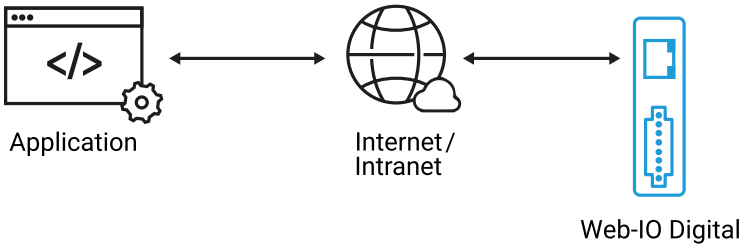
Aplicación al Web-IO digital:

Dirigir y supervisar Web-IO digital con Java

Resumen de productos
Sinopsis de aplicaciones

Java es un idioma de programación muy difundido, fácil de aprender e independiente de una plataforma. En la versión estándar incluye ya todas las clases y métodos, que se necesitan para la programación de aplicaciones TCP/IP. Esto caracteriza a Java como una apreciada herramienta para crear aplicaciones, que comunican con el Web-IO digital.

Además de un ambiente actual Java no se necesitan otros recursos para ejecutarla.



El siguiente ejemplo de programa permite supervisar y dirigir un Web-IO 2x Digital y pone a disposición las siguientes funciones:

- Conectar los Outputs
- leer manual o automáticamente Inputs, Outputs y contadores
- Leer y borrar contadores individuales o todos juntos

¿No tiene todavía un Web-IO y quiere probar el ejemplo presentado?
No hay problema: Le ponemos a disposición el Web-IO Digital 2xInput, 2xOutput PoE gratis durante 30 días. Rellene sencillamente un pedido muestra y le enviaremos el Web-IO para probar a cuenta abierta. Si nos devuelve el aparato dentro de los 30 días, le abonamos la factura completa.

Al pedido muestra

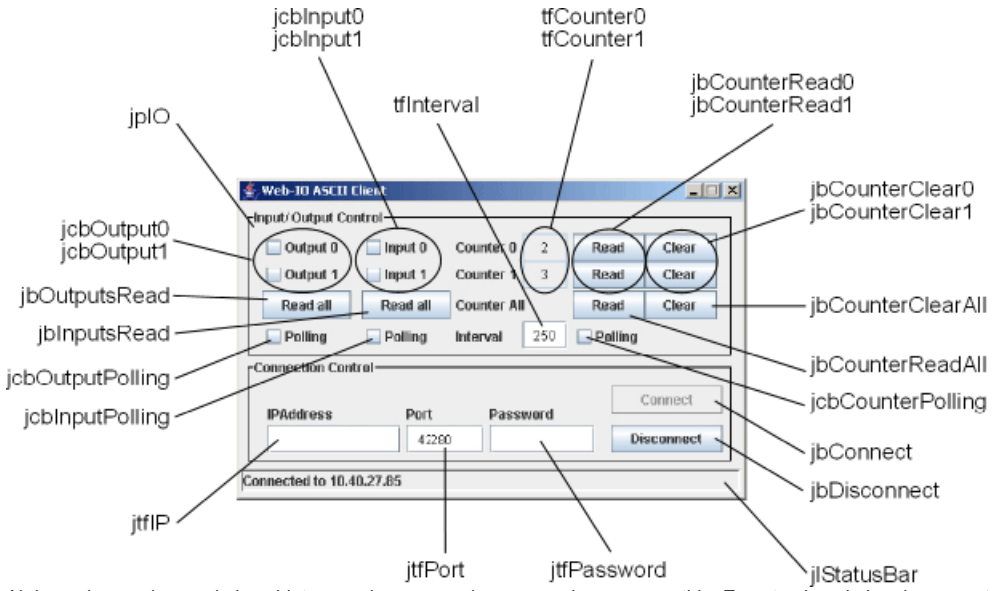
Preparativos

Ya ha abastecido su Web-IO Digital

- con corriente,
- entradas y salidas conectadas
- conectado a su red
- dotado con una dirección IP - conWuTility no hay problemas.

Resumen de los diferentes elementos de mando y de indicación

En la siguiente figura los diferentes elementos de mando y de indicación han sido titulados con el nombre usado en el código de programa. La asignación deber servir como referencia y facilitar así el determinar el ejemplo siguiente.



Al denominar cada uno de los objetos, es de gran ayuda usar nombres con sentido. En este ejemplo la primera parte del nombre describe la clase de objeto y la segunda la función.

1. Importar las fuentes necesitadas

Al comienzo de cada programa tienen que importarse todas las fuentes necesitadas.

```
import java.awt.Color;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
    import java.awt.event.KeyEvent;
    import java.awt.event.KeyListener;
    import java.io.IOException;
    import java.io.InputStream;
    import java.io.OutputStream;
    import java.net.Socket;
    import java.net.UnknownHostException;
    import java.util.StringTokenizer;

    import javax.swing.BorderFactory;
    import javax.swing.JButton;
    import javax.swing.JCheckBox;
    import javax.swing.JComponent;
    import javax.swing.JFrame;
    import javax.swing.JLabel;
    import javax.swing.JPanel;
    import javax.swing.JTextField;
    import javax.swing.Timer;
    import javax.swing.border.BevelBorder;
```

2. Ampliación de la clase

La clase "Cliente" recibe todas las propiedades de la clase "JFrame" para representar la ventana de programa e incluye las interfaces "ActionListener" (Reconocimiento de sucesos de botón), "KeyListener" (Reconocimiento de cambios en campos de texto) y "Runnable" (permiten Threads) mediante la importación de sus métodos.

```
public class Client extends JFrame implements ActionListener, KeyListener, Runnable {
        ...
        }
```

3. Instanciamiento de los elementos gráficos

Para los elementos gráficos (Buttons, CheckBoxes, Label, Panel, TextFields) se crean instancias globales para que estén disponibles en cada método del programa

```
private JButton jbConnect = new JButton("Connect");
    private JButton jbDisconnect = new JButton("Disconnect");
    private JButton jbOutputsRead = new JButton("Read all");
    private JButton jbInputsRead = new JButton("Read all");
    private JButton jbCounterRead0 = new JButton("Read");
    private JButton jbCounterRead1 = new JButton("Read");
    private JButton jbCounterReadAll = new JButton("Read");
    private JButton jbCounterClear0 = new JButton("Clear");
    private JButton jbCounterClear1 = new JButton("Clear");
    private JButton jbCounterClearAll = new JButton("Clear");
    private JCheckBox jcbCounterPolling = new JCheckBox("Polling");
    private JCheckBox jcbOutput0 = new JCheckBox("Output 0");
    private JCheckBox jcbOutput1 = new JCheckBox("Output 1");
    private JCheckBox jcbOutputPolling = new JCheckBox("Polling");
    private JCheckBox jcbInput0 = new JCheckBox("Input 0");
    private JCheckBox jcbInput1 = new JCheckBox("Input 1");
    private JCheckBox jcbInputPolling = new JCheckBox("Polling");
    private JLabel jlStatusBar = new JLabel("No connection");
    private JPanel jpIO = new JPanel();
    private JTextField jtfInterval = new JTextField("250");
    private JTextField jtfCounter0 = new JTextField("0");
    private JTextField jtfCounter1 = new JTextField("0");
    private JTextField jtfIP = new JTextField();
    private JTextField jtfPort = new JTextField("80");
    private JTextField jtfPassword = new JTextField();
```

4. Elementos para la transmisión de datos y el comportamiento temporal

Para la transmisión de datos a través de una interfaz de zócalo se utilizan en este ejemplo instancias de las clases "InputStream", "OutputStream" y "Socket".

```
       private InputStream isInStream;
       private OutputStream osOutStream;
       private Socket soClientSocket;
```

El Timer, que dirige el Polling, se deriva de la clase "Timer" (javax.swing.Timer).

```
private Timer tiPoll;
```

5. El método "main"

El método "main" se llama automáticamente al arrancar el programa. Llama al Constructor de la clase.

```
  public static void main(String[] args) {
       new Client();
       }
```

6. El Constructor

En el Constructor se definen tamaño, posición y comportamiento de los elementos de indicación y la venta de programa. El Constructor se llama sin entrega de parámetros.

```
  public Client() {
        ...
       }
```

A continuación se crea el JPanel "jpIO". Para ello se dimensiona el elemento de indicación, o se desactiva, para crear el estado de partida, y se equipa con un Listener, para que pueda reaccionar a entradas del usuario. Por último se añaden los elementos al JPanel.

```
jcbOutput0.setBounds(15, 25, 80, 20);
    jcbOutput0.setEnabled(false);
    jcbOutput0.addActionListener(this);
    jcbOutput1.setBounds(15, 50, 80, 20);
    jcbOutput1.setEnabled(false);
    jcbOutput1.addActionListener(this);
    jbOutputsRead.setBounds(15, 75, 80, 25);
    jbOutputsRead.setEnabled(false);
    jbOutputsRead.addActionListener(this);
    jcbOutputPolling.setBounds(15, 105, 80, 20);
    jcbOutputPolling.setEnabled(false);
    jcbInput0.setBounds(105, 25, 80, 20);
    jcbInput0.setEnabled(false);
    jcbInput0.addActionListener(this);
    jcbInput1.setBounds(105, 50, 80, 20);
    jcbInput1.setEnabled(false);
    jcbInput1.addActionListener(this);
    jbInputsRead.setBounds(105, 75, 80, 25);
    jbInputsRead.setEnabled(false);
    jbInputsRead.addActionListener(this);
    jcbInputPolling.setBounds(105, 105, 80, 20);
    jcbInputPolling.setEnabled(false);
    JLabel jlCounter0 = new JLabel("Counter 0");
    jlCounter0.setBounds(190, 25, 55, 20);
    jlCounter0.setEnabled(false);
    JLabel jlCounter1 = new JLabel("Counter 1");
    jlCounter1.setBounds(190, 50, 55, 20);
    jlCounter1.setEnabled(false);
    JLabel jlCounterAll = new JLabel("Counter All");
    jlCounterAll.setBounds(190, 77, 70, 20);
    jlCounterAll.setEnabled(false);
    JLabel jlInterval = new JLabel("Interval");
    jlInterval.setBounds(190, 105, 50, 20);
    jlInterval.setEnabled(false);
    tfCounter0.setBounds(250, 23, 40, 25);
    jtfCounter0.setEnabled(false);
    jtfCounter0.setHorizontalAlignment(JTextField.CENTER);
    jtfCounter0.setEditable(false);
    jtfCounter1.setBounds(250, 48, 40, 25);
    jtfCounter1.setEnabled(false);
    jtfCounter1.setHorizontalAlignment(JTextField.CENTER);
    jtfCounter1.setEditable(false);
    jtfInterval.setBounds(250, 102, 40, 25);
    jtfInterval.setEnabled(false);
    jtfInterval.setHorizontalAlignment(JTextField.CENTER);
    jtfInterval.addKeyListener(this);
    jbCounterRead0.setBounds(295, 23, 65, 25);
    jbCounterRead0.setEnabled(false);
    jbCounterRead0.addActionListener(this);
    jbCounterRead1.setBounds(295, 48, 65, 25);
    jbCounterRead1.setEnabled(false);
    jbCounterRead1.addActionListener(this);
    jbCounterReadAll.setBounds(295, 75, 65, 25);
    jbCounterReadAll.setEnabled(false);
    jbCounterReadAll.addActionListener(this);
    jcbCounterPolling.setBounds(295, 105, 80, 20);
    jcbCounterPolling.setEnabled(false);
    jbCounterClear0.setBounds(360, 23, 65, 25);
    jbCounterClear0.setEnabled(false);
    jbCounterClear0.addActionListener(this);
    jbCounterClear1.setBounds(360, 48, 65, 25);
    jbCounterClear1.setEnabled(false);
    jbCounterClear1.addActionListener(this);
    jbCounterClearAll.setBounds(360, 75, 65, 25);
    jbCounterClearAll.setEnabled(false);
    jbCounterClearAll.addActionListener(this);
    jpIO.setLayout(null);
    jpIO.setBounds(5, 5, 440, 135);
    jpIO.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBorder(Color.black), "
    jpIO.add(jcbOutput0);
    jpIO.add(jcbOutput1);
    jpIO.add(jbOutputsRead);
    jpIO.add(jcbOutputPolling);
    jpIO.add(jcbInput0);
    jpIO.add(jcbInput1);
    jpIO.add(jbInputsRead);
    jpIO.add(jcbInputPolling);
    jpIO.add(jlCounter0);
    jpIO.add(jlCounter1);
    jpIO.add(jlCounterAll);
    jpIO.add(jlInterval);
    jpIO.add(jtfCounter0);
    jpIO.add(jtfCounter1);
    jpIO.add(jtfInterval);
    jpIO.add(jbCounterRead0);
    jpIO.add(jbCounterRead1);
    jpIO.add(jbCounterReadAll);
    jpIO.add(jcbCounterPolling);
    jpIO.add(jbCounterClear0);
    jpIO.add(jbCounterClear1);
    jpIO.add(jbCounterClearAll);
```

El siguiente fragmento de programa crea los elementos de mando y de indicación, que se resumen en la superficie con el concepto "Connection Control". El JPanel, al que se añaden estos elementos, está localmente instanciado, pues fuera del Constructor no se puede acceder más a él.

```
JLabel jlIP = new JLabel("IPAddress");
    jlIP.setBounds(20, 40, 120, 20);
    jtfIP.setBounds(20, 60, 120, 26);
    jtfIP.setHorizontalAlignment(JTextField.CENTER);
    JLabel jlPort = new JLabel("Port");
    jlPort.setBounds(145, 40, 70, 20);
    jtfPort.setBounds(145, 60, 70, 26);
    jtfPort.setHorizontalAlignment(JTextField.CENTER);
    JLabel jlPassword = new JLabel("Password");
    jlPassword.setBounds(220, 40, 95, 20);
    jtfPassword.setBounds(220, 60, 95, 26);
    jtfPassword.setHorizontalAlignment(JTextField.CENTER);
    jbConnect.setBounds(330, 25, 100, 25);
    jbConnect.addActionListener(this);
    jbDisconnect.setBounds(330, 60, 100, 25);
    jbDisconnect.setEnabled(false);
    jbDisconnect.addActionListener(this);
    JPanel jpConnect = new JPanel();
    jpConnect.setLayout(null);
    jpConnect.setBounds(5, 140, 440, 95);
    jpConnect.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBorder(Color.blaci
    jpConnect.add(jlIP);
    jpConnect.add(jlPort);
    jpConnect.add(jlPassword);
    jpConnect.add(jtfIP);
    jpConnect.add(jtfPort);
    jpConnect.add(jtfPassword);
    jpConnect.add(jbConnect);
    jpConnect.add(jbDisconnect);
```

La barra de estado informa durante el desarrollo del programa del estado de la unión de zócalo a un Web-IO. Ya durante el instanciamiento se le asignó el valor de Default "No Connection". En el Constructor se define la representación en forma de tamaño, posición y tipo del marco.

```
        jlStatusBar.setLayout(null);
        jlStatusBar.setBounds(1, 240, 450, 20);
        jlStatusBar.setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
```

Finalmente se determinan las características de la ventana y se añaden los elementos anteriormente creados. Como último paso se visualiza la ventana de programa. La fase de inicialización está así concluida y el programa está ahora listo para el servicio.

```
        setTitle("Web-IO ASCII Client");
        setLocation(400, 300);
        setLayout(null);
        setSize(460, 290);
        setResizable(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(jpIO);
        add(jpConnect);
        add(jlStatusBar);
        setVisible(true);
```

7. Procesar los sucesos del Button y del Timer

Procesamiento de sucesos (en general): Si se registró un ActionListener en los elementos de indicación y de mando (p. ej. botones y campos de texto), se llama el método "actionPerformed" con sucesos activados. El parámetro entregado incluye entre otros la información, de qué componente ha disparado el suceso.

```
public void actionPerformed(ActionEvent arg0) {
        ...
        }
```

Conexión: Pulsando la superficie de contacto "Connect" se inicia una conexión al Web-IO indicado. Los elementos de mando y de indicación se conectar según el estado a activo o inactivo y la barra de estado refleja el punto de avance de la conexión. Si se abrió bien el zócalo y se generaron con éxito los flujos para la entrada y salida de datos, arranca un Thread, que procesa los datos entrantes. Se inicia un Timer, que solicita datos conforme al comportamiento exigido de Polling. Aparece un error durante la conexión, se sale a través de un excepción.

```
    if (arg0.getSource() == jbConnect)
     {
     jbConnect.setEnabled(false);
     jlStatusBar.setText("Trying to connect to " + jtfIP.getText());
     try
      {
      soClientSocket = new Socket(jtfIP.getText(),Integer.parseInt(jtfPort.getText()));
      isInStream = soClientSocket.getInputStream();
      osOutStream = soClientSocket.getOutputStream();
      new Thread(this).start();
      tiPoll = new Timer(Integer.parseInt(jtfInterval.getText()),this);
      tiPoll.start();
      for (int i = 0; i < jpIO.getComponentCount();i++)
       {
       ((JComponent) jpIO.getComponent(i)).setEnabled(true);
       }
      jbDisconnect.setEnabled(true);
      jlStatusBar.setText("Connected to " + jtfIP.getText());
      return;
      }
     catch (NumberFormatException e)
      {
      e.printStackTrace();
      }
     catch (UnknownHostException e)
      {
      e.printStackTrace();
      }
     catch (IOException e)
      {
      e.printStackTrace();
      }
     jbConnect.setEnabled(true);
     jlStatusBar.setText("Error - No Connection");
     }
```

Conexión deseada: Pulsar la superficie de contacto "Disconnect" llama al método "disconnect". Este contiene instrucciones que desconectan controladamente la conexión y preparan al programa para la toma de una nueva conexión. Puesto que la conexión también puede introducirse a través de una excepción, la conexión se da en un método separado y no entro del método "actionPerformed".

```
  else if (arg0.getSource() == jbDisconnect) {
      disconnect();
     }
```

Outputs schalten: Manipulando la cajas de chequeo "jcbOutput0" y "jcbOutput1" puede conectarse la salida respectiva. Si una de las cajas de chequeo activa un suceso, dependiendo del estado fijado se envía un String a través del zócalo al Web-IO, que pone en práctica la selección.

```
    else if (arg0.getSource() == jcbOutput0)
     {
     if (jcbOutput0.isSelected())
      {
      write("GET /outputaccess0?PW=" + jtfPassword.getText() + "&State=ON&");
      }
      else
      {
      write("GET /outputaccess0?PW=" + jtfPassword.getText() + "&State=OFF&");
      }
     }
    else if (arg0.getSource() == jcbOutput1)
     {
     if (jcbOutput1.isSelected())
      {
      write("GET /outputaccess1?PW=" + jtfPassword.getText() + "&State=ON&");
      }
     else
      {
      write("GET /outputaccess1?PW=" + jtfPassword.getText() + "&State=OFF&");
      }
     }
```

Leer Outputs e Inputs, leer y borrar contador: Las entradas y salidas pueden leerse completamente a través de una superficie de contacto. Para ello se envía una correspondiente cadena de comando al Web-IO. Se comporta igualmente con los contadores. En cada superficie de contacto se ha depositado una cadena de comando, que activa el resultado correspondiente.

```
    else if (arg0.getSource() == jbOutputsRead) {
     write("GET /output?PW=" + jtfPassword.getText() + "&");
     }
    else if (arg0.getSource() == jbInputsRead) {
     write("GET /input?PW=" + jtfPassword.getText() + "&");
     }
    else if (arg0.getSource() == jbCounterRead0) {
     write("GET /counter0?PW=" + jtfPassword.getText() + "&");
     }
    else if (arg0.getSource() == jbCounterRead1) {
     write("GET /counter1?PW=" + jtfPassword.getText() + "&");
     }
    else if (arg0.getSource() == jbCounterReadAll) {
     write("GET /counter?PW=" + jtfPassword.getText() + "&");
     }
    else if (arg0.getSource() == jbCounterClear0) {
     write("GET /counterclear0?PW=" + jtfPassword.getText() + "&");
     }
    else if (arg0.getSource() == jbCounterClear1) {
     write("GET /counterclear1?PW=" + jtfPassword.getText() + "&");
     }
    else if (arg0.getSource() == jbCounterClearAll) {
     write("GET /counterclear?PW=" + jtfPassword.getText() + "&");
     }
```

Polling: El Timer, que se inició con la conexión, activa también cíclicamente un suceso. Según el estado de las tres cajas de chequeo de Polling, en cada ciclo se consultan las salidas, las entradas y los contadores.

```
else if (arg0.getSource() == tiPoll) {
     if (jcbOutputPolling.isSelected()) {
     write("GET /output?PW=" + jtfPassword.getText() + "&");
     }
     if (jcbInputPolling.isSelected()) {
     write("GET /input?PW=" + jtfPassword.getText() + "&");
     }
     if (jcbCounterPolling.isSelected()) {
     write("GET /counter?PW=" + jtfPassword.getText() + "&");
     }
    }
```

8. Procesar cambios en los campos de texto

Cambio del intervalo de Polling: Cambios del intervalo de Polling se activan sin confirmación. Un cambio se reconoce mediante KeyListeners, que se añadió al campo de texto "tfInterval". Para ello la clase tiene que implementar los métodos de la interfaz ("keyPressed"," KeyReleased" y "keyTyped"). Al cambiar el contenido del campo de texto, en el método "keyPressed" el valor actualizado se convierte en un formato int y se entrega al Timer. Si no se puede convertir el contenido del campo de texto en una cifra, se sale a través de la excepción. En este caso no se cambia la cuota de Polling.

```
public void keyPressed(KeyEvent arg0) {
    }
    public void keyReleased(KeyEvent arg0) {
     try {
     tiPoll.setDelay(Integer.parseInt(jtfInterval.getText()));
     } catch (NumberFormatException e) {
     }
    }
    public void keyTyped(KeyEvent arg0) {
    }
```

9. Recepción y procesamiento de datos

El método "run" se inicia al establecerse una conexión de zócalo y funciona como Thread "casi" paralelamente al propio programa. En el método el zócalo se controla permanentemente en un bucle while respecto a datos entrantes. Al interrumpirse la conexión se lee el valor -1, que es el criterio de interrupción del bucle. Al recibir un valor >0, éste se convierte en un signo según la tabla ASCII y se añade al string de recepción. Al recibir un 0 el string se ha recibido completamente y comienza el procesamiento. El comienzo de cada string ("output;", "input;", "counter;" y "counter") informa sobre la asignación de los datos. Después de la identificación de las informaciones se extraen éstas y se representan.

```
public void run() {
     int iInput, iState;
     String sIn = "";
     StringTokenizer stToken;
     try {
      while
     ((iInput = isInStream.read()) != -1) {
      if
     (iInput > 0) {
       sIn += (char) iInput;
      }
     else {
         if (sIn.startsWith("input")) {
        iState = Integer.parseInt(sIn.replaceFirst("input;", ""));
       jcbInput0.setSelected(((iState & 1) > 0) ? true : false);
        jcbInput1.setSelected(((iState & 2) > 0) ? true : false);
       }
     else if (sIn.startsWith("output")) {
        iState = Integer.parseInt(sIn.replaceFirst("output;",""));
       jcbOutput0.setSelected(((iState & 1) > 0) ? true : false);
        jcbOutput1.setSelected(((iState & 2) > 0) ? true : false);
       }
     else if (sIn.startsWith("counter")) {
       if (sIn.startsWith("counter;")) {
        sIn = sIn.replaceFirst("counter;", "");
        stToken = new StringTokenizer(sIn, ";");
        jtfCounter0.setText(stToken.nextToken());
        jtfCounter1.setText(stToken.nextToken());
       }
     else {
        stToken = new StringTokenizer(sIn, ";");
        if (stToken.nextToken().equals("counter0")) {
        jtfCounter0.setText(stToken.nextToken());
        }
     else {
        jtfCounter1.setText(stToken.nextToken());
        }
        sIn = sIn.replaceFirst("counter", "");
       }
       }
        sIn = "";
       }
      }
      disconnect();
      } catch
     (IOException e) {
      }
     }
```

10. Enviar strings de comando

Con el método "write" se escriben los strings de comando a transmitir al zócalo. La escritura se hace en dos pasos. La instrucción "write" entrega el string como cadena Byte al zócalo. La instrucción "flush" envía los Bytes al punto contrario. Si durante este proceso aparece un error, se interrumpe a través de la excepción.

```
private void write(String sOutput) {
     try {
     osOutStream.write(sOutput.getBytes());
     osOutStream.flush();
     }
     catch (IOException e) {
     }
    }
```

11. Desconexión

Las instrucciones para una desconexión correcta están colocadas en un método separado, que se llama por un lado a través de una superficie de contacto "Disconnect" y por el otro con una interrupción de la conexión. Primero se desactiva la superficie de contacto "Disconnect", se emite la noticia a través de la conexión a la línea de estado y se para el timer de Polling. A continuación se cierra el zócalo. Si el proceso tiene éxito, todos los elementos de entrada y de indicación se colocan en el estado de partida. Si no funciona el cierre, la superficie se ajusta de nuevo para una conexión existente.

```
private void disconnect() {
     jbDisconnect.setEnabled(false);
     jlStatusBar.setText("Trying to disconnect from " + jtfIP.getText());
     try {
     tiPoll.stop();
     soClientSocket.close();
     for (int i = 0; i < jpIO.getComponentCount(); i++) {
      ((JComponent)jpIO.getComponent(i)).setEnabled(false);
     }
     jbConnect.setEnabled(true);
     jlStatusBar.setText("No connection");
     return;
     } catch (IOException e) {
     e.printStackTrace();
     }
     jbDisconnect.setEnabled(true);
     jlStatusBar.setText("Error - Connected to " + jtfIP.getText());
    }
```

El programa ejemplo asiste todas las funciones corrientes del Web-IO en el modo String de comando, optimado para el Web-IO 2x entradas digitales, 2x salidas digitales. Para los otros modelos Web-IO tienen que realizarse adaptaciones en el programa. Otros ejemplos de programa para la programación del zócalo los encontrarán en las páginas de herramientas al Web-IO. Una descripción detallada de la interfaz del zócalo de los modelos Web-IO digitales la encontrarán en el manual de referencia.

Descargar el programa ejemplo

¿No tiene todavía un Web-IO y quiere probar el ejemplo presentado?
No hay problema: Le ponemos a disposición el Web-IO Digital 2xInput, 2xOutput PoE gratis durante 30 días. Rellene sencillamente un pedido muestra y le enviaremos el Web-IO para probar a cuenta abierta. Si nos devuelve el aparato dentro de los 30 días, le abonamos la factura completa.

Al pedido muestra

W&T
www.WuT.de

Le atendemos personalmente:

Wiesemann & Theis
GmbH
Porschestr. 12
42279 Wuppertal
Tel: +49 202/2680-110 (lu-vi de 8-17 horas)
Fax: +49-202/2680-265
info@wut.de