JavaPOS Driver

version 1.0h

Table of content

1. About

This driver is designed to provide support for data interchange between POS system compatible with JavaPOS device driver standard and physical devices produced by Novitus.
The driver consists of jar file containing the driver (javapos-1.0.jar) and files containing driver dependencies (commons-lang3-3.5.jar javapos-1.14.2.jar jSerialComm-2.4.0.jar log4j-api-2.11.2.jar log4j-core-2.11.2.jar). All the dependencies along with the main driver .jar file have to be on Java CLASSPATH in order to properly load and enable the driver.
In addition to the above, jpos.xml file containing definition of JavaPOS devices has to be properly configured and available to the application that utilizes the driver.
The driver has been developed and tested using Java JDK 8, however, it does not use any of Java 8 features, therefore it should be possible to use it also with previous versions of Java JDK – JDK 7 and JDK 6. Upon reuquest, the driver build using SDK 7 or SDK 6 can also be provided.

2. Configuration file

Driver is configured in a standard way for JavaPOS device drivers through entry in jpos.xml file. The single configuration entry consists of JavaPOS logical device entry containing logical device name, factory and service class names and configuration parameters defining type of connection to the device and parameters of the connection.

The below shows an example jpos.xml file containing entry for one Hde device configured to be accessible through serial port connection.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE JposEntries PUBLIC "-//JavaPOS//DTD//EN"
                             "jpos/res/jcl.dtd">
<JposEntries>

    <JposEntry logicalName="Hde-serial">
        <creation factoryClass="pl.novitus.javapos.factory.DefaultServiceFactory"
serviceClass="pl.novitus.javapos.service.HDeFiscalPrinter"/>
        <vendor name="Novitus" url="http://www.novitus.pl"/>
        <jpos category="FiscalPrinter" version="1.14"/>
        <product description="HDe Fiscal Printer" name="Novitus HDe Fiscal Printer Service for
JavaPOS(TM) Standard" url="http://www.javapos.com"/>
                              <prop name="connection" value="serial"/>
                              <prop name="port" value="ttyACM0"/>
                              <prop name="baudRate" value="9600"/>
                              <prop name="parity" value="none"/>
                              <prop name="stopBits" value="1"/>
                              <prop name="bits" value="8"/>
                              <prop name="flowControl" value="xon/xoff"/>
    </JposEntry>

</JposEntries>
```

There may be many *JposEntry* definitions for different devices and for same device with different connection parameters. The only requirement is that *logicalName* is unique.

3. Supported devices

The driver currently supports the following devices:

Device	Driver service class	Driver factory class
HD Online Fiscal Printer	pl.novitus.javapos.service.HDeFiscalPrinter	pl.novitus.javapos.factory.DefaultServiceFactory

4. HD Online Fiscal Printer

Supports Novitus HD Online fiscal printer.

4.1 Connection types

Connection type	Parameter	Required	Default value	Meaning
serial *(serial port)*	port	yes		System name of serial port to use to connect with the device. On Windows, the name is like COMx (COM1, …) On Linux, this is the serial device name, like ttyS0 (RS-2332) or ttyACM0 (USB serial ports)
	baudRate	no	9600	Baud rate of the connection
	parity	no	N	Parity: N – none E – even O – odd
	stopBits	no	1	Number of stop bits (1, 1.5, 2)
	bits	no	8	Number of character bits
	flowControl	no	xonxoff	Type of flow control (xonxoff, dtrdsr, ctsrts or none)
	timeout	no	5000	Communication timeout in milliseconds
tcp *(tcp/ip protocol)*	port	yes		Port of the device
	server	yes		IP/serven name of the device
	timeout	no	5000	Communication timeout in milliseconds

To list available serial port names, call the driver in the following way:

```
java -cp <classpath> pl.novitus.javapos.utils.connection.SerialPortConnection list
```

where <classpath> is java classpath containing all the necessary driver .jar files as described in part 1.

The output may look as follows:

```
SerialPortConnection: showPorts
Serial ports available:

portName         description                    information
--------------------------------------------------------------------------------
ttyACM0           Novitus Virtual COM port       Novitus Virtual COM port
ttyACM1           Novitus Virtual COM port       Novitus Virtual COM port

Found 2 port(s).
```

4.2 Asynchronous mode

Current version of driver does not support ansynchronous modes.

4.3 Supported methods

The following methods are supported by current version of the driver (1.0b):

Method	S	Remarks
open	√	
close	√	
claim	√	
release	√	
checkHealth		
clearInput		
clearInputProperties		
clearOutput		
directIO	√	Following commands are supported (defined in `pl.novitus.javapos.Consts.FiscalPrinter.DirectIO` interface). Result is returned via data argument, which is expected to be initialized to empty `ArrayList<Byte>`. DIO_READPACKET: Read packet from fiscal printer and returns its content (without packet marker characters and checksum) as data parameter. Packet checksum is expected and verified. DIO_READPACKET_NOCS: Read packet from fiscal printer and returns its content (without packet marker characters and checksum) as data parameter. Packet checksum is not expected and not verified. DIO_WRITEPACKET: Sends data as packet to the fiscal printer. Packet markers and checksum are added automatically.

		<table><tr><td>DIO_STATUS_ENQ</td><td>Sends ENQ query to the fiscal printer and returns status byte received in first element of data array.</td></tr><tr><td>DIO_STATUS_DLE</td><td>Sends DLE query to the fiscal printer and returns status byte received in first element of data array.</td></tr><tr><td>DIO_BEEP</td><td>Sends BEL character to the fiscal printer, doesn't return any results.</td></tr><tr><td>DIO_CAN</td><td>Sends CAN character to the fiscal printer, doesn't return any results.</td></tr><tr><td>DIO_SETTIMEOUT</td><td>Sets connection timeout. Value of timeout in seconds should be passed as first element of data array.</td></tr></table>
compareFirmwareVersion		
resetStatistics		
retrieveStatistics		
updateFirmware		
updateStatistics		
setCurrency		
setDate	√	
setHeaderLine		
setPOSID	√	
setStoreFiscalID		
setTrailerLine		
setVatTable	√	
setVatValue	√	
beginFiscalReceipt	√	Supported types: FPTR_RT_SALES, FPTR_RT_CASH_IN, FPTR_RT_CASH_OUT, FPTR_RT_SIMPLE_INVOICE
endFiscalReceipt	√	
printDuplicateReceipt		
printRecCash	√	Supported types: FPTR_RT_CASH_IN, FPTR_RT_CASH_OUT
printRecItem	√	There's a buffer for receipt item in the driver to support item adjustments, so the item sent to the driver may be not printed immediately. This method also provides support for refundable packages. An item is treated as package when its name starts with '#' character and is numeric value between 0 and 9999. Package sent through this method is counted as package sale (see receipt example).
printRecItemVoid	√	
printRecItemAdjustment	√	
printRecItemAdjustmentVoid	√	
printRecItemFuel		
printRecItemFuelVoid		
printRecItemRefund	√	Supports package returns. An item is treated as package when its name starts with '#' character and is numeric value between 0 and 9999. Package sent through this method is counted as package return (see receipt example)
printRecItemRefundVoid		
printRecMessage	√	
printRecPackageAdjustment	√	
printRecPackageAdjustVoid		

<table>
<tr><td>printRecRefund</td><td></td><td></td></tr>
<tr><td>printRecRefundVoid</td><td></td><td></td></tr>
<tr><td>printRecSubtotal</td><td>√</td><td></td></tr>
<tr><td>printRecSubtotalAdjustment</td><td></td><td></td></tr>
<tr><td>printRecSubtotalAdjustVoid</td><td></td><td></td></tr>
<tr><td>printRecTaxID</td><td></td><td></td></tr>
<tr><td>printRecTotal</td><td>√</td><td>Payment type and required payment parameters are sent through description parameter following the schema:

paymentType[;paymentParam1[;paymentParam2[;…]]]

The following table shows allowed payment types (defined in Consts.FiscalPrinter.PaymentType) and their parameters:

<table>
<tr><th>payment type</th><th>parameters</th></tr>
<tr><td>0 (cash)</td><td>-</td></tr>
<tr><td>1 (card)</td><td><table><tr><th>parameter</th><th>meaning</th></tr><tr><td>description</td><td>description of the payment</td></tr></table></td></tr>
<tr><td>2 (check)</td><td><table><tr><th>parameter</th><th>meaning</th></tr><tr><td>description</td><td>description of the payment</td></tr></table></td></tr>
<tr><td>3 (voucher)</td><td><table><tr><th>parameter</th><th>meaning</th></tr><tr><td>description</td><td>description of the payment</td></tr></table></td></tr>
<tr><td>4 (other)</td><td><table><tr><th>parameter</th><th>meaning</th></tr><tr><td>description</td><td>description of the payment</td></tr></table></td></tr>
<tr><td>5 (credit)</td><td><table><tr><th>parameter</th><th>meaning</th></tr><tr><td>description</td><td>description of the payment</td></tr></table></td></tr>
<tr><td>6 (transfer)</td><td><table><tr><th>parameter</th><th>meaning</th></tr><tr><td>description</td><td>description of the payment</td></tr></table></td></tr>
<tr><td>7 (currency)</td><td><table><tr><th>parameter</th><th>meaning</th></tr><tr><td>currName</td><td>name of the currency (ie. EUR or USD)</td></tr><tr><td>xRate</td><td>exchange rate (4 decimal places multiplied by 10000)</td></tr></table></td></tr>
</table>

It is possible to add multiple payment types as needed to single receipt. Payment amounts should sum up to the amount due. If the total payment amounts is greater that amount due, the change will be calculated by the printer. If it is less and no cash payment was added, the printer will automatically add cash payment for outstanding amount.

If currency payment is added, the payment parameter of printRecTotal should be value in currency. Negative payment amount for currency will register change in currency (change is by default given in PLN).
For other types of payment, negtive amount revokes the payment of the given type.

PrintRecTotal does not close the receipt. To complete receipt, call endFiscalReceipt.

See Receipt Examples for usage examples.</td></tr>
<tr><td>printRecVoid</td><td>√</td><td></td></tr>
<tr><td>printRecVoidItem</td><td>√</td><td></td></tr>
<tr><td>beginFiscalDocument</td><td></td><td></td></tr>
</table>

endFiscalDocument		
printFiscalDocumentLine		
beginItemList		
endItemList		
verifyItem	√	
printPeriodicTotalsReport		
printPowerLossReport		
printReport	√	
printXReport	√	
printZReport	√	
printNormal	√	
beginInsertion		
beginRemoval		
endInsertion		
endRemoval		
beginFixedOutput	√	
beginTraining		
endFixedOutput	√	
endTraining		
printFixedOutput	√	
getData	√	Supported data: FPTR_GD_FIRMWARE, FPTR_GD_PRINTER_ID, FPTR_GD_RECEIPT_NUMBER, FPTR_GD_DAILY_TOTAL, FPTR_GD_CURRENT_TOTAL, FPTR_GD_DAILY_TOTAL, FPTR_GD_TENDER (FPTR_PDL_CASH, FPTR_PDL_CHEQUE, FPTR_PDL_VOUCHER, FPTR_PDL_PAY_CARD), FPTR_GD_DESCRIPTION_LENGTH (FPTR_DL_ITEM, FPTR_DL_ITEM_ADJUSTMENT, FPTR_DL_TOTAL)
getDate	√	
getTotalizer	√	FPTR_GT_GROSS totalizer type supported for receipt and day
getVatEntry	√	
clearError		
resetPrinter	√	

4.4 Examples

4.4.1 Print receipt example

The code below prints a standard fiscal receipt with two payment types (cash and card), with package return and sale and a global 10% discount to items with A tax rate.

```
package javapos.client.examples;

import jpos.FiscalPrinter;
import jpos.FiscalPrinterConst;
import jpos.JposException;
```

```
import pl.novitus.javapos.Consts;

/**
 * Print receipt example
 */
public class PrintReceiptExample {

    public static void main(String[] args) {

        try {
            // create fiscal printer object
            FiscalPrinter printer = new FiscalPrinter();

            // connect to physical device according to logical device configuration in jpos.xml
            printer.open("HDe-tcp");

            // claim the printer
            printer.claim(2000);

            // from now on, the printer is ready to accept commands. Let's print a receipt, therefore:

            // set receipt type: standard receipt
            printer.setFiscalReceiptType(FiscalPrinterConst.FPTR_RT_SALES);
```

receipt sequence

```
            // start fiscal receipt
            printer.beginFiscalReceipt(true);

            printer.printRecItem("Bread", 15000, 1000, 1, 15000, "");
            printer.printRecItem("Butter", 100000, 2000, 1, 50000, "");
            printer.printRecItem("Cheese", 175700, 355, 1, 495000, "kg");
            printer.printRecItem("Water", 20000, 1000, 1, 20000, "");
            printer.printRecItem("#10", 5000, 1000, 1, 5000, "");                // refundable package sale (code 10)
            printer.printRecItemRefund("#20", 7500, 1000, 1, 7500, "");          // refundable package return (code 20)

            // (more receipt commands, like printRecItemAdustment etc. are allowed here, see JavaPOS documentation)

            // for example - print additional messages at the end of receipt
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_CARD);
            printer.printRecMessage("Mastercard");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_CARD_NUMBER);
            printer.printRecMessage("**** **** **** 1234");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_VALID_TO);
            printer.printRecMessage("12.2021");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_POINT_GRANTED);
            printer.printRecMessage("5");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_POINTS_TOTAL);
            printer.printRecMessage("230");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_WITH_DISCOUNT);
            printer.printRecMessage("130.55");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_POINTS_BONUS);
            printer.printRecMessage("16");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_REGISTER_NUMB);
            printer.printRecMessage("WW12345");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_DOT_LINE);
            printer.printRecMessage("");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_EMPTY_LINE);
            printer.printRecMessage("");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_ADVANCE);
            printer.printRecMessage("100.00 zł");
            printer.setMessageType(FiscalPrinterConst.FPTR_MT_FREE_TEXT);
            printer.printRecMessage("To jest dowolny tekst");

            // receipt subtotal. Sum of all item values except packages (15000 + 100000 + 175700 + 20000)
            // is passed here (this call is optional)
            printer.printRecSubtotal(310700);

            // add 10% discount to tax rate group A (index 1)
            printer.printRecPackageAdjustment(FiscalPrinterConst.FPTR_AT_PERCENTAGE_DISCOUNT, "stały klient", "1,10");

            /* the value to be paid is receipt total + sum of packages sold - sum of packages returned. */

            // at least one total payment form is required
            // total is receipt subtotal - rebates (310700 - 31100 = 279600)
            // sum of payments should be total + packages sold - packages acquired

            printer.printRecTotal(279600, 177100, Integer.toString(Consts.FiscalPrinter.PaymentType.CASH));
            printer.printRecTotal(279600, 100000, Integer.toString(Consts.FiscalPrinter.PaymentType.CARD) + ";VISA
```

```
                ****4535");

            // close receipt
            printer.endFiscalReceipt(true);

            // the receipt is now completed. We can start another one (printer.beginFiscalReceipt()), or...
            // send non receipt command or...

            // release printer
            printer.close();

            // the printer is now closed. No further operations are possible unless
            // open()/claim() are called again.

        } catch(JposException e) {
            System.out.println("Operation failed. Error: " + e.getMessage() + ", error code: " + e.getErrorCode());
        }

    }
}
```

4.4.2 Print receipt example (currency payment)

```
package javapos.client.examples;

import jpos.FiscalPrinter;
import jpos.FiscalPrinterConst;
import jpos.JposException;
import pl.novitus.javapos.Consts;

/**
 * Print receipt example
 */
public class PrintReceiptExampleEur {

    public static void main(String[] args) {

        try {
            // create fiscal printer object
            FiscalPrinter printer = new FiscalPrinter();

            // connect to physical device according to logical device configuration in jpos.xml
            printer.open("HDe-tcp");

            // claim the printer
            printer.claim(2000);

            // from now on, the printer is ready to accept commands. Let's print a receipt, therefore:

            // set receipt type: standard receipt
            printer.setFiscalReceiptType(FiscalPrinterConst.FPTR_RT_SALES);

            // start fiscal receipt
            printer.beginFiscalReceipt(true);

            printer.printRecItem("Bread", 15000, 1000, 1, 15000, "");
            printer.printRecItem("Butter", 100000, 2000, 1, 50000, "");
            printer.printRecItem("Cheese", 175700, 355, 1, 495000, "kg");
            printer.printRecItem("Water", 20000, 1000, 1, 20000, "");
            printer.printRecItem("#10", 5000, 1000, 1, 5000, "");                 // refundable package sale (code 10)
            printer.printRecItemRefund("#20", 7500, 1000, 1, 7500, "");           // refundable package return (code 20)

            // (more receipt commands, like printRecItemAdustment etc. are allowed here, see JavaPOS documentation)

            // receipt subtotal. Sum of all item values except packages (15000 + 100000 + 175700 + 20000)
            // is passed here (this call is optional)
            printer.printRecSubtotal(310700);

            // add 10% discount to tax rate group A (index 1)
            printer.printRecPackageAdjustment(FiscalPrinterConst.FPTR_AT_PERCENTAGE_DISCOUNT, "stały klient", "1,10");

            /* the value to be paid is receipt total + sum of packages sold - sum of packages returned. */

            // add payment in EUR
```

```
            // exchange rate is 3.99PLN/1EUR. 10 EUR was tendered, which equals 39.90 PLN
            printer.printRecTotal(279600, 100000, Integer.toString(Consts.FiscalPrinter.PaymentType.CURRENCY) +
                  ";EUR;39900");

            // Register change in EUR. Payment due is 27.71 (27.96 + 0.50 — 0.75) which equals 6,95 EUR.
            // The change is 3.05 EUR.
            // If this is omitted, change will be calculated in PLN
            printer.printRecTotal(279600, -30500, Integer.toString(Consts.FiscalPrinter.PaymentType.CURRENCY) +
                  ";EUR;39900");

            // close receipt
            printer.endFiscalReceipt(true);

            // the receipt is now completed. We can start another one (printer.beginFiscalReceipt()), or...
            // send non receipt command or...

            // release printer
            printer.close();

            // the printer is now closed. No further operations are possible unless
            // open()/claim() are called again.

        } catch(JposException e) {
            System.out.println("Operation failed. Error: " + e.getMessage() + ", error code: " + e.getErrorCode());
        }

    }
}
```

4.4.3 Print invoice example

The code below prints a standard invoice with three items.

```
package javapos.client.examples;

import jpos.FiscalPrinter;
import jpos.FiscalPrinterConst;
import jpos.JposException;
import pl.novitus.javapos.Consts;

package javapos.client.examples;

import jpos.FiscalPrinter;
import jpos.FiscalPrinterConst;
import jpos.JposException;
import pl.novitus.javapos.Consts;

/**
 * invoice example
 */
public class PrintInvoiceExample {

    public static void main(String[] args) {

        try {
            // create fiscal printer object
            FiscalPrinter printer = new FiscalPrinter();

            // connect to physical device according to logical device configuration in jpos.xml
            printer.open("HDe-tcp");

            // claim the printer
            printer.claim(2000);

            // from now on, the printer is ready to accept commands. Let's print a receipt, therefore:

            // set receipt type: standard invoice
            printer.setFiscalReceiptType(FiscalPrinterConst.FPTR_RT_SIMPLE_INVOICE);

            // set invoice header data (invoice number|tax id|buyer's name|other header data|…)
            // * required
            printer.setAdditionalHeader("FV12345678|123-456-78-90|MiniMax sp z o.o.|ul. Maślana 145a|00-321 Warszawa|Nr
rej. WWX 52C12||");
```

```
            // start invoice
            printer.beginFiscalReceipt(true);

            printer.printRecItem("Bread", 15000, 1000, 1, 15000, "");
            printer.printRecItem("Butter", 100000, 2000, 1, 50000, "");
            printer.printRecItem("Cheese", 175700, 355, 1, 495000, "kg");

            // subtotal. Sum of all item values (15000 + 100000 + 175700) is passed here
            // (this is optional)
            printer.printRecSubtotal(290700);

            // total
            printer.printRecTotal(290700, 290700, "");

            // close invoice
            printer.endFiscalReceipt(true);

            // the receipt is now completed. We can start another one (printer.beginFiscalReceipt()), or...
            // send non receipt command or...

            System.out.println(printer.getDayOpened());

            // release printer
            printer.close();

            // the printer is now closed. No further operations are possible unless
            // open()/claim() are called again.

        } catch(JposException e) {
            System.out.println("Operation failed. Error: " + e.getMessage() + ", error code: " + e.getErrorCode());
        }
    }
}
```

4.4.4 dayOpened example

```
package javapos.client.examples;

import jpos.FiscalPrinter;
import jpos.FiscalPrinterConst;
import jpos.JposException;

/**
 *
 * Day opened example
 */
public class DayOpenedExample {

    public static void main(String[] args) {
        try {
            // create fiscal printer object
            FiscalPrinter printer = new FiscalPrinter();

            // connect to physical device according to logical device configuration in jpos.xml
            printer.open("HDe-tcp");

            // claim the printer
            printer.claim(2000);

            // check if day has been already opened (a sale was registered since last daily report)
            boolean dayOpened = printer.getDayOpened();
            System.out.println("Day opened: " + (dayOpened ? "yes" : "no"));

            // release printer
            printer.close();

            // the printer is now closed. No further operations are possible unless
            // open()/claim() are called again.

        } catch(JposException e) {
            System.out.println("Operation failed. Error: " + e.getMessage() + ", error code: " + e.getErrorCode());
        }
    }
}
```

4.4.5 getData example

```
package javapos.client.examples;

import jpos.FiscalPrinter;
import jpos.FiscalPrinterConst;
import jpos.JposException;

public class GetDataExample {

    private FiscalPrinter printer = null;

    private void printValue(String str, int valId) throws JposException {
        String [] data = new String[1];
        int[] optArgs = {};
        printer.getData(valId, optArgs, data);
        System.out.println(str + ": " + data[0]);
    }

    public void doExample() {
        try {
            // create fiscal printer object
            printer = new FiscalPrinter();

            // connect to physical device according to logical device configuration in jpos.xml
            printer.open("HDe-tcp");

            // claim the printer
            printer.claim(2000);

            printValue("Firmware (FPTR_GD_FIRMWARE)", FiscalPrinterConst.FPTR_GD_FIRMWARE);
            printValue("Printer ID (FPTR_GD_PRINTER_ID)", FiscalPrinterConst.FPTR_GD_PRINTER_ID);
            printValue("Last receipt number (FPTR_GD_RECEIPT_NUMBER)", FiscalPrinterConst.FPTR_GD_RECEIPT_NUMBER);
            printValue("Last invoice number (FPTR_GD_FISCAL_DOC)", FiscalPrinterConst.FPTR_GD_FISCAL_DOC);
            printValue("Daily total (FPTR_GD_DAILY_TOTAL)", FiscalPrinterConst.FPTR_GD_DAILY_TOTAL);
            printValue("Last fiscal report number (FPTR_GD_Z_REPORT)", FiscalPrinterConst.FPTR_GD_Z_REPORT);

            // release printer
            printer.close();

            // the printer is now closed. No further operations are possible unless
            // open()/claim() are called again.

        } catch(JposException e) {
            System.out.println("Operation failed. Error: " + e.getMessage() + ", error code: " + e.getErrorCode());
        }
    }

    public static void main(String[] args) {
        new GetDataExample().doExample();
    }
}
```

4.4.6 getVatEntry example

```
package javapos.client.examples;

import java.math.BigDecimal;
import jpos.FiscalPrinter;
import jpos.JposException;

public class GetVatEntryExample {

    public static void main(String[] args) {
        try {
            // create fiscal printer object
```

```
            FiscalPrinter printer = new FiscalPrinter();

            // connect to physical device according to logical device configuration in jpos.xml
            printer.open("HDe-tcp");

            // claim the printer
            printer.claim(2000);

            int[] vatRate = new int[1];

            // get using tax rate index
            for(int idx = 1; idx < 4; ++idx) {
                printer.getVatEntry(idx, 0, vatRate);
                BigDecimal vr = new BigDecimal(vatRate[0]);
                vr = vr.scaleByPowerOfTen(-2);
                System.out.println("VAT rate index: " + idx + ", rate: " + vr + "%");
            }

            // get using tax rate ID
            for(char idx = 'A'; idx < 'D'; ++idx) {
                printer.getVatEntry(idx, 0, vatRate);
                BigDecimal vr = new BigDecimal(vatRate[0]);
                vr = vr.scaleByPowerOfTen(-2);
                System.out.println("VAT rate ID: " + idx + ", rate: " + vr + "%");
            }

            // release printer
            printer.close();

            // the printer is now closed. No further operations are possible unless
            // open()/claim() are called again.

        } catch(JposException e) {
            System.out.println("Operation failed. Error: " + e.getMessage() + ", error code: " + e.getErrorCode());
        }
    }
}
```

4.4.7 setDate example

```
package javapos.client.examples;

import java.math.BigDecimal;
import jpos.FiscalPrinter;
import jpos.JposException;


public class SetDateExample {

    public static void main(String[] args) {
        try {
            // create fiscal printer object
            FiscalPrinter printer = new FiscalPrinter();

            // connect to physical device according to logical device configuration in jpos.xml
            printer.open("HDe-tcp");

            // claim the printer
            printer.claim(2000);

            // The date and time is passed as a string in the format "ddmmyyyyhhmm"
            printer.setDate("200520191325");

            // release printer
            printer.close();

            // the printer is now closed. No further operations are possible unless
            // open()/claim() are called again.

        } catch(JposException e) {
            System.out.println("Operation failed. Error: " + e.getMessage() + ", error code: " + e.getErrorCode());
        }
```

```
    }
}
```

4.4.8 setVatTable example

```
package javapos.client.examples;

import jpos.FiscalPrinter;
import jpos.JposException;
import pl.novitus.javapos.Consts;


public class SetVatTableExample {

    public static void main(String[] args) {
        try {
            // create fiscal printer object
            FiscalPrinter printer = new FiscalPrinter();

            // connect to physical device according to logical device configuration in jpos.xml
            printer.open("HDe-tcp");

            // claim the printer
            printer.claim(2000);

            // set some rates
            printer.setVatValue('D', "17.00");                    // 17%
            printer.setVatValue('E', Consts.TaxRate.FREE);        // 'tax free'
            printer.setVatValue('F', Consts.TaxRate.UNUSED);      // set index as 'not used'

            printer.setVatTable();

            // release printer
            printer.close();

            // the printer is now closed. No further operations are possible unless
            // open()/claim() are called again.

        } catch(JposException e) {
            System.out.println("Operation failed. Error: " + e.getMessage() + ", error code: " + e.getErrorCode());
        }
    }
}
```

5. Logging

Driver uses log4j2 logging framework. To enable logging, one can create a valid log4j2 configuration file (properties format, xml format is not supported) and supply path to the configuration file to application that loads the driver as log4j.configurationFile system property.

It can be done by setting Java VM parameter in the way as follows:

```
-Dlog4j.configurationFile=<path_to_config_file>
```

An example configuration to write logs to file:

```
status = INFO
name = PropertiesConfig
filters = threshold
filter.threshold.type = ThresholdFilter
filter.threshold.level = debug
appenders = file
appender.file.type = File
appender.file.name = file
appender.file.fileName = /tmp/log.txt
appender.file.layout.type = PatternLayout
appender.file.layout.pattern = %d [%t] %-5p %c.%M - %m%n
rootLogger.level = debug
rootLogger.appenderRefs = stdout
rootLogger.appenderRef.stdout.ref = file
```

Refer to Apache Log4j2 website for more information regarding log configuration.

https://logging.apache.org/log4j/2.x/manual/configuration.html

6. Versions and changes

version	changes
1.0	first release
1.0a	tcp connection support receipt payment methods support DirectIO support GetData and GetTotalizer methods support bug fixes
1.0b	bug fixes better device state handling better I/O handling timeout handling improvements for long running device tasks (ie. Z-Report) added multilanguage support (only English is supported at this time) documentation corrections
1.0c	bug fixes fixed broken Mazovia encoding set 4 decimal places for all prices/values invoice receipt support upgrades to libraries (jSerialComm upgraded to 2.5.0)
1.0d	performance optimizations minor bug fixes printRecPackageAdjustment support documentation updates – more examples added
1.0e	Support for refundable packages minor bug fixes
1.0h	Bug fixes Currency payment support