```java
/*
 * Copyright (C) 2016 redxef.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301  USA
 */
package powerspy.client;

import com.fazecast.jSerialComm.SerialPort;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;
import java.util.Random;
import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import powerspy.baselib.*;
import static powerspy.baselib.IODefs.*;
import static powerspy.client.Defs.*;

/**
 *
 * @author redxef
 */
public class Frame extends JFrame {

        private static final String FLT_FORMAT = "%.2f";

        private static final ArrayInputStream dummy_is;
        private static final Thread dummy_stream;
        private static final Random r = new Random();

        static {
                dummy_is = new ArrayInputStream();

                dummy_stream = new Thread() {

                        private void insertKey(char key)
                        {
                                byte[] b = new byte[3];
                                b[0] = START_OF_TEXT;
                                b[1] = UINT8;
                                b[2] = (byte) (key & 0xff);

```

```
					dummy_is.insert(b, 0, b.length);
				}

				private void insertValue(int i)
				{
					byte[] b = new byte[5];
					b[0] = START_OF_TEXT;
					b[1] = INT24;
					b[2] = (byte) (i >> 16 & 0xff);
					b[3] = (byte) (i >> 8 & 0xff);
					b[4] = (byte) (i & 0xff);

					dummy_is.insert(b, 0, b.length);
				}

				@Override
				public void run()
				{
					while (true) {
						insertKey(K_CURRENT);
						insertValue(r.nextInt(5000));
						insertKey(K_APPARENTEPOWER);
						insertValue(r.nextInt(1150000));
						insertKey(K_REALPOWER);
						insertValue(r.nextInt(1150000));
						insertKey(K_REACTIVEPOWER);
						insertValue(r.nextInt(1150000));

						try {
							Thread.sleep(1000);
						} catch (InterruptedException ex) {
						}
					}
				}
			};
		}

		private static final float FIXMUL = 0.88f;
		private final Dimension BASIC_SIZE = new Dimension(250, 370);
		private Controller c;
		protected JComboBox<SerialPort> ports;
		private JLabel info;
		private JProgressBar pb;
		private JLabel min;
		private JLabel max;
		private Timer progress_update;
		private int target;
		private JTable t;
		private SerialPort curr_port;

		private int prev_width;
		private long last_time;

		/**
		 * Creates a new Frame for displaying the data sent from PowerSpy.
		 */
		public Frame()
```

```
        {
                super();
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                getContentPane()
                        .setLayout(new BoxLayout(getContentPane(), BoxLayout.Y_AX

                initCombobox();
                initProgressBar();
                initProgressTimer();
                initDesc();
                initJTable();
                initJLabel();
                initResize();

                pack();
                setSize(BASIC_SIZE);
                setPreferredSize(BASIC_SIZE);
                setMinimumSize(getSize());
                //setMaximumSize(getSize());
                setLocationRelativeTo(null);

                curr_port = null;
                prev_width = BASIC_SIZE.width;
                last_time = System.currentTimeMillis();

        }

        /**
         * Links the Controller to this Frame
         *
         * @param c the Controller to link
         */
        public void installController(Controller c)
        {
                this.c = c;
        }

        //<editor-fold defaultstate="collapsed" desc="init combo box">
        private void initCombobox()
        {
                ports = new JComboBox<>();
                ports.setRenderer(new PortListCellRenderer());
                ports.addActionListener((ActionEvent e) -> {
                        connect(e);
                });

                add(ports);
        }
        //</editor-fold>

        //<editor-fold defaultstate="collapsed" desc="init jlabel">
        private void initJLabel()
        {
                JPanel p = new JPanel(new FlowLayout(FlowLayout.LEFT));
                info = new JLabel(" ");
                p.add(info);
                add(p);
```

```java
	}
	//</editor-fold>

	//<editor-fold defaultstate="collapsed" desc="init progress bar">
	private void initProgressBar()
	{
		JPanel p = new JPanel();
		p.setLayout(new BorderLayout());
		pb = new JProgressBar() {

			@Override
			public void setValue(int n)
			{
				super.setValue(n);

				if (t != null)
					if (t.getSelectedRow() == -1)
						setString("-");
					else
						setString(t.getValueAt(
							t.getSelectedRow(), 1)
							.toString());
			}
		};
		pb.setUI(ProgressCircleUI.getPSDesign());
		pb.setForeground(PS_ORANGE);

		pb.setValue(0);
		pb.setStringPainted(true);

		p.add(pb);
		add(p);
	}
	//</editor-fold>

	//<editor-fold defaultstate="collapsed" desc="progress bar timer">
	private void initProgressTimer()
	{
		//some fancy animation
		progress_update = new Timer(5, (ActionEvent e) -> {
			int diff = pb.getValue() - target;
			diff = (diff > 0) ? diff : -diff;
			diff = (int) (1 / Math.log(diff / 15.0));
			if (diff <= 0)
				diff = 1;

			if (pb.getValue() > target)
				pb.setValue(pb.getValue() - diff);
			else if (pb.getValue() < target)
				pb.setValue(pb.getValue() + diff);
			else
				progress_update.stop();
		});
		progress_update.start();
	}
	//</editor-fold>

	//<editor-fold defaultstate="collapsed" desc="init desc">
	private void initDesc()
	{
		JPanel p = new JPanel(new GridLayout(1, 2));
		min = new JLabel(" ", SwingConstants.CENTER);
		max = new JLabel(" ", SwingConstants.CENTER);

		p.add(min);
		p.add(max);
		add(p);
	}
	//</editor-fold>

	//<editor-fold defaultstate="collapsed" desc="init table">
	private void initJTable()
	{
		t = new JTable(7, 2) {

			@Override
			public boolean isCellEditable(int row, int column)
			{
				return false;
			}
		};
		t.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
		t.getSelectionModel().addListSelectionListener((ListSelectionEver
			updateVals();
		});
		t.setValueAt("Current", 0, 0);
		t.setValueAt("Real Power", 1, 0);
		t.setValueAt("Apparent Power", 2, 0);
		t.setValueAt("Reactive Power", 3, 0);
		t.setValueAt("Raw Current", 4, 0);
		t.setValueAt("Offset", 5, 0);
		t.setValueAt("Raw Supply", 6, 0);
		add(t);
	}
	//</editor-fold>

	//<editor-fold defaultstate="collapsed" desc="resize handler">
	private void initResize()
	{
		addComponentListener(new ComponentListener() {
			@Override
			public void componentResized(ComponentEvent e)
			{
				if (System.currentTimeMillis() - last_time < 10)
					return;
				int xges = getWidth();
				int yges = getHeight();

				int xpb = pb.getWidth();
				int ypb = pb.getHeight();

				int xdif = xges - xpb;
				int ydif = yges - ypb;

```

```
                                int xcord = getX();
                                int ycord = getY();

                                if (prev_width + 200 > xges) {
                                        if (ypb > xpb) {
                                                ypb = (int) (xpb * FIXMUL);
                                        } else {
                                                xpb = (int) (ypb / FIXMUL);
                                        }
                                } else if (prev_width - 200 < xges) {
                                        if (ypb < xpb) {
                                                ypb = (int) (xpb * FIXMUL);
                                        } else {
                                                xpb = (int) (ypb / FIXMUL);
                                        }
                                }

                                e.getComponent().setBounds(xcord, ycord, xdif + x
                                prev_width = getWidth();
                                last_time = System.currentTimeMillis();
                        }

                        @Override
                        public void componentMoved(ComponentEvent e)
                        {
                        }

                        @Override
                        public void componentShown(ComponentEvent e)
                        {
                        }

                        @Override
                        public void componentHidden(ComponentEvent e)
                        {
                        }
                });
        }
        //</editor-fold>

        private void connect(ActionEvent e)
        {

                new Thread() {

                        @Override
                        public void run()
                        {
                                SerialPort sp = (SerialPort) ports.getSelectedIte

                                if (sp == null)
                                        return;
                                if (sp == curr_port)
                                        return;

                                curr_port = sp;
                                info.setText("connecting...");
                                curr_port.openPort();

                                if (curr_port.isOpen()) {
                                        info.setText("connected");
                                } else {
                                        info.setText("failed to connect; setting
                                        c.terminate();
                                        c.setPSInputStream(new PSInputStream(dumm
                                        dummy_stream.start();
                                        c.start();
                                        return;
                                }

                                sp.writeBytes(new byte[]{0}, 1);
                                if (c != null) {
                                        c.terminate();
                                        c.setPSInputStream(new PSInputStream(sp.g
                                        c.start();
                                }
                        }
                }.start();
        }

        //<editor-fold defaultstate="collapsed" desc="helper methods for progress
        private void setMin(String s)
        {
                min.setText("<html><div style='text-align: center;'>" + s + "</ht
        }

        private void setMax(String s)
        {
                max.setText("<html><div style='text-align: center;'>" + s + "</ht
        }

        private void updateVals()
        {
                Object val_;
                float val;
                if (t.getSelectedRow() == -1) {
                        target = 70;
                } else if (t.getSelectedRow() == 0) {//current
                        val_ = t.getValueAt(0, 1);
                        if (val_ == null)
                                return;
                        val = Float.parseFloat((String) val_);
                        target = (int) ((val * 100) / (MAX_AMPS - MIN_AMPS));
                        setMin(Integer.toString(MIN_AMPS));
                        setMax(Integer.toString(MAX_AMPS));
                } else {
                        val_ = t.getValueAt(t.getSelectedRow(), 1);
                        if (val_ == null)
                                return;
                        val = Float.parseFloat((String) val_);
                        target = (int) ((val * 100) / (MAX_POWER - MIN_POWER));
                        setMin(Integer.toString(MIN_POWER));
                        setMax(Integer.toString(MAX_POWER));
                }
                progress_update.start();
        }
        //</editor-fold>

        /**
         * Sets the current in the Table
         *
         * @param d the value to set
         */
        public void setCurrent(float d)
        {
                t.setValueAt(String.format(FLT_FORMAT, d), 0, 1);
                updateVals();
        }

        /**
         * Sets the real power in the Table
         *
         * @param d the value to set
         */
        public void setRealPower(float d)
        {
                t.setValueAt(String.format(FLT_FORMAT, d), 1, 1);
                updateVals();
        }

        /**
         * Sets the apparent power in the Table
         *
         * @param d the value to set
         */
        public void setApparentPower(float d)
        {
                t.setValueAt(String.format(FLT_FORMAT, d), 2, 1);
                updateVals();
        }

        /**
         * Sets the reactive power in the Table
         *
         * @param d the value to set
         */
        public void setReactivePower(float d)
        {
                t.setValueAt(String.format(FLT_FORMAT, d), 3, 1);
                updateVals();
        }

        /**
         * Sets the raw current value in the Table
         *
         * @param d the value to set
         */
        public void setRawCurrent(float d)
        {
                t.setValueAt(String.format(FLT_FORMAT, d), 4, 1);
                updateVals();
```

```
	}

	/**
	 * Sets the offset of the voltage in the Table
	 *
	 * @param d the value to set
	 */
	public void setOffs(float d)
	{
		t.setValueAt(String.format(FLT_FORMAT, d), 5, 1);
		updateVals();
	}

	/**
	 * Sets the raw voltage value in the Table
	 *
	 * @param d the value to set
	 */
	public void setRawVoltage(float d)
	{
		t.setValueAt(String.format(FLT_FORMAT, d), 6, 1);
		updateVals();
	}
}
```