SmartCardAPI Developer Guide

CardWerk Technologies
2615 George Busbee Pkwy, Suite 11-312
Kennesaw, GA 30144
USA

Email: support@smartcard-api.com
Online: https://smartcard-api.com

CardWerk SmartCard API (Professional)

Developer Guide rev. 18JAN2021

Overview 2
- Product Offerings 3
- Summary of Advantages 3
- List of Features 3
- CardWerk SmartCard API (Professional) Development Kit 4

SmartCard API Framework Architecture 6
- Namespace overview 6

High-level API 8
- Working with the CardHandle 8
- The Framework 10
- Smart Card UI 11
- Card Reader Configuration 12
- Utility Classes 14

PC/SC Workgroup level API 15
CT-API level API 16
Deployment 17
Card Terminal Configuration 18

Overview

Smart cards are becoming increasingly popular as a means for convient personal security and authentication in many application areas such as e-Banking and e-Commerce. Many Windows applications have been written supporting smart cards for various purposes. Millions of users have a smart card in their pocket without even knowing – as SIM card it has become part of every GSM phone and EMV-based transactions.

With the .NET technology, a framework for future application development was introduced by Microsoft. A vast number of Windows applications are based on the .NET Framework. The .NET Framework became even more important for application developers with the release of the .NET Compact Framework.

As of today, however, the .NET Framework does not include smart card support. Therefore application developers using .NET had to resort to creating their own P/Invoke wrappers to access the Windows native PC/SC Workgroup API from within the .NET environment. If done right, a difficult and time consuming task, time that would better be spent on developing the application itself.

CardWerk SmartCard API (Professional) is the first .NET class library and framework for easy access to smart card readers and smart cards from .NET applications. Using the CardWerk SmartCard API (Professional) an application developer no longer needs to deal with P/Invoke stubs, but rather uses a convenient API that provides an even higher level of abstraction than the Windows native PC/SC Workgroup API.

CardWerk SmartCard API (Professional) is not just a wrapper for the PC/SC Workgroup API but provides a framework that can be extended to support any smart card reader technology. Indeed, CardWerk SmartCard API (Professional) comes with support for CT-API based smart card readers. The CT-API is a German standard for smart card reader access that includes advanced features such as multiple slots, PIN pad, display and secure PIN entry.

Application developers familiar with the Java programming environment may recognise the similar concept of the OpenCard Framwork for the Java platform. From a very simple point of view, CardWerk SmartCard API (Professional) could be said to provide the same services to the .NET environment as the OpenCard Framwork to the Java environment. The design of CardWerk SmartCard API (Professional) was influenced by the Java OpenCard Framwork and application developers familiar with OpenCard will quickly become acquainted with the CardWerk SmartCard API (Professional). However, the CardWerk SmartCard API (Professional) is not just a bland copy of the OpenCard Framework, but a major improvement – easier to use yet with greatly advanced features and full .NET utilization.

We took this idea even a step further by introducing CardModule assemblies to allow integration of some popular smart cards with a few lines of code and without knowing anything about the actual card edge interface on APDU level.

Product Offerings

CardWerk SmartCard API (Professional) is available as a software development kit (SDK). It includes extensive documentation and many sample applications with source code.

Buying the CardWerk SmartCard API (Professional) development kit entitles you to redistribute the binary CardWerk SmartCard API (Professional) assembly to your customers without paying additional royalties.

Maintenance contracts, site licenses, and source code licences are available. We also offer on site training and custom development and consulting services.

Summary of Advantages

- Reduces development time by eliminating the need to create your own P/Invoke wrapper and by providing a higher level and more convenient API.
- Makes your applications independent of the underlaying smart card system. The application can be used with PC/SC Workgroup API smart card readers and CT-API smart card readers without any change.
- Continuously maintained product that is extended to cope with future needs.
- Contactless card classes for DESFire, iCLASS, Mifare can be written on top of SmartCard API. We even provide some sample classes to get you up and coding in no time.
- Proprietary reader systems can be supported via custom adapter modules.

List of Features

The CardWerk SmartCard API (Professional) has many advanced features, such as support for secure PIN entry, not found in similar products. The following list is an overview of the more important features:

Completely implemented in .NET based on the common language system (CLS) subset.

Supports C#, VisualBasic.NET (VB.NET), J#, Delphi for .NET and every other CLS compliant .NET programming language.

Supports all PC/SC Workgroup API compatible smart card readers.

Supports T=0 and T=1 asynchronous smart cards.

Supports contactless cards (via PC/SC 2.01)

Supports synchronous memory cards with CT-API card readers and some PC/SC Workgroup compliant card readers and proprietary APIs such as MCARD API, a storage card API for Identiv (formerly SCM) card readers.

OMNIKEY's Scardsyn API for storage cards via CardModule.MemoryCard.dll assembly.

HID iCLASS card support available upon request.

Supports transmission of APDUs over TPDU based APIs such as the PC/SC Workgroup API.

Supports all CT-API compatible smart card readers.

Supports parallel usage of multiple smart card readers.

Framework can be extended to support proprietary smart card reader APIs.

Includes card reader configuration tool

Supports secure PIN entry for PIN verification and PIN modification with PC/SC and CT-API smart card terminals that include a PIN pad.

Supports complete GUI-less operation, but also provides convient customizable GUI-based user dialogs for PIN verification and PIN modification.

GUI elements localized in English and German. Additional languages can be added in future releases upon request.

Supports asynchronous, event driven applications and synchronous applications.

Provides a convenient high level API, a low level PC/SC Workgroup API compliant API and a low level CT-API based API.

Software development kit includes various sample applications with documented source code.

Binaries based on NET Framework 3.5, 4.5 and 4.6.1

CardWerk SmartCard API (Professional) Development Kit

The CardWerk SmartCard API (Professional) is delivered with a development kit that contains extensive documentation and sample applications including source code.

- Binary redistributable assemblies.

- Technical documentation in PDF format.

- Extensive online reference documentation in CHM help format.

- Fully commented C# sample applications including source code.

The following C# sample applications are included:

- "HelloCard“ - HelloWorld-type sample program to demonstrate card detection. If the card responds to a PC/SC 2.01 ReadUID() APDU, the UID is displayed. Otherwise a contact card is assumed. In either case the ATR is displayed.

- "HelloICLASS" - Allows access to HID-issued PACS bit data. Extracts facility code and card number for some standard card formats.

- "HelloMemoryCard" - Sample program to access storage cards such as SLE 4442, SLE 4428, I2C and cards via managed wrapper code for OMNIKEY scardsyn.dll propprietary, unmanaged API

- "HelloMifare“ - HelloWorld-type sample program to demonstrate access of contactless storage cards via CardWerk's PC/SC 2.01 part 3 CardModule.CLIC.

- "HelloPIV“ – Sample for CAC, NIST PIV and TWIC cards used by U.S. Government agencies and contractors. Shows how to read a CHUID. Extracts Expiration date, GUID and FASCN from raw CHUID data. Reads X509 certificate and card capability containers

- "HelloPROX“ – Sample for HID PROX card access. Includes code to extract card number (CN) and facility code (FAC) applying some of the known standard HID card formats.

- "HelloGeldkarte" – Reads the current available amount of a German GeldKarte electronic purse. (prev releases: Taschenkartenleser)

- "HelloSim" – Reads the phone book entries of a GSM SIM card. This sample also demonstrates PIN verification and modification. (prev releases: SimTrivia)

- "HelloMct" – Reads some data from a German health insurance smart card (KSK and eGK supported). This sample demonstrates working with synchronous memory cards.(prev releases: MctTrivia)

SmartCard API Framework Architecture

Namespace overview

The classes of CardWerk's SmartCard API Framework are grouped into five distinct namespaces. The classes of all namespaces are bundled in the single .NET assembly "SmartCard.dll".

Namespace	*Abstraction*
Subsembly.SmartCard	High level access to smart card reader and smart cards, completely independent of actual smart card reader API or driver technology.
Subsembly.SmartCard.PcSc	Low level PC/SC Workgroup API compliant smart card reader access. Provides full PC/SC Workgroup API functionality.
Subsembly.SmartCard.CtApi	(depreceated) Low level CT-API based smart card reader access. Provides full CT-API and CT-BCS functionality.
Subsembly.SmartCard.MCard	(depreceated) Low level MCARD API based memory card access and SmartCard API Framework extension. Depends on Subsembly.SmartCard.PcSc and Subsembly.SmartCard.
SmartCardAPI.CardModule.xyz - assemblies to support a variety of card edges CardModules are only available for SmartCardAPI(Professional).	
.CAC	U.S. DOD Common Access Card (CAC) card module including helper functions to access CAC card. Requires Subsembly.SmartCard.
.CLICS	Contactless integrated circuit modul according to PC/SC 2.01 part 3. Requires Subsembly.SmartCard.
.DESFire	NXP DESFire EV1 with host-side crypto module.
.HID.ICLASS	HID iCLASS and SEOS raw Wiegand data access via OMNIKEY desktop readers. Supports PACS bit access on OMNIKEY readers. **Upon request.**
.HID.PROX	Provides access to HID Prox card raw Wiegand data on OMNIKEY card readers.

Namespace	*Abstraction*
.MemoryCard	Supports storage cards (i.e synchronous cards) on OMNIKEY contact card readers.
.MifareClassic	NXP Mifare Classic card access via PC/SC 2.01 part 3 compliant card readers. Requires CardModule.CLICS.
.NfcTag	NXP Mifare Ultralight and NTAG support
.PIV	U.S. Gov. NIST PIV Card API including helper functions to access CHUID, FASCN, GUID and Expiration Date. Depends on Subsembly.SmartCard. **Not available for Express version.**
SmartCardAPI.DataModule.xyz - assemblies to support commonly used data blobs DataModules are only available for SmartCardAPI(Professional).	
.Wiegand	Convenience class for raw Wiegand Data. Has built-in methods tyo extract data items fos known card fornats such as HID H10301 26 bit format and many more. Can be used with iCLASS and Prox CardModule.
SmartCardAPI.ReaderModule.xyz - assemblies to support proprietary reader APIs ReaderModules are only available for SmartCardAPI(Professional).	
.Omnikey.Scardsyn	.NET wrapper for OMNIKEY scardsyn.dll and scardsynx64.dll.
.Omnikey.SecureChannel	To establish secure channel between host application and OMNIKEY reader. Contains HID proprietary crypto protocol.
SmartCardAPI.SecurityModule.xyz - assemblies for cryptographic operations SecurityModules are only available for SmartCardAPI(Professional).	

The namespace Subsembly.SmartCard defines an interface ICardTerminal that can be implemented in order to support any smart card reader API or driver. CardWerk SmartCard API (Professional) includes three implementations of this interface. One for PC/SC Workgroup API compliant smart card readers, one CT-API based implementation, and one MCARD API based memory card implementation.

As a general rule, applications should be solely based on the high level API provided by the namespace Subsembly.SmartCard whenever possible. This guarantees full independence of the underlying native API and therefore support for future smart card APIs. Low level, often unmanaged libraries may still provide card reader specific functionality. However the application can be coded on top of a much higher, easier to use abstraction layer.

High-level API

The High-level API is located in the Subsembly.SmartCard namespace which represents the heart of the CardWerk SmartCard API (Professional) product.

Working with the CardHandle

From the application point of view the high level API consists of the two highly abstracted primary classes CardTerminalManager and CardHandle. The CardTerminalManager is a singleton class that acts as the central hub of the SmartCard API Framework. The CardTerminalManager controls all registered card terminals and monitors card terminal slots for card insertion or removal, raising events as necessary. Once a card is inserted, a CardHandle instance can be acquired for it from the CardTerminalManager. Acquiring such instance automatically powers up the card at which time an answer to reset (ATR) should be available. As the CardHandle class handles any access to a smart card the host application must keep a reference to the CardHandle instance throughout the whole card session.

Among others, the following methods are provided by the CardHandle class. These methods allow ISO7816-4 compliant card communication without having to go into details of the underlaying APDUs. Please refer to the CardWerk SmartCard API (Professional) reference for a detailed description of all methods of the CardHandle class.

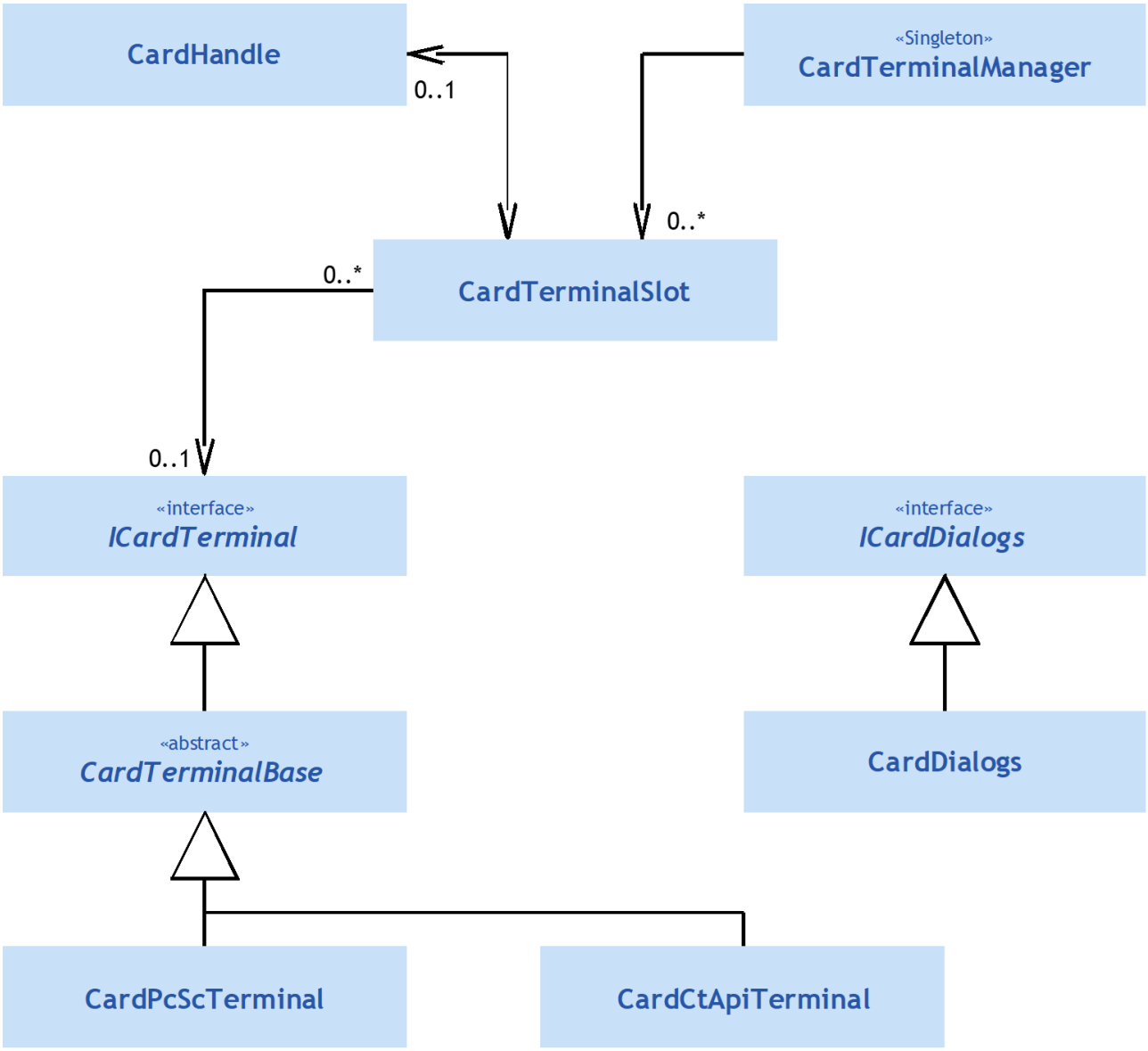
Method / Property	*Description*
SendCommand	Sends any command APDU to the card and returns the response APDU.
SelectApplication	Selects a card application by its application ID.
SelectFile	Selects an elementary file (EF) on the card.
ReadBinary	Reads binary data from the currently selected elementary file (EF) with transparent structure.
UpdateBinary	Updates binary data in the currently selected elementary file (EF) with transparent structure.
ReadRecord	Reads a record from the currently selected elementary file (EF) with record structure.
UpdateRecord	Updates a record in the currently selected elementary file (EF) with record structure.
VerifyPin	Performs a PIN verification either using secure PIN entry of the

Method / Property	*Description*
	card terminal, if supported, or using an ordinary dialog box.
ChangePin	Performs verifying and changing of a PIN either using some secure means provided by the card terminal or using an ordinary dialog box.

If the card edge supports more than these basic functions, you can extend the card handle class to adapt your host application to your card.

The Framework

The following figure provides a more detailed overview of the SmartCard API Framework constructed by the classes of the Subsembly.SmartCard namespace. In addition to the classes shown in this figure there are several utility classes explained further below.



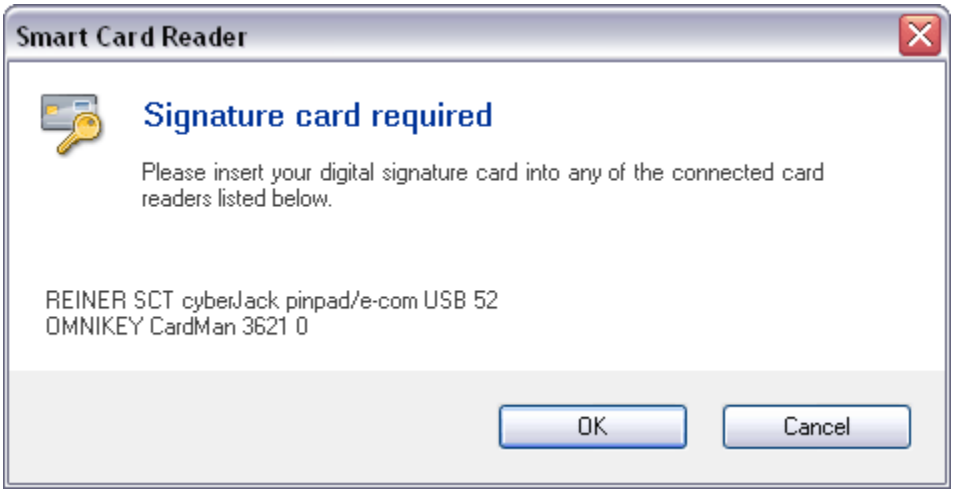
For each card reader slot there is an instance of the class CardTerminalSlot that connects the CardTerminalManager and the CardHandle to the ICardTerminal interface.

Additional types of card readers can be integrated into the framework by implementing the ICardTerminal interface. The CardTerminalBase provides a boilerplate implementation of this interface that can be used as a base class for your particular card reader integration. The CardWerk SmartCard API (Professional) already includes three implementations of the ICardTerminal interface based on the CardTerminalBase class. The CardPcScTerminal class implements the ICardTerminal interface based on the native PC/SC Workgroup API, the CardCtApiTerminal class implements the interface based on a native 32 Bit CT-API Windows DLL.

Smart Card UI (depreceated)

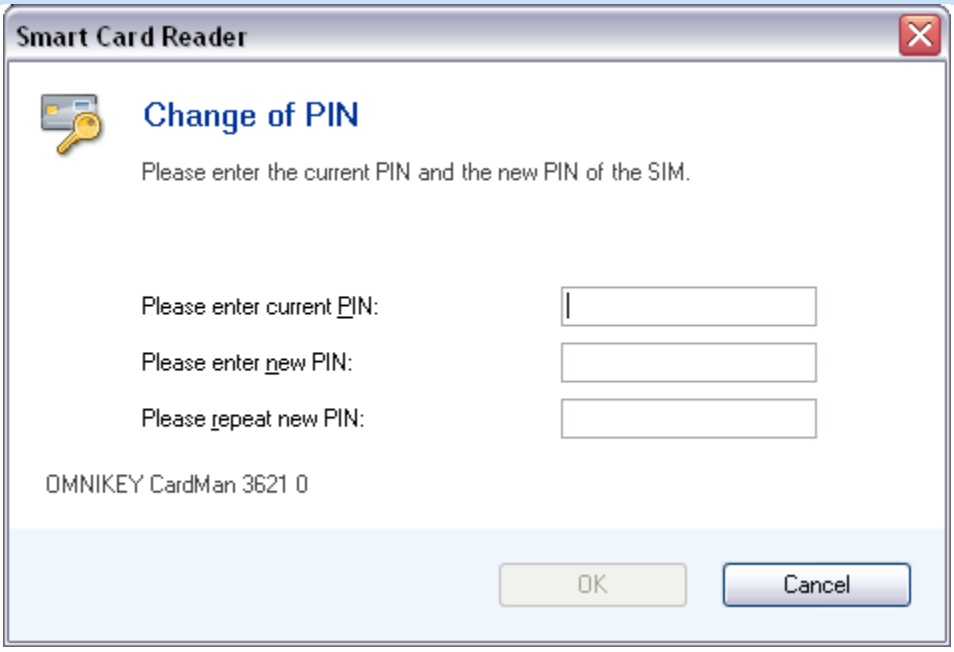
The high level Subsembly.SmartCard namespace includes optional GUI elements that can be completely customized or completely replaced. The GUI elements are abstracted away behind the ICardDialogs interface. An application that wants to replace the standard GUI provided by the CardDialogs class simply implements this interface and pass it to the methods that need it. Alternatively the default CardDialogs class can be instantiated and customized through various parameters, or the application can restrict itself to methods that do not show any GUI elements.

The following picture shows the default insert card prompt dialog box implemented by the CardDialogs class. The actual prompt text is provided by the calling application. By clicking on the Setup button the user can make some last minute changes to the card reader configuration. At the bottom the names of all card readers available for card insertion are listed.



The following picture shows the default verify PIN dialog box implemented by the CardDialogs class. Again, the prompt text is provided by the calling application. At the bottom the name of the card reader is shown.

Finally, the following picture shows the default change PIN dialog box implemented by the CardDialogs class. Again the prompt text is provided by the calling application and the name of the card reader is shown.



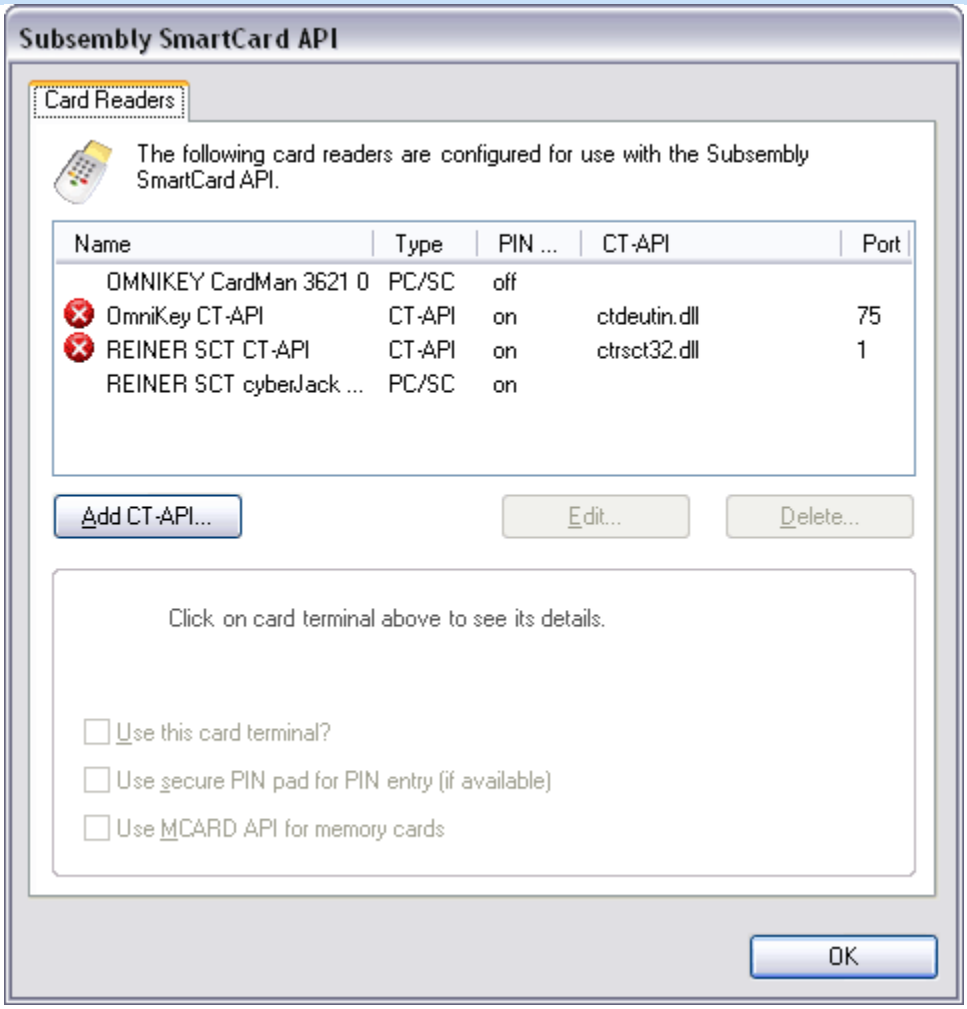
Again, the application can provide its own dialog boxes and is not required to use the standard dialog boxes provided by the CardWerk SmartCard API (Professional).

Card Reader Configuration

For CT-API support a smart card application needs configuration information such as the name of the CT-API DLL of the card reader and a port number. This and other configuration information is managed by the CardTerminalRegistry class. The CardTerminalRegistry uses an XML file to persistently store this configuration information.

The CardTerminalRegistry also implements a card reader configuration tool that can be launched through a simple call to the ShowConfigurator method of the CardTerminalRegistry class or through the CardTerminalConfigurator.exe that is included with the CardWerk SmartCard API (Professional).

The following screenshot shows the card terminal configurator with four configured card terminals.



Among other information, a configuration entry in the card terminal configuration file contains the following XML elements.

XML Tag	*applies to*	*Purpose*
AssignedName	all	Unique identification of a configured card terminal.
AssemblyName	all	Name of the assembly that contains the actual ICardTerminal implementation that plugs into the framework.
ClassName	all	Fully qualified name of class that contains the actual ICardTerminal implementation that plugs into the framework
Enabled	all	Indicates whether this card reader shall be used or not.
UseSecurePin	all	Flag that indicates whether secure PIN entry shall be used, or not.
ReaderName	PC/SC	Name of card reader as it is known by the PC/SC Workgroup API.

XML Tag	applies to	Purpose
CtApiDll	CT-API	Filename of the DLL that implements the CT-API specification for the configured CT-API card reader.
CtApiPort	CT-API	Port number of port that the card reader connects to.

Utility Classes

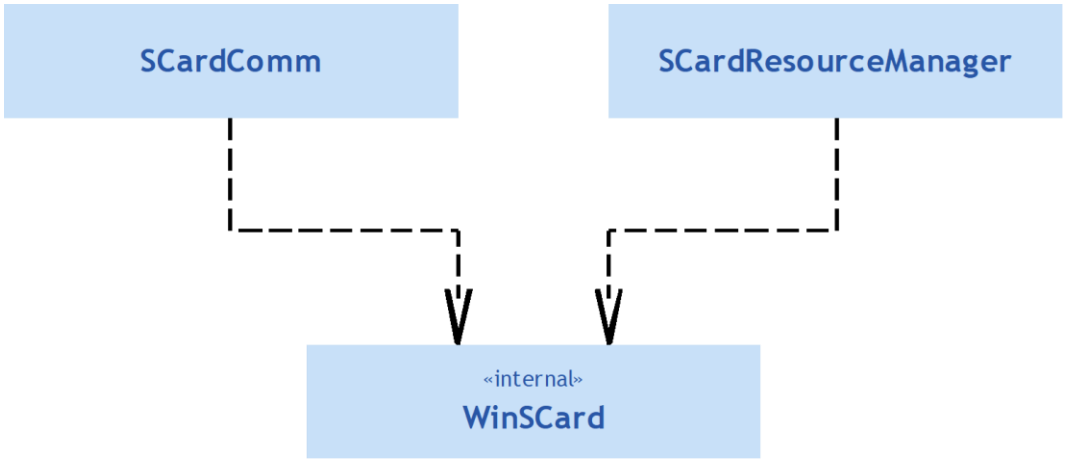
The Subsembly.SmartCard namespace provides several utility classes that help procesing smart card commands and smart card data. The following table provides an overview of the utility classes included in the Subsembly.SmartCard namespace.

Class	Purpose
CardCommandAPDU	To create, parse and modify ISO 7816-4 compliant command APDUs with short or extended length fields.
CardResponseAPDU	To create, parse and modify ISO 7816-4 compliant response APDUs.
CardDataObject	To create, parse and modify BER-TLV or SIMPLE-TLV encoded data objects or sequences of such data objects.
CardHex	Utility functions for converting hexadecimal strings to and from byte arrays or numeric values.

PC/SC Workgroup level API

The PC/SC Workgroup level API is located in the Subsembly.SmartCard.PcSc namespace. The PC/SC Workgroup level API conveniently exposes native Windows API residing in winscard.dll to a .NET environment. The architecture as well as the naming of classes and symbols is kept similar to the PC/SC Workgroup specification to emphasise the close relationship between managed and unmanaged code.

The following figure provides an overview of the primary classes of the Subsembly.SmartCard.PcSc namespace.



The internal class WinSCard provides the P/Invoke stubs to the native API implemented in Microsoft Windows' native winscard.dll. Note that the WinSCard class is not available to applications.

CT-API level API

The CT-API level API is located in the Subsembly.SmartCard.CtApi namespace. The CT-API is a very simple procedural API. In order to resemble this as closely as possible, the CT-API level API consists of a single primary class CtApiWrapper. All CT-API functionality is provided through this class.

The CtApiWrapper includes convenience methods for the complete range of CT-BCS defined operations and the B1 defined operations.

Instances of the CtApiWrapper class can be created for any local CT-API DLL with arbitrarily named function names. Actually, the name and path the of CT-API DLL and optionally the names of the three CT-API functions CT_init, CT_data, and CT_close are provided to the constructor of the CtApiWrapper class. The CtApiWrapper constructor dynamically creates an assembly with appropriate P/Invoke stubs at runtime using Reflection Emit technology.

Deployment

CardWerk SmartCard API (Professional) consists of the single assembly SmartCard.dll. In addition, there can be optional satellite resource DLLs that contain the localised texts for the user interface. In order to deploy the CardWerk SmartCard API (Professional) all files just have to be copied into your main application directory.

Besides the English resources built into SmartCard.dll itself, the current distribution includes a German resources satellite assembly. Further resource languages can be added in future releases.

The deployment of the complete CardWerk SmartCard API (Professional) consists of the following files:

```
SmartCard.dll
de/SmartCard.resources.dll
```

The German resources satellite assembly is located in the file SmartCard.resources.dll and must be copied into a subdirectory named "de". The directory name must reflect the target language, in order to be picked up by the main assembly. This guarantees that the .NET Framework will automatically use German resources on a German Windows environment.

Please contact us if you need support for an additional language. We can send you the English text file for translation and will then include the additional resource in the subsequent release.

Card Terminal Configuration

The CardWerk SmartCard API (Professional) card terminal configuration is stored in a local XML file with the filename "Registry.xml". The file is located in the Subdirectory Subsembly/SmartCard which itself is located in the common application data folder. The location of the common application data folder is determined through the following .NET method call:

```
Environment.GetFolderPath(Environment.SpecialFolder.CommonApplicationData);
```

The actual location depends on the Windows version and Windows localisation. On a German Windows XP system the file "Registry.xml" is found in:

```
\Dokumente und Einstellungen\All Users\Anwendungsdaten\Subsembly\SmartCard
```

On an American Windows XP system "Registry.xml" resides in:

```
\Documents and Settings\All Users\Application Data\Subsembly\SmartCard
```

The registry file is solely maintained by the CardTerminalRegistry class and must never be accessed directly from code. This is mainly because future versions of the CardWerk SmartCard API (Professional) may be subject to changes on format level.

In order to plug in a custom implementation of the ICardTerminal interface for a proprietary smart card reader not supported by CardWerk SmartCard API (Professional), the configuration file must be manually edited. Assuming a specialized ICardTerminal implementation for the smart card reader „XYZreader“ has been created as a class with the fully qualified name SmartCard.XYZreader.YYZreaderCardTerminal in an assembly named SmartCard.XYZreader.dll, the following entry must be added to the configuration file below the <CardTerminalRegistry> root node:

```
<CardTerminal>
   <AssignedName>My XYZreader Reader</AssignedName>
   <AssemblyName>SmartCard.XYZreader.dll</AssemblyName>
   <ClassName>SmartCard.XYZreader.XYZreaderCardTerminal</ClassName>
   <Enabled>true</Enabled>
   <UseSecurePin>false</UseSecurePin>
   <Config>
      <ReaderPort>4</ReaderPort>
      <ProprietaryParameter>BEEP</ProprietaryParameter>
   </Config>
</CardTerminal>
```

The element <Config> contains proprietary configuration elements that are only understood by the XYZreaderCardTerminal class and are ignored by the CardWerk SmartCard API (Professional) itself. In this example an element <ReaderPort> is used to configure the port where the smart card reader is actually connected. The CardWerk SmartCard API (Professional) passes this information to the XYZreaderCardTerminal class through the method ICardTerminal.Init. In this case the <ReaderPort> element may match a constant defined in XYZreader.h to indicate a reader interface such a COM port or USB.

In order to have the CardWerk SmartCard API (Professional) successfully load the indicated class, one must deploy the SmartCard.XYZreader.dll assembly together with SmartCard.dll.

In any case, it is useful to understand the format of the registry file for diagnostic reasons. The format of the registry file is defined in the fully commented XML Schema file CardTerminalRegistry.xsd that is included in the CardWerk SmartCard API (Professional) development kit and shown here.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <!-- TODO
        Strings should have further constraints!
        Can we define default values?
   -->

   <!-- The CardTerminalRegistry is just a sequence of CardTerminalConfig elements.
        It may contain zero or any number of entries. The CardTerminalRegistry is the root
        element of the persistent card terminal registration XML file.
   -->
   <xsd:element name="CardTerminalRegistry">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="CardTerminalConfig" maxOccurs="unbounded" />
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

   <!-- The CardTerminalConfig element provides all information about a card
        terminal that has been configured for CardWerk SmartCard API for .NET.
   -->
   <xsd:element name="CardTerminalConfig">
      <xsd:complexType>
         <xsd:sequence>

            <!-- Name that uniquely identifies a card terminal to the user. This
                 name may be arbitrarily assigned by the user, as long as it is unique
                 within the entire CardTerminalRegistry.
            -->
            <xsd:element name="AssignedName" type="xsd:string" />

            <!-- Name of the .NET assembly that contains the ICardTerminal
                 implementation for this card terminal. If this is missing, then the
                 class is sought in the currently executing assembly.
            -->
            <xsd:element name="AssemblyName" type="xsd:string" minOccurs="0" />

            <!-- Name of the .NET class that contains the ICardTerminal
                 implementation for this card terminal.
            -->
            <xsd:element name="ClassName" />

            <!-- Flag that indicates whether this card terminal shall be used or
                 not. If absent, then the card terminal is enabled. Only if this is
                 explicitly set as false, then this card terminal configuration will be
                 ignored.
            -->
            <xsd:element name="Enabled" type="xsd:boolean" />

            <!-- Optional flag that indicates whether the secure PIN entry of a
                 class 2 card terminal shall be used. If this is absent, then the secure
                 PIN entry will be used when supported by the card terminal, i.e. the flag
                 will be assumed as true. If this flag is "true" and the card terminal does
                 not support secure PIN entry, then nothing will happen, we will just resort
                 to unsecure PIN entry.
            -->
            <xsd:element name="UseSecurePin" type="xsd:boolean" minOccurs="0" />

            <!-- This element contains additional configuration options that are
                 particular to the card terminal class.
            -->
            <xsd:element ref="Config" minOccurs="0" />

         </xsd:sequence>
      </xsd:complexType>
```



```
</xsd:element>

<!-- The Config element is just a container for any driver specific
     configuration elements
-->
<xsd:element name="Config">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:any maxOccurs="unbounded" processContents="lax" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<!--
     Configuration elements of CT-API entry
-->

<!-- The CtApiDll element provides the filename of the shared CT-API library,
     optionally including its complete path. If only the filename without the path is
     given, then the CT-API library must be located somewhere in the default shared
     library search path of the system (e.g. on a Windows system in the Windows or
     System directories).
-->
<xsd:element name="CtApiDll" type="xsd:string" />
<!-- The CtApiPort element provides a port number to be passed to the CT_init
     function of the CT-API.
-->
<xsd:element name="CtApiPort" type="xsd:unsignedShort" />
<!-- The CtInitProcName provides an alternative procedure name for the CT_init
     function of the CT-API (see Annex A of the CT-API specification. If this element is
     omitted, then it defaults to CT_init.
-->
<xsd:element name="CtInitProcName" type="xsd:string" />
<!-- The CtDataProcName provides an alternative procedure name for the CT_data
     function of the CT-API (see Annex A of the CT-API specification. If this element is
     omitted, then it defaults to CT_data.
-->
<xsd:element name="CtDataProcName" type="xsd:string" />
<!-- The CtCloseProcName provides an alternative procedure name for the CT_close
     function of the CT-API (see Annex A of the CT-API specification. If this element is
     omitted, then it defaults to CT_close.
-->
<xsd:element name="CtCloseProcName" type="xsd:string" />

<!-- This just shows what is expected in the Config element by the CT-API
     driver...
-->
<xsd:element name="CtApiConfig">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="CtApiDll" />
         <xsd:element ref="CtApiPort" />
         <xsd:element ref="CtInitProcName" minOccurs="0" />
         <xsd:element ref="CtDataProcName" minOccurs="0" />
         <xsd:element ref="CtCloseProcName" minOccurs="0" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<!--
     Configuration elements of PC/SC Workgroup entry
-->

<xsd:element name="PcScReaderName" type="xsd:string" />

<!-- This just shows what is expected in the Config element by the PC/SC
     driver...
-->
<xsd:element name="PcScConfig">
   <xsd:complexType>
      <xsd:sequence>
```



```
         <xsd:element ref="PcScReaderName" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<!--
      CT-API Product specification

      This is not used yet, but may be used to define the configuration options of a CT-API
      smart card reader in a future release.
-->

<xsd:element name="CtApiSpec">
   <xsd:complexType>
      <xsd:sequence>
         <!-- The CtApiDll must provide the filename of the CT-API DLL excluding
              the path, e.g. "CT32.DLL".
         -->
         <xsd:element ref="CtApiDll" />
         <!-- The CtApiPort element may be used to provide a list of applicable
              port numbers from which the user shall choose. If only a single value is
              given, then it is implicitly used by the application without giving the
              user a choice. If this element is not provided, then the user may choose
              any port number.
         -->
         <xsd:element ref="CtApiPort" minOccurs="0" maxOccurs="unbounded" />
         <!-- If the CT-API DLL uses other procedure names than CT_init, CT_data,
              and CT_close, then they may be provided by the following elements. For every
              missing element, the default procedure name is assumed.
         -->
         <xsd:element ref="CtInitProcName" minOccurs="0" />
         <xsd:element ref="CtDataProcName" minOccurs="0" />
         <xsd:element ref="CtCloseProcName" minOccurs="0" />
         <!-- In order to be able to uniquely identify a particular CT-API card
              terminal its manufacturer data object value must be provided. If this
              element is given, it must provide at least the first five bytes (CT
              manufacturer). The idea is, that the CtApiDll in combination with the
              CtManufacturer provide a unique identification of the card terminal to the
              application.
         -->
         <xsd:element ref="CtManufacturer" minOccurs="0" />
         <!-- If this driver/device has parallel support for CT-API and PC/SC then
              this element shall be used to provide the associated PC/SC reader name
              without its number suffix, e.g. "Cherry SmartBoard XX44". This is useful to
              avoid parallel access through both APIs and to enable simple switching from
              CT-API to PC/SC and vice versa.
         -->
         <xsd:element ref="PcScReaderName" minOccurs="0" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
</xsd:schema>
```

