Ludovic Rousseau's blog

My activities related to smart card and Free Software (as in free speech).

Thursday, April 29, 2010

PCSC sample in Python

Here is the PCSC sample in Python language I promised in PC/SC sample in different languages.

Installation

The wrapper project is hosted by sourceforge at http://pyscard.sourceforge.net/. The files (source code and installer for many different systems: GNU/Linux, Mac OS X, Windows) are available at http://sourceforge.net/projects/pyscard/files/.

But if you have a Debian system it is easier to use:

```
apt-get install python-pyscard
```

The licence is Lesser General Public License either version 2.1 of the License, or (at your option) any later version.

Low level API

Source code

```
#! /usr/bin/env python

from smartcard.scard import *
import smartcard.util

SELECT = [0x00, 0xA4, 0x04, 0x00, 0x0A, 0xA0, 0x00, 0x00, 0x00, 0x62,
    0x03, 0x01, 0x0C, 0x06, 0x01]
COMMAND = [0x00, 0x00, 0x00, 0x00]

try:
    hresult, hcontext = SCardEstablishContext(SCARD_SCOPE_USER)
    if hresult != SCARD_S_SUCCESS:
        raise Exception('Failed to establish context : ' +
            SCardGetErrorMessage(hresult))
    print 'Context established!'

    try:
        hresult, readers = SCardListReaders(hcontext, [])
        if hresult != SCARD_S_SUCCESS:
            raise Exception('Failed to list readers: ' +
                SCardGetErrorMessage(hresult))
        print 'PCSC Readers:', readers

        if len(readers) < 1:
            raise Exception('No smart card readers')

        reader = readers[0]
        print "Using reader:", reader

        try:
            hresult, hcard, dwActiveProtocol = SCardConnect(hcontext, reader,
                SCARD_SHARE_SHARED, SCARD_PROTOCOL_T0 | SCARD_PROTOCOL_T1)
            if hresult != SCARD_S_SUCCESS:
                raise Exception('Unable to connect: ' +
                    SCardGetErrorMessage(hresult))
            print 'Connected with active protocol', dwActiveProtocol
```

Google+ Badge

Ludovic Rousseau blog

G+ Follow

Blog Archive

- ► 2017 (33)
- ► 2016 (49)
- ► 2015 (51)
- ► 2014 (61)
- ► 2013 (38)
- ► 2012 (27)
- ► 2011 (46)
- ▼ 2010 (55)
  - ► December (5)
  - ► November (5)
  - ► October (9)
  - ► September (1)
  - ► August (8)
  - ► July (1)
  - ► June (10)
  - ► May (6)
  - ▼ April (10)
    - PCSC sample in Python
    - Free software Tokend above PKCS#11 (for Mac OS X)
    - Source code of PKCS#11 for .NET cards
    - PCSC sample in Perl
    - My Ohloh page
    - pcsc-lite and CCID beta versions
    - PCSC sample in C
    - CCID driver in Mac OS X
    - PC/SC sample in different languages
    - Parsing an ATR

Search This Blog

Search

Subscribe To

Posts

Comments

Google+ Followers

```
            try:
                hresult, response = SCardTransmit(hcard, dwActiveProtocol,
                    SELECT)
                if hresult != SCARD_S_SUCCESS:
                    raise Exception('Failed to transmit: ' +
                        SCardGetErrorMessage(hresult))
                print 'Select: ' + smartcard.util.toHexString(response,
                    smartcard.util.HEX)
                hresult, response = SCardTransmit(hcard, dwActiveProtocol,
                    COMMAND)
                if hresult != SCARD_S_SUCCESS:
                    raise Exception('Failed to transmit: ' +
                        SCardGetErrorMessage(hresult))
                print 'Command: ' + smartcard.util.toHexString(response,
                    smartcard.util.HEX)
            finally:
                hresult = SCardDisconnect(hcard, SCARD_UNPOWER_CARD)
                if hresult != SCARD_S_SUCCESS:
                    raise Exception('Failed to disconnect: ' +
                        SCardGetErrorMessage(hresult))
                print 'Disconnected'

        except Exception, message:
            print "Exception:", message

    finally:
        hresult = SCardReleaseContext(hcontext)
        if hresult != SCARD_S_SUCCESS:
            raise Exception('Failed to release context: ' +
                    SCardGetErrorMessage(hresult))
        print 'Released context.'

except Exception, message:
    print "Exception:", message

import sys
if 'win32' == sys.platform:
    print 'press Enter to continue'
    sys.stdin.read(1)
```

Output

```
$ ./sample1.py
Context established!
PCSC Readers: ['Gemalto GemPC Pinpad 00 00']
Using reader: Gemalto GemPC Pinpad 00 00
Connected with active protocol 2
Select: 0x90 0x00
Command: 0x48 0x65 0x6C 0x6C 0x6F 0x20 0x77 0x6F 0x72 0x6C 0x64 0x21 0x90 0x00
Disconnected
Released context.
```

Comments

Using the low level API is very verbose. You have access to each PCSC function from Python.

For example I use this API to write some Unitary Tests for pcsc-lite (http://svn.debian.org/wsvn/pcsclite/trunk/PCSC/UnitaryTests/#_trunk_PCSC_UnitaryTests_)

But why use a high level language if the code is as complex as in C?

High level API

Source code

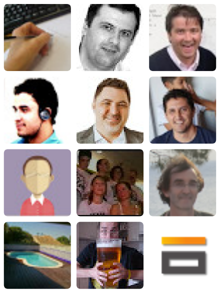
```
#! /usr/bin/env python

from smartcard.System import readers

# define the APDUs used in this script
```

Ludovic Rousseau b...

Follow



336 have us in circles

View all

```
SELECT = [0x00, 0xA4, 0x04, 0x00, 0x0A, 0xA0, 0x00, 0x00, 0x00, 0x62,
    0x03, 0x01, 0x0C, 0x06, 0x01]
COMMAND = [0x00, 0x00, 0x00, 0x00]

# get all the available readers
r = readers()
print "Available readers:", r

reader = r[0]
print "Using:", reader

connection = reader.createConnection()
connection.connect()

data, sw1, sw2 = connection.transmit(SELECT)
print data
print "Select Applet: %02X %02X" % (sw1, sw2)

data, sw1, sw2 = connection.transmit(COMMAND)
print data
print "Command: %02X %02X" % (sw1, sw2)
```

Output

```
$ ./sample2.py
Available readers: ['Gemalto GemPC Pinpad 00 00']
Using: Gemalto GemPC Pinpad 00 00
[]
Select Applet: 90 00
[72, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100, 33]
Command: 90 00
```

Comments

The code is much more compact and easy to read. In particular you do not have to explicitly check for the results. In case of error at the PC/SC level the caller will receive a `smartcard.Exceptions.*` exception.

For example if no card is present in the first reader you get:

```
$ ./sample2.py
Available readers:  ['Gemalto GemPC Pinpad 00 00']
Using: Gemalto GemPC Pinpad 00 00
Traceback (most recent call last):
  File "./sample2.py", line 18, in
    connection.connect()
  File "/usr/lib/python2.5/site-packages/smartcard/CardConnectionDecorator.py", line 5
3, in connect
    self.component.connect( protocol, mode, disposition )
  File "/usr/lib/python2.5/site-packages/smartcard/pcsc/PCSCCardConnection.py", line 1
11, in connect
    raise CardConnectionException( 'Unable to connect with protocol: ' + dictProtocol
[pcscprotocol] + '. ' + SCardGetErrorMessage(hresult) )
smartcard.Exceptions.CardConnectionException: 'Smartcard Exception: Unable to connect
 with protocol: T0 or T1. No smart card inserted.!'
```

It is easy to use a `try:` block to catch the exception and do whatever you need to do in such case.

Conclusion

Python is the language is use nowadays to write programs if I can choose the language. So of course pyscard is the PC/SC wrapper I use.

Flattr

G+

Labels: Python

Newer Post Home Older Post

Bitcoin

License: by-nc-sa