

# .NET Smart Card API

Data: 10/12/2007

Author: Ugo Chirico – <http://www.ugosweb.com>

Resource Manager API for accessing smart cards is contained in the dynamic link library (dll) `winscard.dll` that is implemented in C and thus it is unmanaged code. The .NET Framework 2.0 and also the recent version 3.0, doesn't supply a managed package which maps Resource Manager API with a specific set of managed classes. Therefore to use Resource Manager API directly in your C# and/or VB.NET code there is only one chance: using platform invoke services, such as the `System.Runtime.InteropServices` namespace and the CLR supplied by the framework to import the unmanaged functions exported by `winscard.dll` in your C# and/or VB.NET managed code. Platform invoke enables managed code to call functions exported from an unmanaged dll such as Win32 API, `winscard` API and custom dlls while the CLR handles dll loading and parameter marshaling.

However, importing functions from an unmanaged dll, especially from `winscard.dll`, requires some advanced skills in C/C++ and .NET and compels a lot of tedious work in declaring the prototypes related to the functions to import and in dealing with custom parameters marshalling.

In order to avoid from dealing with unmanaged code and avoiding also from many tedious work, we have developed a C++ managed extension package of classes which maps Resource Manager API by an high-level set of classes usable from C# and VB.NET. Next figure shows the class hierarchy of .NET Smart Card API, aka, NSCAPI.



The class hierarchy of .NET Smart Card API

The main class is SmartCardManager which supplies functions to manage readers and reader's events. The property PluggedReaders gives the list of readers plugged to the PC while the property ActiveReaders gives the list of readers having a smart card plugged-in (active).

The class Readers is a list of Reader object. The method FindByName allows searching a specific reader giving its name.

The class Reader supplies methods to manage a reader and to connect to the inserted smart card. The method connect allows connecting to the inserted smart card and returns a SmartCard object. The connection can be: shared (MODE\_SHARED) to share smart cards with other applications; exclusive (MODE\_EXCLUSIVE) to avoid from sharing smart cards with other applications; direct (MODE\_DIRECT) to get direct control on the reader. Such a method allows also specifying the preferred transmission protocol: T=0 (SmartCard.Protocol\_T0); T=1 (SmartCard.Protocol\_T1); raw (SmartCard.Protocol\_RAW).

The class SmartCard allows sending/receiving APDU to the smart card. By the method send it enables sending Command APDUs and receiving, as result, a Response APDU.

.NET Smart Card API can be download from the author's web site at the following url:

<http://www.ugosweb.com>

After downloading you must install it by clicking on the setup executable.

The following are examples about working with .NET Smart Card API in C# and VB.NET. However, on the author's web site, <http://www.ugosweb.com>, you can find many other samples and manuals about NSCAPI.

## Sending an APDU to the Smart Card

In order to connect to a smart card we must:

- 1) instantiate a SmartCardManager object
- 2) find a reader with a smart card inserted.
- 3) connect to the smart card in the reader found.
- 4) send the Command APDU
- 5) checking the Response APDU

The following listing is related to the previous items

### C#

```
import USCToolkit.NSCAPI;
...

// instantiates smartcard manager
SmartCardManager scman = new SmartCardManager();

// gets the list of active readers (having a smart card plugged-in)
Readers readers = scman.ActiveReaders;

SmartCard smartcard;

// if there isn't any active reader wait for a smart card inserted
if(readers.Count == 0)
{
    // wait for a smart card
    smartcard = scman.WaitForSmartCardInserted();
}
else
{
    // takes the first reader in the list
    Reader reader = readers[0];

    // connects to the plugged-in smart card with default parameters
    SmartCard smartcard = reader.connect();
}

// gets the ATR
Array atr = sc.ATR;

// Creates an APDU
CommandAPDU apdu = new CommandAPDU();
```

```

apdu.CLA = 0x00;
apdu.INS = 0xCA; // GET_DATA
apdu.P1 = 0x00;
apdu.P2 = 0x80;
apdu.LE = 0x00; // entire buffer required

// sends the command APDU and gets the Response APDU
ResponseAPDU resp = smartcard.Send(apdu);

// checks SW1 and SW2 by using Success property
if(resp.Success)
{
    Console.write("SW1: " + resp.SW1);
    Console.write("SW2: " + resp.SW2);
}

```

## VB.NET

```

Imports USCToolkit.NSCAPI
...

Dim scman As SmartCardManager
Dim readers As Readers
Dim reader As Reader
Dim smartcard As SmartCard
Dim atr As Array
Dim cAPDU As CommandAPDU
Dim rAPDU As ResponseAPDU

' gets an instance of SmartCardManager
scman = New SmartCardManager()

' gets active readers (readers with a smart card plugged-in)
readers = scman.ActiveReaders

' if there isn't any reader wait for smart card insertion
If (readers.Count = 0) Then
    'wait
    smartcard = scman.WaitForSmartCardInserted()
Else
    'takes the first reader
    reader = readers(0)
    'connect to the selected reader
    smartcard = reader.Connect()
End If

'gets the ATR
atr = smartcard.ATR

' Creates an APDU
cAPDU = New CommandAPDU
cAPDU.CLA = &H0
cAPDU.INS = &HCA ' GET DATA Command
cAPDU.P1 = &H0
cAPDU.P2 = &H80
cAPDU.LE = &H0 ' entire buffer required

' sends the command APDU and gets the Response APDU
rAPDU = smartcard.Send(cAPDU)

' checks SW1 and SW2 by using Success property
If (rAPDU.Success) Then
    Console.Write("SW1: " & rAPDU.SW1)
    Console.Write("SW2: " & rAPDU.SW2)
End If

```

## Using .NET Smart Card API with virtual smart card emulator

You can try the samples in the previous paragraphs also with the virtual smart card supplied by the emulator (see §3.7). To use the virtual smart card, simply, install the emulator as described in §3.7 and copy the file `winscard.dll`, you can find in the installation directory of the emulator, in the working directory of your C# / VB.NET application.