**Die Klasse EmuCheckConnection**

**import net.sf.yad2xx.\*;**

**public class EmuCheckConnection {**

**// Es gibt eine Klasse Device. Das Attribut device vom Typ  
 // Device soll Eigenschaften und Methoden zu dem  
 // angeschlossenen Leistungsmessgeraet enthalten.**

**private Device device = null;  
 // Attribut zur Regelung des Threads, siehe unten**

**private boolean connected = true;**

**public EmuCheckConnection() throws ??? {  
 // Listen Sie alle angeschlossenen Geraete auf.  
 // Es gibt eine Klasse FTDIInterface, welche eine  
 // statische Methode getDevices() enthaelt, welche  
 // ein Array von angeschlossenen Geraeten als   
 // Rueckgabewert hat.**

**???**

**System.*out*.println("Anzahl gefundener FTD2 Devices: "**

**+ devices.length);**

**// Das Device, dessen Beschreibung mit NZR anfaengt, ist   
 // das Leistungsmessgeraet. Das Attribut device soll  
 // damit belegt werden. Die Beschreibung eines Devices  
 // ermitteln Sie mit Hilfe der Methode getDescripton()   
 // der Klasse Device mit Rueckgabewert vom Typ String.  
 ???  
 this.device.open();**

**// Setzen Sie die DatenCharacteristika auf 7 DataBits,**

**// 1 StopBit, gerade Paritaet (2).  
 // Dann setzen Sie die Baudrate auf 300.**

**// Geben Sie die Beschreibung des Devices heraus.  
 System.*out*.println("Verbunden mit Device: "**

**+ ???);**

**}**

**public void connect()throws ??? {**

**byte[] start**

**= new byte[]{0x2F, 0x3F, 0x21, 0x0D, 0x0A};**

**This.device.write(start);**

**System.*out*.println("Einleitung Kommunikation USBCheck");**

**}**

**public void disconnect()throws ??? {**

**this.connected = false;**

**byte[] ende = new byte[]{0x01, 0x42, 0x30, 0x03};**

**this.device.write(ende);**

**this.device.close();**

**System.*out*.println("Ende Kommunikation USBCheck");**

**}**

**public void sendProgrammingMode()throws ??? {**

**byte[] nachricht = new byte[]**

**{0x06, 0x30, 0x30, 0x31, 0x0D, 0x0A};**

**this.device.write(nachricht);**

**System.*out*.println("Programming Mode");**

**}**

**public void sendRequest(MESSWERT m)**

**throws FTD2xxException{**

**device.write(m.getRequest());**

**System.*out*.println("Request " + m.getObis() + " "**

**+ m.toString());**

**}**

**// Damit das Device staendig liest, muss die Klasse  
 // EmuCheckConnection von der Klasse Thread ableiten. Das  
 // Thread wird am Ende des Konstruktors gestartet. Weiterhin  
 // muss die Methode run ueberschrieben werden.**

**public void run() {**

**// Das Device soll so lange Leseversuche durchfuehren,**

**// wie die Verbindung existiert (, siehe Attribut**

**// connected). Es soll nur gelesen werden, falls der**

**// QueueStatus des devices ungleich 0 ist. Den**

**// gelesenen Wert finden Sie in dem Byte-Array der**

**// Laenge 1, welcher als Parameter mitgegeben wird.**

**// Dieser wird in in die Konsole geschrieben.  
 // Dazu konvertieren Sie ihn vorher in ein char-Wert.**

**???**

**}**

**}**

**Übersicht über die Klassen aus net.sf.yad2xx.\***

**Klasse FTDIInterface**

/\*\*

\* A Java Native Interface (JNI) wrapper that adapts the FTDI

\* D2XX library to a more OO based approach. This Singleton type

\* provides a very thin Java layer over the top of the native C

\* language code and is central to the libraries operation.  
 \*/

**Initialisierungsblock**

/\*\*

\* Loads the native library on first class usage. Library location

\* is JVM/platform dependent.

\*/

static {

System.loadLibrary("FTDIInterface");

}

**Methode**

/\*\*

\* Returns an array of Devices which describe attached D2XX devices.

\* Combines calls to FT\_CreateDeviceInfoList and   
 \* FT\_GetDeviceInfoList. <p>

\* Copies values returned from FT\_GetDeviceInfoList into individual   
 \* Device objects.

\*

\* @return array of Devices describing attached D2XX devices

\*

\* @throws FTDIException FT\_CreateDeviceInfoList returned a non-  
 \* zero

\*/

public static native Device[] getDevices() throws FTDIException;

**Klasse Device**

**Methoden**

/\*\*

\* Return the device description.

\*

\* @return device description

\*/

public String getDescription() {

return description;

}

/\*\*

\* Begin a session with the device.

\*

\* @throws IllegalStateException if device already open

\* @throws FTDIException D2XX API call failed

\*/

public void open() throws FTDIException {

if (isOpen())

throw new IllegalStateException("Device in use");

FTDIInterface.open(this);

}

/\*\*

\* Close the device.

\*

\* @throws FTDIException D2XX API call failed,

\*/

public void close() throws FTDIException {

if (ftHandle != 0) {

FTDIInterface.close(this);

}

}

/\*\*

\* Sets the data characteristics for the device.

\*

\* @param wordLength number of bits per word, must be 8 or 7

\* @param stopBits number of stop bits, must be 1 or 2

\* @param parity must be one of the parity values e.g. 0 or 1 or 2

\*

\* @throws FTDIException D2XX API call failed,

\*/

public void setDataCharacteristics(byte wordLength, byte stopBits,  
 byte parity) throws FTDIException {

FTDIInterface.setDataCharacteristics(ftHandle, wordLength,   
 stopBits, parity);

}

/\*\*  
 \* Sets the baud rate for the device.

\*

\* @param baudRate baud rate

\* @throws FTDIException D2XX API call failed

\*/

public void setBaudRate(int baudRate) throws FTDIException {

FTDIInterface.setBaudRate(ftHandle, baudRate);

}

/\*\*  
 \* Write data to the device.  
 \*   
 \* @param buffer bytes to write to device  
 \*  
 \* @return number of bytes actually written  
 \*  
 \* @throws FTDIException D2XX API call failed  
 \*/  
public int write(byte[] buffer) throws FTDIException {  
 return write(buffer, buffer.length);  
}

/\*\*

\* Returns number of bytes in receive queue.

\*

\* @return number of bytes in receive queue

\*

\* @throws FTDIException D2XX API call failed

\*/

public int getQueueStatus() throws FTDIException {

return FTDIInterface.getQueueStatus(ftHandle);

}

/\*\*  
 \* Reads data from device up to the size of the buffer. Note that   
 \* this call will block if the requested number of bytes is not  
 \* immediately available. Call getQueueStatus to get the number of  
 \* bytes actually available to avoid blocking.   
 \*

\* @param buffer bytes read from device. Buffer  
 \* length determines maximum number of bytes read  
 \*

\* @return number of bytes actually read  
 \* @throws FTDIException D2XX API call failed  
 \*/

public int read(byte[] buffer) throws FTDIException {

return FTDIInterface.read(ftHandle, buffer, buffer.length);

}