ReadSIM.java

```
// ReadSIM
// Demontration of reading and partly decoding ADNs and SMS from a GSM 11.11 SIM
// File: ReadSIM.java
//
// This program uses the PassThruCardService from OCF opencard.opt.util package.
// Additional informations and reference implementation: www.opencard.org.
// Testet with Towitoko Chipdrive micro on USB port and the appropriate PC/SC dri
// from Towitoko on Windows 98. Testet with Omnikey CardMan Dongle 6020 on USB po
// and the appropriate PC/SC drivers from Omnikey on Windows 2000. Testet with
// JBuilder 6 and JDK 1.3.1 and a GSM 11.11 Smart Card.
//
// Uses OCFPCSC1.DLL, OCFPCSC1.DLL, pcsc-wrapper-src.jar, base-core.jar, base-opt
// from OCF 1.2
// The content of the file "opencard.properties" must be changed to:
// OpenCard.terminals = com.ibm.opencard.terminal.pcsc10.Pcsc10CardTerminalFactor
// For understanding the mechanisms of OCF it is useful to look first into the
// SimpleOCF example.
//
// Remark 1: This example do not support any error mechanism in case of
//           not existing DFs, EFs, wrong file structure, wrong data structure
//           or similar things within the EFs.
// Remark 2: The decoding of the GSM 7 bit alphabet is done by a simple
//           transformation into ASCII, which is not correct for some specific
//           characters.
// Remark 3: The decoding considers only the GSM 7 bit alphabet
//           and not the possible Unicode/UCS-2 codings.
// Remark 4: The SMS message type 5 (SMS already send to the net) and 7 (SMS
//           to send to the net) are not decoded.
// Remark 5: The coding of SMS is specified in GSM 23.040 (availabe from www.etsi
//
// WARNING:  The usage of this programm is at your own risk. The execution of thi
//           could damage your smart card because of using the VERIFY CHV command
//           have to set the right PIN before usage of this propgram. Use only te
//           to work with this program.
//
// This source code is under GNU general public license (see www.opensource.org f
// Please send corrections and ideas for extensions to Wolfgang Rankl (www.wrankl
// Copyright 2003 by Wolfgang Rankl, Munich
//
//  18. Dez. 2003 - V 5, rw: fix another SMS character length calculation bug,
//                           better presentation of SMS content, 5th published ve
//  17. Dez. 2003 - V 4, rw: some clarifications in documentation, implement a be
//                           character decoding, fix SMS character length calcula
//                           add GPL statement, 4th published version
//  23. Jan. 2003 - V 3, rw: test with new card reader, review of the source code
//                           3rd published version
//  14. Nov. 2002 - V 2, rw: decoding of the ADN and partly SMS,
//                           better conversion from unsign byte to integer,
//                           2nd published version
//  30. Oct. 2002 - V 1, rw: initial runnable version, 1st published version
//------------------------------------------------------------------------------

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import opencard.core.service.*;
import opencard.core.terminal.*;
import opencard.opt.util.*;

public class ReadSIM implements ActionListener {
  static final boolean DEBUG = false;    // true: show APDUs; false: show file co
```

```
static final int THE_SAME = 0;          // used for comparisons
static final int IFD_TIMEOUT = 0;       // unit: seconds
static final int MAX_APDU_SIZE = 250;   // unit: byte
static final int CAPDU_P1 = 2;          // index of P1 within a command APDU
static final int CAPDU_LE = 4;          // index of Le within a command APDU
static final int CAPDU_D1 = 4 + 1;      // index of data byte 1 Le within a comm
static final int CAPDU_D2 = 4 + 2;      // index of data byte 2 Le within a comm

// command SELECT FILE with 2 byte FID, according to GSM 11.11
// CLA || INS || P1 || P2 || Lc || DATA 1 (= FID high) || DATA 2 (= FID low)
static final byte[] CMD_SELECT_BY_FID = {(byte)0xA0, (byte)0xA4,
                                         (byte)0x00, (byte)0x00, (byte)0x02,
                                         (byte)0x00, (byte)0x00 };

// command READ RECORD, according to GSM 11.11
// CLA || INS || P1 (record number) || P2 (mode) || Le (record length)
static final byte[] CMD_READ_RECORD = {(byte)0xA0, (byte)0xB2,
                                       (byte)0x00, (byte)0x04, (byte)0x00 };

// command GET RESPONSE, according to GSM 11.11
// CLA || INS || P1 || P2 || Le
static final byte[] CMD_GET_RESPONSE = {(byte)0xA0, (byte)0xC0,
                                        (byte)0x00, (byte)0x00, (byte)0x00 };

// command VERIFY CHV, according to GSM 11.11
// verify CHV 1 with CHV (= PIN) = "1234"
// CLA || INS || P1 || P2 || Lc || DATA 1 ... DATA 8 (= PIN)
// Format PIN: 4 byte ASCII left justified padded with 'FF'
static final byte[] CMD_VERIFY_CHV = {(byte)0xA0, (byte)0x20,
                                      (byte)0x00, (byte)0x01, (byte)0x08,
                                      (byte)0x31, (byte)0x32, (byte)0x33, (byte
                                      (byte)0xFF, (byte)0xFF, (byte)0xFF, (byte

// FID of DF Telecom
static final byte[] FID_DF_TELECOM = {(byte)0x7F, (byte)0x10};

// FID of EF ADN
static final byte[] FID_EF_ADN = {(byte)0x6F, (byte)0x3A};

// FID of EF SMS
static final byte[] FID_EF_SMS = {(byte)0x6F, (byte)0x3C};

// Returncode '9Fxx': command propper executed, xx byte data available
static final byte RC_CMD_PROPPER_EXEC = (byte) 0x9F;

// pointer to the number of records (1 byte) within the response of a SELECT FI
static final int R_SELECTFILE_NO_OF_RECORDS = 15-1;

// pointer to the file size (2 byte) within the response of a SELECT FILE
static final int R_SELECTFILE_FILE_SIZE = 3-1;

static private int n, i_fid;
static private int FileSize, RecordLength, NoOfRecords;
static private byte[] i;
private PassThruCardService ptcs;
private Properties sysProps;
private Frame f;
private Panel p;
private static TextArea t;
private Button b;
static private ResponseAPDU response;
static private CommandAPDU command;

public ReadSIM() {     // build the UI
```

```
    f = new Frame();
    f.setTitle("Message Window for ReadSIM");
    p = new Panel();
    p.setLayout(new FlowLayout(FlowLayout.CENTER, 0, 0));
    t = new TextArea(40, 110);
    t.setFont(new Font("Monospaced", 0, 12));   // Courier, 12 point size
    t.setBackground(new Color(0, 0, 0));        // black
    t.setForeground(new Color(200, 255, 120));  // yellowgreen
    p.add(t);
    b = new Button("Exit");
    b.addActionListener(this);
    p.add(b);
    f.add(p);
    f.pack();
    f.show();
  } // ReadSIM

  public void actionPerformed(ActionEvent evt) {
  // exit programm if exit button is pressed
    if (evt.getSource() == b) System.exit(0);
  } // actionPerformed

  public static void showText(String Text) {
    t.append(Text);
  } // showText

  public void showATR(byte[] ATR) {
    int n, x;
    String s = new String();

    t.append("ATR: ");
    for (n=0; n<i.length; n++) {
      x = (int) (0x000000FF & ATR[n]);
      s = Integer.toHexString(x).toUpperCase();
      if (s.length()== 1) s = "0" + s;
      t.append(s + " ");
    }  // for
    t.append("\n");
  } // showATR

  public String decodeADNRecord(byte[] RspAPDU, int LenRspAPDU) {
    int x, n, LenRecord, LenName, LenNumber;
    String b = new String();
    String c = new String();
    String s = new String("");

    x = (int) (0x000000FF & RspAPDU[0]);
    if (x == 0xFF) return s;

    LenRecord = LenRspAPDU - 2;     // length of the record = APDU length without
    LenName = LenRecord - 14;
    LenNumber = RspAPDU[LenName];  // read length of dialing number from the ADN

    s = "Name: ";
    for (n = 0; n < LenName; n++) {
      x = (int) (0x000000FF & RspAPDU[n]);
      if (x == 0xFF) break;            // end of dialing number reached
      s = s + convertGSM2Unicode(x);  // conversion: 7 bit GSM character -> Java
    }  // for

    s = s + "  Dialing Number: ";
    for (n = LenName + 1; n < LenName + 1 + LenNumber; n++) {
      x = (int) (0x000000FF & RspAPDU[n]);
      if (x == 0xFF) break;
```

```
      c = Integer.toHexString(x).toUpperCase();
      if (c.length()== 1) c = "0" + c;
      if (n == (LenName + 1)) {
        // it is the 1st value of the dialing numbers
        if (c.compareTo("91") == THE_SAME) c = "+";  // it is an international di
        if (c.compareTo("81") == THE_SAME) c = "";   // it is a standard (ISDN) d
      } // if
      else {
        // if it is not the 1st value of the dialing number -> swap high and low
        b = c.substring(1,2);
        b = b + c.substring(0,1);
        c = b;
      } // else
      s = s + c + " ";
    } // for
    return s;
  } // decodeADNRecord

  public String decodeSMSRecord(byte[] RspAPDU, int LenRspAPDU) {
    int x, n, y, Bytebefore, Bitshift, LenRecord, EndOfSms = 0;
    int MessageStart, LenMessage;
    String b = new String();
    String c = new String();
    String s = new String("");

    LenRecord = LenRspAPDU - 2;  // length of the record = APDU length without SW

    //----- look for the end of the SMS
    //----- don't convert the unused characters (= 0xFF) at the end of the SMS
    for (n = LenRecord ; n > 0; n--) {
      EndOfSms = n;
      x = (int) (0x000000FF & RspAPDU[n-1]);
      if (x != 0xFF) break;
    }  // for

    x = (int) (0x000000FF & RspAPDU[0]);
    if (x == 0x00) {
      s = "unused record";
      return s;
    } // if
    else if (x == 0x01) s = "SMS from net and read:      ";
    else if (x == 0x03) s = "SMS from net and to read:   ";
    else if (x == 0x05) s = "SMS already send to the net";
    else if (x == 0x07) s = "SMS to send to the net";
    else s = "not specified SMS status";

    if (x == 0x01 | x == 0x03) {              // the following decodes only a SMS
      //----- find the beginning of the SMS message
      MessageStart = 1;                       // consider SMS status byte
      x = (int) (0x000000FF & RspAPDU[1]);    // length nof SMSC dialing number
      MessageStart = MessageStart + 1 + x;
      MessageStart = MessageStart + 1;       // control information
      x = RspAPDU[MessageStart];             // number of digits of the sender
      x = x / 2;                             // conversion: digits -> bytes
      x = x + 2;                             // add 2 byte for number of digits o
      MessageStart = MessageStart + x;
      MessageStart = MessageStart + 1 + 1;   // message type, message alphabet
      MessageStart = MessageStart + 7;       // time stamp
      x = (int) (0x000000FF & RspAPDU[MessageStart]); // length of SMS message in

      LenMessage = x;

      //----- copy SMS message from APDU field into a SMS field
      int SMS[] = new int[LenMessage + 1];
```

```
      for (n = 1; n <= LenMessage; n++) {
        // convertion: unsigned byte -> integer
        SMS[n] = (int) (0x000000FF & RspAPDU[MessageStart + n]);
      } // for

      //----- convert the whole SMS record
      Bytebefore = 0;
      for (n = 1; n <= LenMessage; n++) {
        x = SMS[n] ;                      // get a byte from the SMS
        Bitshift = (n-1) % 7;             // calculate number of neccssary bit shif
        y = x;
        y = y << Bitshift;                // shift to get a conversion 7 bit compac
        y = y | Bytebefore;               // add bits from the byte before this byt
        y = y & 0x0000007F;               // delete all bits except bit 7 ... 1 of
        s = s + convertGSM2Unicode(y);    // conversion: 7 bit GSM character -> Jav
        if (Bitshift == 6) {
          Bitshift = 1;
          y = x;
          y = y >>> Bitshift;              // shift to get a conversion 7 bit compa
          y = y & 0x0000007F;              // delete all bits except bit 7 ... 1 of
          s = s + convertGSM2Unicode(y); // conversion: 7 bit GSM character -> Ja
          Bytebefore = 0;
        } // if
        else {
          Bytebefore = x;
          Bitshift = 7 - Bitshift;
          Bytebefore = Bytebefore >>> Bitshift;  // shift to get a conversion 7 b
          Bytebefore = Bytebefore & 0x000000FF;  // mask for one byte
        } // else
      }  // for EndOfSms
    } // if
    return s;
  }  // decodeSMSRecord

  /**
   * Convert one GSM standard alphabet character into a Unicode character
   * @param b one GSM standard alphabet character
   * @return one Unicode character
   * The simple approch for a 7 bit GSM character to Java Unicode
   * conversion is to use the following code fragment: "s = s + (char) y;"
   * This conversion do not consider special characters and therefore the convert
   * text will look sometime a little bit strange.
   */
  public static char convertGSM2Unicode(int b) {
    char c;

    if ((b >= 0x41) && (b <= 0x5A)) {    // character is between "A" and "Z"
      c = (char) b;
      return c;
    }  // if
    if ((b >= 0x61) && (b <= 0x7A)) {    // character is between "a" and "z"
      c = (char) b;
      return c;
    }  // if
    if ((b >= 0x30) && (b <= 0x39)) {    // character is between "0" and "9"
      c = (char) b;
      return c;
    }  // if

    switch (b) {
      case 0x00 : c = '@'; break;
      case 0x02 : c = '$'; break;
      case 0x0A : c = '\n'; break;
      case 0x0D : c = '\r'; break;
```

```
      case 0x11 : c = '_'; break;
      case 0x1E : c = 'ß'; break;
      case 0x20 : c = ' '; break;
      case 0x21 : c = '!'; break;
      case 0x22 : c = '\"'; break;
      case 0x23 : c = '#'; break;
      case 0x25 : c = '%'; break;
      case 0x26 : c = '&'; break;
      case 0x27 : c = '\''; break;
      case 0x28 : c = '('; break;
      case 0x29 : c = ')'; break;
      case 0x2A : c = '*'; break;
      case 0x2B : c = '+'; break;
      case 0x2C : c = ','; break;
      case 0x2D : c = '-'; break;
      case 0x2E : c = '.'; break;
      case 0x2F : c = '/'; break;
      case 0x3A : c = ':'; break;
      case 0x3B : c = ';'; break;
      case 0x3C : c = '<'; break;
      case 0x3D : c = '='; break;
      case 0x3E : c = '>'; break;
      case 0x3F : c = '?'; break;
      case 0x5B : c = 'Ä'; break;
      case 0x5C : c = 'Ö'; break;
      case 0x5E : c = 'Ü'; break;
      case 0x5F : c = '§'; break;
      case 0x7B : c = 'ä'; break;
      case 0x7C : c = 'ö'; break;
      case 0x7E : c = 'ü'; break;
      default:    c = ' '; break; // unknown character
    }  // switch
    return c;
  }  // convertGSM2Unicode

  public void showCmdAPDU(byte[] CmdAPDU, int LenCmdAPDU) {
    int n, x;
    String s = new String();

    t.append("IFD->ICC: ");
    for (n=0; n<LenCmdAPDU; n++) {
      x = (int) (0x000000FF & CmdAPDU[n]);
      s = Integer.toHexString(x).toUpperCase();
      if (s.length() == 1) s = "0" + s;
      t.append(s + " ");
    }  // for
    t.append("\n");
   } // showCmdAPDU

  public void showRspAPDU(byte[] RspAPDU, int LenRspAPDU) {
    int n, x;
    String s = new String();

    t.append("ICC->IFD: ");
    for (n=0; n<LenRspAPDU; n++) {
      x = (int) (0x000000FF & RspAPDU[n]);
      s = Integer.toHexString(x).toUpperCase();
      if (s.length()== 1) s = "0" + s;
      t.append(s + " ");
    }  // for
    t.append("\n");
  } // showRspAPDU

  public static void main( String [] args ){
```

```
ReadSIM rsim = new ReadSIM();
rsim.showText("Start Program: ReadSIM\n\n");
// get and set system properties for OCF, PC/SC and PassThruCardServce
Properties sysProps = System.getProperties();
sysProps.put ("OpenCard.terminals", "com.ibm.opencard.terminal.pcsc10.Pcsc10C
sysProps.put ("OpenCard.services", "opencard.opt.util.PassThruCardServiceFact

try {
  SmartCard.start();
  CardRequest cr = new CardRequest(CardRequest.ANYCARD, null, PassThruCardSer
  cr.setTimeout(IFD_TIMEOUT);                    // set timeout for IFD
  SmartCard sc = SmartCard.waitForCard(cr);    // wait  for ICC in the IFD
  rsim.showText("INFO: activate smart card\n\n");

  if (sc != null) {  // no error occur, a smart card is in the terminal
    PassThruCardService ptcs = (PassThruCardService) sc.getCardService(PassTh
    command = new CommandAPDU(MAX_APDU_SIZE); // set APDU buffer size

    // get ATR
    CardID cardID = sc.getCardID();
    i = cardID.getATR();
    rsim.showATR(i);

    //----- verify the PIN for access to EF SMS and EF ADN
    // VERIFY CHV ( = PIN)
    rsim.showText("VERIFY CHV\n");
    command.setLength(0);
    command.append(CMD_VERIFY_CHV);
    if (DEBUG == true) rsim.showCmdAPDU(command.getBytes(), command.getLength
    response = ptcs.sendCommandAPDU(command);
    if (DEBUG == true) rsim.showRspAPDU(response.getBytes(),response.getLengt

    // ----- read and display all SMS (short messages) -----
    // select DF Telecom
    rsim.showText("\nSELECT DF Telecom\n");
    command.setLength(0);
    command.append(CMD_SELECT_BY_FID);
    command.setByte(CAPDU_D1, FID_DF_TELECOM[0]);    // set FID high
    command.setByte(CAPDU_D2, FID_DF_TELECOM[1]);    // set FID low
    if (DEBUG == true) rsim.showCmdAPDU(command.getBytes(), command.getLength
    response = ptcs.sendCommandAPDU(command);
    if (DEBUG == true) rsim.showRspAPDU(response.getBytes(),response.getLengt
    // select EF SMS
    command.setLength(0);
    rsim.showText("\nSELECT EF SMS\n");
    command.append(CMD_SELECT_BY_FID);
    command.setByte(CAPDU_D1, FID_EF_SMS[0]);    // set FID high
    command.setByte(CAPDU_D2, FID_EF_SMS[1]);    // set FID low
    if (DEBUG == true) rsim.showCmdAPDU(command.getBytes(), command.getLength
    response = ptcs.sendCommandAPDU(command);
    if (DEBUG == true) rsim.showRspAPDU(response.getBytes(),response.getLengt
    // send GET RESPONSE and fetch the data from the previous SELECT FILE com
    rsim.showText("GET RESPONSE\n");
    command.setLength(0);
    command.append(CMD_GET_RESPONSE);
    command.setByte(CAPDU_LE, response.getByte(1));    // set number of byte t
    if (DEBUG == true) rsim.showCmdAPDU(command.getBytes(), command.getLength
    response = ptcs.sendCommandAPDU(command);
    if (DEBUG == true) rsim.showRspAPDU(response.getBytes(),response.getLengt
    command.setByte(CAPDU_LE, response.getByte(1));    // set number of byte t
    FileSize = response.getByte(R_SELECTFILE_FILE_SIZE)*256 + response.getByt
    RecordLength = response.getByte(R_SELECTFILE_NO_OF_RECORDS);
    NoOfRecords = FileSize/RecordLength;
    rsim.showText("EF SMS, size: " + FileSize + " bytes\n");
```

```
rsim.showText("EF SMS, record length: " + RecordLength + " bytes\n");
rsim.showText("EF SMS, number of records: " + NoOfRecords + "\n");

// this is the loop for reading all records within the EF SMS
rsim.showText("INFO: read content of EF SMS\n");
String SMSRecord = new String();
for (n = 1; n <= NoOfRecords; n++) {          // loop, to read all record
  command.setLength(0);
  command.append(CMD_READ_RECORD);
  command.setByte(CAPDU_P1, n);               // set record number
  command.setByte(CAPDU_LE, RecordLength);    // set record length = byte
  if (DEBUG == true) rsim.showCmdAPDU(command.getBytes(), command.getLeng
  response = ptcs.sendCommandAPDU(command);
  if (DEBUG == true) rsim.showRspAPDU(response.getBytes(),response.getLen
  rsim.showText("Record " + n + ": ");
  SMSRecord = rsim.decodeSMSRecord(response.getBytes(),response.getLength
  if (SMSRecord != "") rsim.showText("\t" + SMSRecord + "\n");
  else rsim.showText("\tempty\n");
} // for

// ----- read and decode all ADNs (abbreviated dialing numbers) -----
// select EF ADN
command.setLength(0);
rsim.showText("\n\nSELECT EF ADN, ");
command.append(CMD_SELECT_BY_FID);
command.setByte(CAPDU_D1, FID_EF_ADN[0]);   // set FID high
command.setByte(CAPDU_D2, FID_EF_ADN[1]);   // set FID low
if (DEBUG == true) rsim.showCmdAPDU(command.getBytes(), command.getLength
response = ptcs.sendCommandAPDU(command);
if (DEBUG == true) rsim.showRspAPDU(response.getBytes(),response.getLengt
// send GET RESPONSE and fetch the data from the previous SELECT FILE com
rsim.showText("GET RESPONSE\n");
command.setLength(0);
command.append(CMD_GET_RESPONSE);
command.setByte(CAPDU_LE, response.getByte(1));   // set number of byte t
if (DEBUG == true) rsim.showCmdAPDU(command.getBytes(), command.getLength
response = ptcs.sendCommandAPDU(command);
if (DEBUG == true) rsim.showRspAPDU(response.getBytes(),response.getLengt
command.setByte(CAPDU_LE, response.getByte(1));   // set number of byte t
FileSize = response.getByte(R_SELECTFILE_FILE_SIZE)*256 + response.getByt
RecordLength = response.getByte(R_SELECTFILE_NO_OF_RECORDS);
NoOfRecords = FileSize/RecordLength;
rsim.showText("  EF ADN, size: " + FileSize + " bytes\n");
rsim.showText("  EF ADN, record length: " + RecordLength + " bytes\n");
rsim.showText("  EF ADN, number of records: " + NoOfRecords + "\n");
// this is the loop for reading all records within the EF ADN
rsim.showText("INFO: read content of EF ADN\n");
String ADNRecord = new String();
for (n = 1; n <= NoOfRecords; n++) {   // loop, to read all records
  command.setLength(0);
  command.append(CMD_READ_RECORD);
  command.setByte(CAPDU_P1, n);   // set record number
  command.setByte(CAPDU_LE, RecordLength);   // set record length = byte
  if (DEBUG == true) rsim.showCmdAPDU(command.getBytes(), command.getLeng
  response = ptcs.sendCommandAPDU(command);
  if (DEBUG == true) rsim.showRspAPDU(response.getBytes(),response.getLen
  rsim.showText("Record " + n + ": ");
  ADNRecord = rsim.decodeADNRecord(response.getBytes(),response.getLength
  if (ADNRecord.length() != 0) rsim.showText("\t" + ADNRecord + "\n");
  else rsim.showText("\tempty\n");
} // for

rsim.showText("\nINFO: deactivate smart card\n");
SmartCard.shutdown();
```

```
      } // if
      else
        rsim.showText("\n\ncould not create smart card object (e.g. no ICC in IFD
    } // try
    catch (Exception except) {
        rsim.showText("\n\nCaught exception '" + except.getClass() + "' - " + ex
    } // catch
  } // main
} // class
```

ReadSIM.java