

```

1 /*
2  * Copyright (C) 2016 redxef.
3  *
4  * This library is free software; you can redistribute it and/or
5  * modify it under the terms of the GNU Lesser General Public
6  * License as published by the Free Software Foundation; either
7  * version 2.1 of the License, or (at your option) any later version.
8  *
9  * This library is distributed in the hope that it will be useful,
10 * but WITHOUT ANY WARRANTY; without even the implied warranty of
11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 * Lesser General Public License for more details.
13 *
14 * You should have received a copy of the GNU Lesser General Public
15 * License along with this library; if not, write to the Free Software
16 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
17 * MA 02110-1301 USA
18 */
19 package powerspy.client;
20
21 import com.fazecast.jSerialComm.SerialPort;
22 import java.awt.*;
23 import java.awt.event.ActionEvent;
24 import java.awt.event.ComponentEvent;
25 import java.awt.event.ComponentListener;
26 import java.util.Random;
27 import javax.swing.*;
28 import javax.swing.event.ListSelectionEvent;
29 import powerspy.baselib.*;
30 import static powerspy.baselib.IODefs.*;
31 import static powerspy.client.Defs.*;
32
33 /**
34  *
35  * @author redxef
36  */
37 public class Frame extends JFrame {
38
39     private static final String FLT_FORMAT = "%.2f";
40
41     private static final ArrayInputStream dummy_is;
42     private static final Thread dummy_stream;
43     private static final Random r = new Random();
44
45     static {
46         dummy_is = new ArrayInputStream();
47
48         dummy_stream = new Thread() {
49
50             private void insertKey(char key)
51             {
52                 byte[] b = new byte[3];
53                 b[0] = START_OF_TEXT;
54                 b[1] = UINT8;
55                 b[2] = (byte) (key & 0xff);
56

```

```

57         dummy_is.insert(b, 0, b.length);
58     }
59
60     private void insertValue(int i)
61     {
62         byte[] b = new byte[5];
63         b[0] = START_OF_TEXT;
64         b[1] = INT24;
65         b[2] = (byte) (i >> 16 & 0xff);
66         b[3] = (byte) (i >> 8 & 0xff);
67         b[4] = (byte) (i & 0xff);
68
69         dummy_is.insert(b, 0, b.length);
70     }
71
72     @Override
73     public void run()
74     {
75         while (true) {
76             insertKey(K_CURRENT);
77             insertValue(r.nextInt(5000));
78             insertKey(K_APPARENTEPOWER);
79             insertValue(r.nextInt(1150000));
80             insertKey(K_REALPOWER);
81             insertValue(r.nextInt(1150000));
82             insertKey(K_REACTIVEPOWER);
83             insertValue(r.nextInt(1150000));
84
85             try {
86                 Thread.sleep(1000);
87             } catch (InterruptedException ex) {
88             }
89         }
90     }
91     };
92 }
93
94 private static final float FIXMUL = 0.88f;
95 private final Dimension BASIC_SIZE = new Dimension(250, 370);
96 private Controller c;
97 protected JComboBox<SerialPort> ports;
98 private JLabel info;
99 private JProgressBar pb;
100 private JLabel min;
101 private JLabel max;
102 private Timer progress_update;
103 private int target;
104 private JTable t;
105 private SerialPort curr_port;
106
107 private int prev_width;
108 private long last_time;
109
110 /**
111  * Creates a new Frame for displaying the data sent from PowerSpy.
112  */
113 public Frame()

```

```

114     {
115         super();
116         setDefaultCloseOperation(EXIT_ON_CLOSE);
117         getContentPane()
118             .setLayout(new BoxLayout(getContentPane(), BoxLayout.Y_AXIS));
119
120         initComponents();
121         initProgressBar();
122         initProgressTimer();
123         initDesc();
124         initJTable();
125         initJLabel();
126         initResize();
127
128         pack();
129         setSize(BASIC_SIZE);
130         setPreferredSize(BASIC_SIZE);
131         setMinimumSize(getSize());
132         //setMaximumSize(getSize());
133         setLocationRelativeTo(null);
134
135         curr_port = null;
136         prev_width = BASIC_SIZE.width;
137         last_time = System.currentTimeMillis();
138
139     }
140
141     /**
142     * Links the Controller to this Frame
143     *
144     * @param c the Controller to link
145     */
146     public void installController(Controller c)
147     {
148         this.c = c;
149     }
150
151     //<editor-fold defaultstate="collapsed" desc="init combo box">
152     private void initComponents()
153     {
154         ports = new JComboBox<>();
155         ports.setRenderer(new PortListCellRenderer());
156         ports.addActionListener((ActionEvent e) -> {
157             connect(e);
158         });
159
160         add(ports);
161     }
162     //</editor-fold>
163
164     //<editor-fold defaultstate="collapsed" desc="init jlabel">
165     private void initJLabel()
166     {
167         JPanel p = new JPanel(new FlowLayout(FlowLayout.LEFT));
168         info = new JLabel(" ");
169         p.add(info);
170         add(p);

```

```

171     }
172     //</editor-fold>
173
174     //<editor-fold defaultstate="collapsed" desc="init progress bar">
175     private void initProgressBar()
176     {
177         JPanel p = new JPanel();
178         p.setLayout(new BorderLayout());
179         pb = new JProgressBar() {
180
181             @Override
182             public void setValue(int n)
183             {
184                 super.setValue(n);
185
186                 if (t != null)
187                     if (t.getSelectedRow() == -1)
188                         setString("-");
189                     else
190                         setString(t.getValueAt(
191                             t.getSelectedRow(), 1)
192                             .toString());
193             }
194         };
195         pb.setUI(ProgressCircleUI.getPSDesign());
196         pb.setForeground(PS_ORANGE);
197
198         pb.setValue(0);
199         pb.setStringPainted(true);
200
201         p.add(pb);
202         add(p);
203     }
204     //</editor-fold>
205
206     //<editor-fold defaultstate="collapsed" desc="progress bar timer">
207     private void initProgressTimer()
208     {
209         //some fancy animation
210         progress_update = new Timer(5, (ActionEvent e) -> {
211             int diff = pb.getValue() - target;
212             diff = (diff > 0) ? diff : -diff;
213             diff = (int) (1 / Math.log(diff / 15.0));
214             if (diff <= 0)
215                 diff = 1;
216
217             if (pb.getValue() > target)
218                 pb.setValue(pb.getValue() - diff);
219             else if (pb.getValue() < target)
220                 pb.setValue(pb.getValue() + diff);
221             else
222                 progress_update.stop();
223         });
224         progress_update.start();
225     }
226     //</editor-fold>
227

```

```

228 //<editor-fold defaultstate="collapsed" desc="init desc">
229 private void initDesc()
230 {
231     JPanel p = new JPanel(new GridLayout(1, 2));
232     min = new JLabel(" ", SwingConstants.CENTER);
233     max = new JLabel(" ", SwingConstants.CENTER);
234
235     p.add(min);
236     p.add(max);
237     add(p);
238 }
239 //</editor-fold>
240
241 //<editor-fold defaultstate="collapsed" desc="init table">
242 private void initJTable()
243 {
244     t = new JTable(7, 2) {
245
246         @Override
247         public boolean isCellEditable(int row, int column)
248         {
249             return false;
250         }
251     };
252     t.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
253     t.getSelectionModel().addListSelectionListener((ListSelectionEvent e)
254         {
255             updateVals();
256         });
257     t.setValueAt("Current", 0, 0);
258     t.setValueAt("Real Power", 1, 0);
259     t.setValueAt("Apparent Power", 2, 0);
260     t.setValueAt("Reactive Power", 3, 0);
261     t.setValueAt("Raw Current", 4, 0);
262     t.setValueAt("Offset", 5, 0);
263     t.setValueAt("Raw Supply", 6, 0);
264     add(t);
265 }
266 //</editor-fold>
267
268 //<editor-fold defaultstate="collapsed" desc="resize handler">
269 private void initResize()
270 {
271     addComponentListener(new ComponentListener() {
272
273         @Override
274         public void componentResized(ComponentEvent e)
275         {
276             if (System.currentTimeMillis() - last_time < 10)
277                 return;
278             int xges = getWidth();
279             int yges = getHeight();
280
281             int xpb = pb.getWidth();
282             int ypb = pb.getHeight();
283
284             int xdif = xges - xpb;
285             int ydif = yges - ypb;

```

```

285         int xcord = getX();
286         int ycord = getY();
287
288         if (prev_width + 200 > xges) {
289             if (ypb > xpb) {
290                 ypb = (int) (xpb * FIXMUL);
291             } else {
292                 xpb = (int) (ypb / FIXMUL);
293             }
294         } else if (prev_width - 200 < xges) {
295             if (ypb < xpb) {
296                 ypb = (int) (xpb * FIXMUL);
297             } else {
298                 xpb = (int) (ypb / FIXMUL);
299             }
300         }
301
302         e.getComponent().setBounds(xcord, ycord, xdif + x, ydif + y);
303         prev_width = getWidth();
304         last_time = System.currentTimeMillis();
305     }
306
307     @Override
308     public void componentMoved(ComponentEvent e)
309     {
310     }
311
312     @Override
313     public void componentShown(ComponentEvent e)
314     {
315     }
316
317     @Override
318     public void componentHidden(ComponentEvent e)
319     {
320     }
321     });
322 }
323 //</editor-fold>
324
325 private void connect(ActionEvent e)
326 {
327     new Thread() {
328
329         @Override
330         public void run()
331         {
332             SerialPort sp = (SerialPort) ports.getSelectedItem();
333
334             if (sp == null)
335                 return;
336             if (sp == curr_port)
337                 return;
338
339             curr_port = sp;
340             info.setText("connecting...");
341

```

```

342         curr_port.openPort();
343
344         if (curr_port.isOpen()) {
345             info.setText("connected");
346         } else {
347             info.setText("failed to connect; setting
348             c.terminate();
349             c.setPSInputStream(new PSInputStream(dummy_stream.start());
350             dummy_stream.start();
351             c.start();
352             return;
353         }
354
355         sp.writeBytes(new byte[] {0}, 1);
356         if (c != null) {
357             c.terminate();
358             c.setPSInputStream(new PSInputStream(sp.g
359             c.start();
360         }
361     }
362     }.start();
363 }
364
365 //<editor-fold defaultstate="collapsed" desc="helper methods for progress
366 private void setMin(String s)
367 {
368     min.setText("<html><div style='text-align: center;'>" + s + "</ht
369 }
370
371 private void setMax(String s)
372 {
373     max.setText("<html><div style='text-align: center;'>" + s + "</ht
374 }
375
376 private void updateVals()
377 {
378     Object val_;
379     float val;
380     if (t.getSelectedRow() == -1) {
381         target = 70;
382     } else if (t.getSelectedRow() == 0) { //current
383         val_ = t.getValueAt(0, 1);
384         if (val_ == null)
385             return;
386         val = Float.parseFloat((String) val_);
387         target = (int) ((val * 100) / (MAX_AMPS - MIN_AMPS));
388         setMin(Integer.toString(MIN_AMPS));
389         setMax(Integer.toString(MAX_AMPS));
390     } else {
391         val_ = t.getValueAt(t.getSelectedRow(), 1);
392         if (val_ == null)
393             return;
394         val = Float.parseFloat((String) val_);
395         target = (int) ((val * 100) / (MAX_POWER - MIN_POWER));
396         setMin(Integer.toString(MIN_POWER));
397         setMax(Integer.toString(MAX_POWER));
398     }

```

```

399         progress_update.start();
400     }
401 //</editor-fold>
402
403 /**
404  * Sets the current in the Table
405  *
406  * @param d the value to set
407  */
408 public void setCurrent(float d)
409 {
410     t.setValueAt(String.format(FLT_FORMAT, d), 0, 1);
411     updateVals();
412 }
413
414 /**
415  * Sets the real power in the Table
416  *
417  * @param d the value to set
418  */
419 public void setRealPower(float d)
420 {
421     t.setValueAt(String.format(FLT_FORMAT, d), 1, 1);
422     updateVals();
423 }
424
425 /**
426  * Sets the apparent power in the Table
427  *
428  * @param d the value to set
429  */
430 public void setApparentPower(float d)
431 {
432     t.setValueAt(String.format(FLT_FORMAT, d), 2, 1);
433     updateVals();
434 }
435
436 /**
437  * Sets the reactive power in the Table
438  *
439  * @param d the value to set
440  */
441 public void setReactivePower(float d)
442 {
443     t.setValueAt(String.format(FLT_FORMAT, d), 3, 1);
444     updateVals();
445 }
446
447 /**
448  * Sets the raw current value in the Table
449  *
450  * @param d the value to set
451  */
452 public void setRawCurrent(float d)
453 {
454     t.setValueAt(String.format(FLT_FORMAT, d), 4, 1);
455     updateVals();

```



```
456     }
457
458     /**
459     * Sets the offset of the voltage in the Table
460     *
461     * @param d the value to set
462     */
463     public void setOffs(float d)
464     {
465         t.setValueAt(String.format(FLT_FORMAT, d), 5, 1);
466         updateVals();
467     }
468
469     /**
470     * Sets the raw voltage value in the Table
471     *
472     * @param d the value to set
473     */
474     public void setRawVoltage(float d)
475     {
476         t.setValueAt(String.format(FLT_FORMAT, d), 6, 1);
477         updateVals();
478     }
479 }
```