API errors.

Inheritance diagram for Subsembly.SmartCard.PcSc.SCardException:



Collaboration diagram for Subsembly.SmartCard.PcSc.SCardException:



Public Member Functions

- SCardException (SCardResponseCode nResponseCode)

    Constructor.

Properties

- SCardResponseCode ResponseCode `[get]`

    The error code.
- override string Message `[get]`

    Overridden in order to create a message that includes the ResponseCode.

6.43.1 Detailed Description

Exception for signalling smart card API errors.

See also

T:System.ApplicationException

6.43.2 Constructor & Destructor Documentation

6.43.2.1 Subsembly.SmartCard.PcSc.SCardException.SCardException (**SCardResponseCode** *nResponseCode*) `[inline]`

Constructor.

Parameters

nResponseCode	.

6.43.3 Property Documentation

6.43.3.1 override string Subsembly.SmartCard.PcSc.SCardException.Message `[get]`

Overridden in order to create a message that includes the ResponseCode.

The message.

See also

P:System.Exception.Message

6.43.3.2 **SCardResponseCode** Subsembly.SmartCard.PcSc.SCardException.ResponseCode `[get]`

The error code.

The response code.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/PcSc/SCard↩ Exception.cs

6.44 Subsembly.SmartCard.PcSc.SCardPinBlock Class Reference

Representation of the PC/SC PINBLOCK structure as defined in PC/SC 2.0 part 9 "IFDs with Extended Capabilities".

Public Attributes

- SCardPinAttributes Attributes

  Refers to the PIN overwritting format as defined by the SCardPinAttributes constants. Can be combined with a binary OR.
- int PINBlockLength

Length of the PINBlock in bytes.

- int NbCharMin

  Minimum number of characters to be a valid PIN value.
- int NbCharMax

  Maximum number of characters to be a valid PIN value.
- int PINBitOffset

  Position in bits in the APDU data field to overwrite with the formatted PIN. It is a position from the left if it is SCardPin↩ Attributes.LeftJustified and a position from right for SCardPinAttributes.RightJustified.
- int NbCharBitLength

  Size in bits of the "number of characters of the entered user PIN" to overwrite in the APDU at nbCharBitOffset.
- int NbCharBitOffset

  Position in bits from the left side in the APDU data field to overwrite the number of characters.

6.44.1 Detailed Description

Representation of the PC/SC PINBLOCK structure as defined in PC/SC 2.0 part 9 "IFDs with Extended Capabilities".

6.44.2 Member Data Documentation

6.44.2.1 int Subsembly.SmartCard.PcSc.SCardPinBlock.NbCharMax

Maximum number of characters to be a valid PIN value.

6.44.2.2 int Subsembly.SmartCard.PcSc.SCardPinBlock.NbCharMin

Minimum number of characters to be a valid PIN value.

6.44.2.3 int Subsembly.SmartCard.PcSc.SCardPinBlock.PINBlockLength

Length of the PINBlock in bytes.

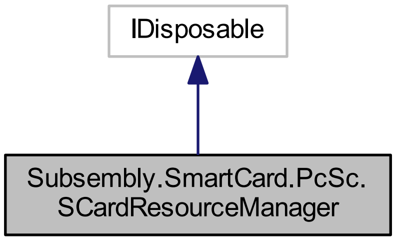
The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/PcSc/SCard↩ PinBlock.cs

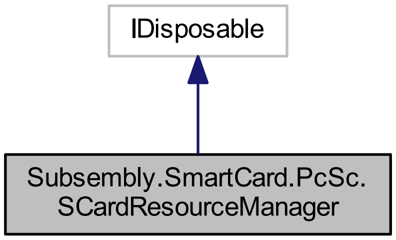
6.45 Subsembly.SmartCard.PcSc.SCardResourceManager Class Reference

The Resource Manager is a key component of the PC/SC Workgroups architecture. It is responsible for managing all PC/SC smart card resources.

Inheritance diagram for Subsembly.SmartCard.PcSc.SCardResourceManager:



Collaboration diagram for Subsembly.SmartCard.PcSc.SCardResourceManager:



Public Member Functions

- void Dispose ()

  Disposes this smart card resource manager context.
- bool EstablishContext (SCardContextScope nScope)

  Establishes the Smart Card Resource Manager Context to be used in subsequent communication with the Smart Card Resource Manager.
- void ReleaseContext ()

  Release a previously established context. This method differs from Dispose in that it verifies whether a context was actually established first and throws an exception if not.
- bool IsValidContext ()

  Indicates whether an established Context is still valid.
- int ListReaderGroups (IList aReaderGroupsList)

  Returns a list of the group names known to the system, that is, defined in the Smart Card Resource Manager database.
- int ListReaders (string[ ] vsGroups, IList aReadersList)

Returns a list of the readers assigned to one or more groups. Any invalid group names are ignored.

- int ListReaders (string sGroup, IList aReadersList)

  Returns a list of the readers assigned to the given group. If the given group name is invalid, then nothing will be returned.

- int ListReaders (IList aReadersList)

  Returns a list of all readers in the system (that is, the `SCard$AllReaders` group).

- bool GetStatusChange (int nTimeout, string sReaderName, SCardTrackStates nCurrentState, out SCardTrack↩ States nEventState)

  The method will block until there is a status change in the specified reader or the specified time-out period expires.

- bool GetStatusChange (int nTimeout, ref SCardTrackStateInfo aTrackState)

  The method will block until there is a status change in the specified reader or the specified time-out period expires.

- bool GetStatusChange (int nTimeout, SCardTrackStateInfo[ ] vaTrackStates)

  The method will block until there is a status change in any one of the specified readers or the specified time-out period expires.

- void Cancel ()

  The method provides a means to terminate outstanding GetStatusChange requests. Usually this is called from another thread.

Properties

- IntPtr Context `[get]`

  As a last resort, the native smart card resource manager handle is exposed here.

6.45.1 Detailed Description

The Resource Manager is a key component of the PC/SC Workgroups architecture. It is responsible for managing all PC/SC smart card resources.

Communication with the PC/SC Smart Card Resource Manager may occur only within a well- defined context. A S↩ CardResourceManager instance provides the methods necessary to create and manage this Context. This context is based on an unmanaged resource and therefore must be orderly disposed of after usage.

A principal responsibility of the Smart Card Resource Manager is maintenance of global information on the smart card service provider, cards, and readers known to the system. This responsibility is fulfilled by the native PC/SC Smart Card Resource Manager. The SCardResourceManager wraps the native Smart Card Resource Manager and provides methods for querying the database. Insert or delete information in the Smart Card Resource Manager database is currently not supported by this class. The ability of writing to the Smart Card Resource Manager database is usually only needed in order to install card terminals, which is outside the scope of this framework. Also, only the reader groups and the reader information is currently provided. The card and service provider information is not supported, yet.

Another service provided by the Smart Card Resource Manage is the tracking of card insertions and removals.

This class implements the combined functionality of the classes RESOURCEMANAGER, RESOURCEDB, RESOUR↩ CEQUERY, and SCARDTRACK of the PC/SC Workgroups architecture. See part 5 "ICC Resource Manager Definition" of the PC/SC Workgroup Specification. The PC/SC Workgroup Specification is available at the `PC/SC Workgroups homepage`.

See also

T:System.IDisposable

6.45.2 Member Function Documentation

6.45.2.1 void Subsembly.SmartCard.PcSc.SCardResourceManager.Cancel ( ) [inline]

The method provides a means to terminate outstanding GetStatusChange requests. Usually this is called from another thread.

The only thing that can be cancelled is an outstanding call to GetStatusChange. Any such outstanding action will be terminated with a status indication that the action was cancelled.

Exceptions

InvalidOperationException	Thrown when the requested operation is invalid.

6.45.2.2 void Subsembly.SmartCard.PcSc.SCardResourceManager.Dispose ( ) [inline]

Disposes this smart card resource manager context.

This method may be called regardless of a context was previously established or not. If no context was established, or the context was already released, then this method does nothing.

See also

M:System.IDisposable.Dispose()

6.45.2.3 bool Subsembly.SmartCard.PcSc.SCardResourceManager.EstablishContext (SCardContextScope *nScope*) [inline]

Establishes the Smart Card Resource Manager Context to be used in subsequent communication with the Smart Card Resource Manager.

To release an established Resource Manager Context, use ReleaseContext.

If the Smart Card Resource Manager Context was already successfully established when this method is called, then an InvalidOperationException will be thrown.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.

Parameters

nScope	The nScope parameter is intended to allow a caller to designate the security context that this Resource Manager Context operates within. Normally, this is associated with the user on whose behalf the calling process is running, and operations will be restricted to devices that the user is allowed to access. Alternately, one could request the security context appropriate for a specific terminal or the system as a whole. See SCardContextScope enumeration for possible scopes.

Returns

True if it succeeds, false if it fails.

6.45.2.4 bool Subsembly.SmartCard.PcSc.SCardResourceManager.GetStatusChange (int *nTimeout,* string *sReaderName,* SCardTrackStates *nCurrentState,* out SCardTrackStates *nEventState*) [inline]

The method will block until there is a status change in the specified reader or the specified time-out period expires.

Exceptions

CardTerminalException	Thrown when a Card Terminal error condition occurs.

Parameters

nTimeout	Maximum amount of time (in milliseconds) to wait for an action. A timeout value of -1 (= 0x←↩ FFFFFFFF INFINITE in Win32) is used to indicate that the calling application is willing to wait forever. A Timeout value of zero is used to indicate that the method should return immediately.
sReaderName	Name of reader to track. If an unknown reader is specified, then an exception is raised.
nCurrentState	The assumed last known state of the reader. Usually this is either the reader state returned through nEventState from a previous call to GetStatusChange, or SCardTrackStates.Unaware when called for the very first time.
nEventState	[out] The new reader tracking state at the time the method returns. The returned value should be passed as the nCurrentState in subsequent calls to this method.

Returns

If the state actually changed within the given timeout, then `true` is returned. If the state did not change, because the action timed out or was cancelled, then `false` is returned.

6.45.2.5 bool Subsembly.SmartCard.PcSc.SCardResourceManager.GetStatusChange (int *nTimeout,* ref SCardTrackStateInfo *aTrackState*) `[inline]`

The method will block until there is a status change in the specified reader or the specified time-out period expires.

Parameters

nTimeout	Maximum amount of time (in milliseconds) to wait for an action. A timeout value of -1 (= 0x←↩ FFFFFFFF INFINITE in Win32) is used to indicate that the calling application is willing to wait forever. A Timeout value of zero is used to indicate that the method should return immediately.
aTrackState	[in,out] SCardTrackStateInfo structure which supplies the reader name and the currently assumed state. On return this structure is updated with the new event state and the ATR of the inserted card, if any.

Returns

If the state actually changed within the given timeout, then `true` is returned. If the state did not change, because the action timed out or was cancelled, then `false` is returned.

6.45.2.6 bool Subsembly.SmartCard.PcSc.SCardResourceManager.GetStatusChange (int *nTimeout,* SCardTrackStateInfo[ ] *vaTrackStates*) `[inline]`

The method will block until there is a status change in any one of the specified readers or the specified time-out period expires.

Once this method returns, the application can determine which readers have undergone a state change, and entered a new state, by examining the SCardTrackStateInfo structures.

Note that this method will provide information on when cards are removed or inserted into the specified reader. It does not indicate anything about card types that may be in the reader. This behavior ensures that the calling application is made aware of all changes, making it possible to present appropriate UI.

WinCE: On Windows CE this method only supports a vaTrackStates parameter with a single array element. If an array with zero or more than one elements is passed, then a NotSupportedException exception will be thrown.

Exceptions

NotSupportedException	Thrown when the requested operation is not supported.
InvalidProgramException	Thrown when an Invalid Program error condition occurs.

Parameters

nTimeout	Maximum amount of time (in milliseconds) to wait for an action. A timeout value of -1 (= 0x↩ FFFFFFFF INFINITE in Win32) is used to indicate that the calling application is willing to wait forever. A Timeout value of zero is used to indicate that the method should return immediately.
vaTrackStates	Array of SCardTrackStateInfo structures which supplies the reader names and the currently assumed states of these readers. On return the structures are updated with the new event state and the ATR of the inserted card, if any.

Returns

If the state actually changed within the given timeout, then `true` is returned. If the state did not change, because the action timed out or was cancelled, then `false` is returned.

6.45.2.7 bool Subsembly.SmartCard.PcSc.SCardResourceManager.IsValidContext ( ) [inline]

Indicates whether an established Context is still valid.

Call this function to determine if a previously established Smart Card Resource Manager Context is still valid. Once a Resource Manager Context has been successfully established by EstablishContext, it may become invalid only, if the Smart Card Resource Manager service process has been shut down.

If the Resource Manager Context was not established at all, before this method is called, then an InvalidOperation↩ Exception will be thrown.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.

Returns

True if this smart card context is still valid. False if not.

6.45.2.8 int Subsembly.SmartCard.PcSc.SCardResourceManager.ListReaderGroups (IList *aReaderGroupsList*) [inline]

Returns a list of the group names known to the system, that is, defined in the Smart Card Resource Manager database.

This group contains all the readers in the system. A new reader given to the smart card subsystem is automatically included in the system-wide reader group, SCard$AllReaders. This group is implicit, therefore it is not returned by the ListReaderGroups method.

This default group, one for each terminal, contains all the readers assigned to the terminal that are not reserved for specific use.

This name is defined in the Win32 Smart Card API header files, but it obviously isn't used.

This name is defined in the Win32 Smart Card API header files, but it obviously isn't used.

<p>The Win32 Smart Card API defines some standard reader groups as follows: <list type="table"> <item> S↩ Card$AllReaders

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.

Parameters

aReader↩ GroupsList	A generic collection to which the names of all known reader groups will be added.

Returns

The number of known reader groups, and thus the number of strings added to aReaderGroupsList, is returned. If no reader groups are known, then zero is returned.

6.45.2.9 int Subsembly.SmartCard.PcSc.SCardResourceManager.ListReaders (string[ ] *vsGroups*, IList *aReadersList*) [inline]

Returns a list of the readers assigned to one or more groups. Any invalid group names are ignored.

A group is returned only if it contains at least one reader. This includes the group `SCard$DefaultReaders`. The group `SCard$AllReaders` cannot be returned, since it only exists implicitly.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.

Parameters

vsGroups	

String array with reader group names. If this is null, then all readers in the system (that is, the `SCard$AllReaders` group) will be listed.

The Windows CE implementation of the Smart Card API does not support the concept of reader groups. Hence, this parameter shall be null on Windows CE environments, or better, use the overloaded ListReaders(IList) method that takes no reader groups parameter.

Parameters

aReadersList	To this generic collection the names of the readers in the requested groups are added. If no matching reader is found, then nothing will be added. This collection will not be cleared, this method merely adds to it.

Returns

Returns the number of readers that were found and added to the aReadersList collection. If none was found, then zero is returned.

6.45.2.10 int Subsembly.SmartCard.PcSc.SCardResourceManager.ListReaders (string *sGroup*, IList *aReadersList*) [inline]

Returns a list of the readers assigned to the given group. If the given group name is invalid, then nothing will be returned.

Exceptions

ArgumentNullException	Thrown when one or more required arguments are null.

Parameters

sGroup	

String that supplies the desired reader group name. This must neither be null nor a zero length string.

The Windows CE implementation of the Smart Card API does not support the concept of reader groups. Hence, this method cannot be called. Use the overloaded ListReaders(IList) method that takes no reader groups parameter instead.

Parameters

aReadersList	To this genric collection the names of the readers in the requested groups will be added. If no matching reader is found, then nothing will be added. This collection will not be cleared, this method merely adds to it.

Returns

Returns the number of readers that were found and added to the aReadersList collection.

6.45.2.11 int Subsembly.SmartCard.PcSc.SCardResourceManager.ListReaders (IList *aReadersList*) `[inline]`

Returns a list of all readers in the system (that is, the `SCard$AllReaders` group).

Parameters

aReadersList	To this generic collection the names of the readers in the requested groups are added. If no reader is found, then nothing will be added. This collection will not be cleared, this method merely adds to it.

Returns

Returns the number of readers that were found and added to the aReadersList collection.

6.45.2.12 void Subsembly.SmartCard.PcSc.SCardResourceManager.ReleaseContext ( ) `[inline]`

Release a previously established context. This method differs from Dispose in that it verifies whether a context was actually established first and throws an exception if not.

If the Resource Manager Context was not established at all, before this method is called, then an InvalidOperation↩ Exception will be thrown.

Exceptions

ObjectDisposedException	Thrown when a supplied object has been disposed.
InvalidOperationException	Thrown when the requested operation is invalid.

6.45.3 Property Documentation

6.45.3.1 IntPtr Subsembly.SmartCard.PcSc.SCardResourceManager.Context `[get]`

As a last resort, the native smart card resource manager handle is exposed here.

It is up to the caller to ensure that the handle is actually valid before using it.

The context.

The documentation for this class was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/PcSc/SCard↩ ResourceManager.cs

6.46 Subsembly.SmartCard.PcSc.SCardTrackStateInfo Struct Reference

Structure that holds information for the purpose of tracking the card reader state. This kind of structure is used by the SCardResourceManager.GetStatusChange method.

Public Attributes

- string sReaderName

  The name of the card reader to track.
- SCardTrackStates nCurrentState

  The assumed last known state of the reader. Usually this is either the reader event state returned from a previous call to SCardResourceManager.GetStatusChange, or SCardTrackStates.Unaware.
- SCardTrackStates nEventState

  The current/new reader tracking state at the time the method returns.
- byte[ ] vbATR

  If a card is inserted and activated, then this will receive the ATR of the inserted card.

6.46.1 Detailed Description

Structure that holds information for the purpose of tracking the card reader state. This kind of structure is used by the SCardResourceManager.GetStatusChange method.

6.46.2 Member Data Documentation

6.46.2.1 SCardTrackStates Subsembly.SmartCard.PcSc.SCardTrackStateInfo.nEventState

The current/new reader tracking state at the time the method returns.

6.46.2.2 string Subsembly.SmartCard.PcSc.SCardTrackStateInfo.sReaderName

The name of the card reader to track.

The documentation for this struct was generated from the following file:

- C:/dev/smartcard-api/smartcard-api/commercial/SmartCardAPI.Professional/SmartCard.Library/PcSc/SCard↩ ResourceManager.cs

Index