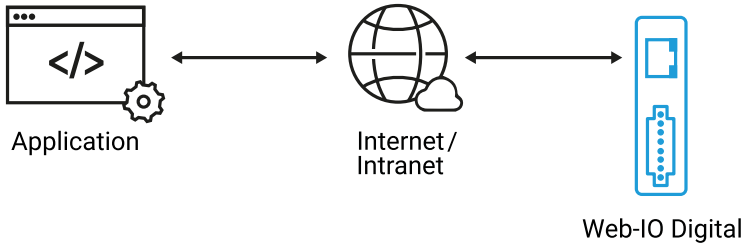
Applicazione relativa al Web-IO digitale:

Web-IO digitale con Java controllo e monitoraggio

Panoramica del prodotto
Panoramica dell'applicazione

Java è un linguaggio di programmazione ampiamente diffuso, semplice da apprendere e indipendente dalla piattaforma. Contiene già nella versione standard tutte le classi e tutti i metodi che sono necessari per la programmazione di applicazioni TCP/IP. Ciò caratterizza Java come tool ideale per creare applicazioni che comunicano con il Web-IO digitale.

Oltre all'ambiente Java corrente non servono ulteriori risorse per l'esecuzione.

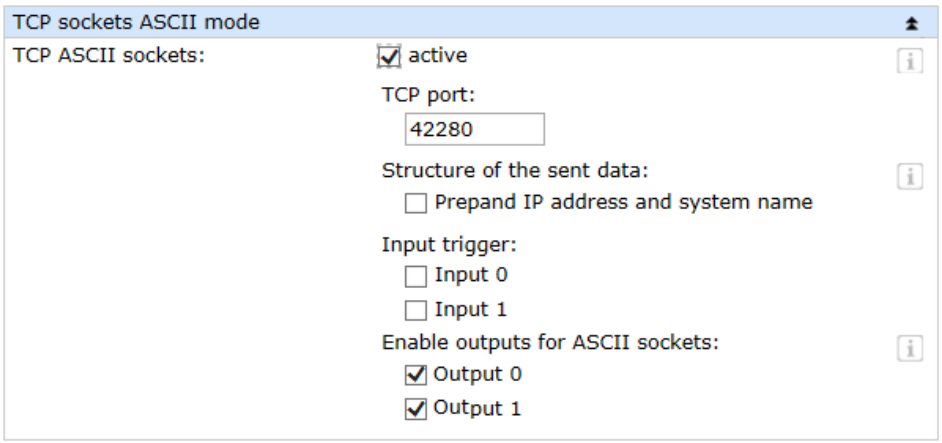


Il seguente esempio di programma permette il monitoraggio e il controllo di un Web-IO 2x digitale e mette a disposizione a tale scopo le seguenti funzioni:

- collegamento degli output
- Lettura manuale o automatica di input, output e counter
- lettura e cancellazione di uno o di tutti i counter

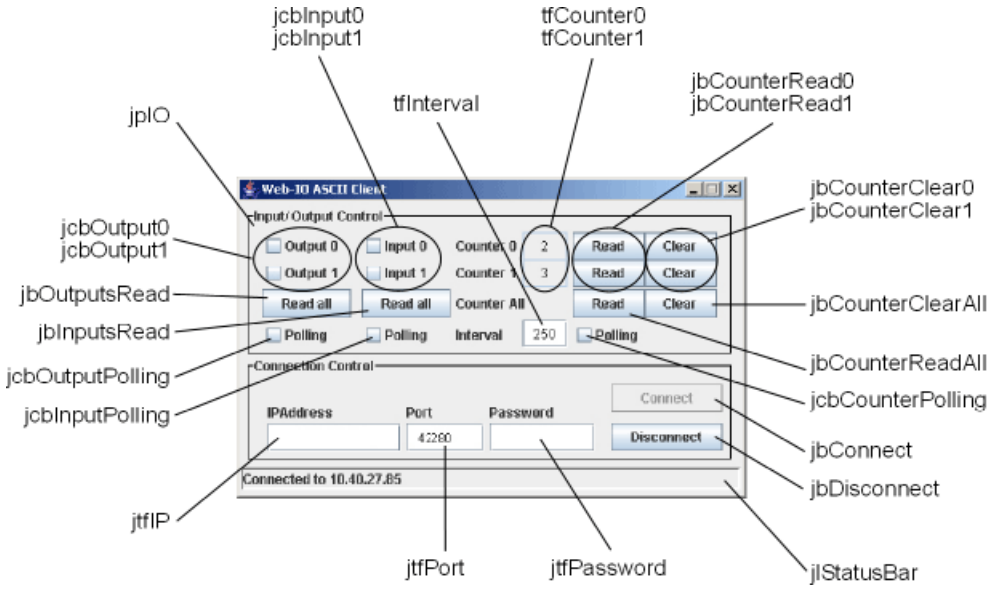
Preparativi

- Mettere sotto tensione il Web-IO e collegare gli IO
- Collegare il Web-IO alla rete
- Assegnazione di indirizzi IP
- Nel Web-IO nell'area *Vie di comunicazione >> API socket*Attivare i *socket ASCII TCP* e *autorizzare gli output per l'attivazione*



Composizione dei diversi elementi di comando e di visualizzazione

Nell'immagine che segue sono denominati i diversi elementi di comando e di visualizzazione con il nome utilizzato nel codice programma. L'assegnazione deve servire da riferimento e facilitare la comprensione dell'esempio che segue.



Nella denominazione dei singoli oggetti è utile utilizzare nomi che ne riprendono il significato. In questo esempio la prima parte del nome descrive il tipo dell'oggetto e la seconda parte la funzione.

Importazione delle sorgenti necessarie

All'inizio di ogni programma devono essere importate tutte le sorgenti necessarie.

```
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. Ampliamenti della classe

La classe "client" riceve tutte le proprietà della classe "JFrame", per visualizzare la finestra del programma e collega le interfacce "ActionListener" (riconoscimento degli eventi pulsante), "KeyListener" (riconoscimento delle modifiche nei campi di testo) e "Runnable" (possibilità di thread) mediante l'importazione dei relativi metodi.

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

Istanziamento degli elementi grafici

Per gli elementi grafici (pulsanti, caselle di spunta, etichette, pannello, campo di testo) vengono create istanze globali affinché siano a disposizione in ogni metodo del programma.

```
  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();
```

Elementi per la trasmissione dei dati e la temporizzazione

Per la trasmissione dei dati mediante l'interfaccia socket in questo esempio vengono utilizzate istanze delle classi "InputStream", "OutputStream" e "Socket".

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

Il timer, che controlla il polling, è derivato dalla classe "Timer" (javax.swing.Timer).

```
private Timer tiPoll;
```

Il metodo "main"

Il metodo "main" viene richiamato automaticamente all'avvio del programma. Richiama il costruttore della classe.

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

Costruttore

Nel costruttore vengono definite le dimensioni, la posizione e il comportamento degli elementi di visualizzazione e della finestra di programmazione. Il costruttore viene richiamato senza trasmissione di parametri.

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

Successivamente viene creato il pannello Java (JPanel) "jpIO". A tale scopo gli elementi di visualizzazione vengono dimensionati, eventualmente resi inattivi, per produrre lo stato iniziale e vengono dotati di un ascoltatore affinché si possa reagire alle immissioni dell'utente. Infine gli elementi vengono aggiunti 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), "Input/ Output Control"));
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);
```

Il frammento di programma successivo crea gli elementi di comando e di visualizzazione che sono raggruppati sull'interfaccia con il termine di "Connection Control". Il JPanel, a cui vengono aggiunti questi elementi, è istanziato localmente poiché oltre al costruttore non vi è alcun altro accesso.

```
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.black), "Connection Control"));
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 di stato informa durante l'esecuzione del programma sullo stato del collegamento socket a un Web-IO. Già durante l'istanziamento è stato assegnato il valore di default "No Connection". Nel costruttore viene definita la visualizzazione sotto forma di dimensioni, posizione e tipo di struttura.

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

Infine vengono definite le proprietà della finestra e vengono aggiunti gli elementi precedentemente creati. Come ultima fase viene resa visibile la finestra del programma. La fase di inizializzazione è con ciò conclusa e il programma è ora pronto per il funzionamento.

```
    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);
```

Elaborazione degli eventi pulsante e timer

Elaborazione eventi (in generale): se con gli elementi di visualizzazione e comando (ad es. pulsanti e campi di testo) è stato registrato un ActionListener, all'attivazione degli eventi viene richiamato il metodo "actionPerformed". Il parametro trasmesso contiene tra l'altro l'informazione sul componente che ha attivato l'evento.

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

Creazione del collegamento Azionando il pulsante "Connect" viene avviata la creazione di un collegamento al Web-IO indicato. Gli elementi di visualizzazione e comando vengono attivati o disattivati in base allo stato e la barra di stato riporta l'avanzamento della creazione del collegamento. Se l'apertura del socket e la generazione degli stream per l'immissione e l'emissione dei dati sono riuscite, viene avviato un thread che elabora i dati interessati. Viene avviato un timer che richiede i dati in base al comportamento di polling richiesto. Se durante la creazione del collegamento si presenta un errore, si verifica un'uscita come exception.

```
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");
}
```

Terminazione intenzionale del collegamento: L'azionamento del pulsante "Disconnect" richiama il metodo "disconnect". Questo metodo contiene istruzioni che terminano in modo controllato il collegamento e preparano il programma a ricevere un nuovo collegamento. Poiché la terminazione del collegamento può essere inizializzata anche come exception, la terminazione del collegamento avviene in un metodo separato e non all'interno del metodo "actionPerformed".

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

Collegamento degli output: Intervenendo sulle caselle di spunta "jcbOutput0" e "jcbOutput1" può essere attivata la relativa uscita. Se una delle caselle di spunta attiva un evento, in base allo stato impostato viene inviata al Web-IO mediante il socket una stringa che converte la selezione.

```
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&");
  }
}
```

Lettura degli output e input, lettura e cancellazione dei counter Gli output e gli input possono essere completamente letti di volta in volta mediante un'interfaccia. A tale scopo viene inviata al Web-IO una corrispondente stringa di comando. Lo stesso avviene con i counter. Per ogni interfaccia è memorizzata una stringa di comando che attiva il corrispondente evento.

```
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: Anche il timer che è stato avviato con la creazione del collegamento attiva ciclicamente un evento. In base allo stato delle tre caselle di spunta per il polling, ad ogni ciclo vengono interrogati gli output, gli input e i counter.

```
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() + "&");
  }
 }
```

Elaborazione di modifiche nei campi di testo

Modifica dell'intervallo di polling: le modifiche dell'intervallo di polling vengono attivate senza conferma. Una modifica viene riconosciuta mediante un KeyListener che è stato aggiunto al campo di testo "tfInterval". A tale scopo la classe deve implementare i metodi dell'interfaccia ("keyPressed"," KeyReleased" e "keyTyped"). Modificando il contenuto del campo di testo, nel metodo "keyPressed" il valore aggiornato viene convertito nel formato int e inoltrato al timer. Se il contenuto del campo di testo non può essere convertito in un numero, si verifica un'uscita come exception. La velocità di polling non viene modificata in questo caso.

```
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) {
}
```

Ricezione ed elaborazione dei dati

Il metodo "run" viene avviato a realizzazione avvenuta di un collegamento socket e gira come thread "per così dire" parallelamente al proprio programma. Nel metodo il socket viene sottoposto in un ciclo while a lungo al controllo dei dati interessati. All'interruzione del collegamento viene letto il valore 1 che rappresenta il criterio di interruzione del ciclo. Se viene ricevuto un valore > 0, questo valore viene convertito in base alla tabella ASCII in un carattere e viene aggiunto alla stringa di ricezione. In caso di ricezione di uno 0 la stringa è stata ricevuta completamente e ha inizio l'elaborazione. L'inizio di ogni stringa ("output;", "input;", "counter;" e "counter") fornisce spiegazioni sull'assegnazione dei dati. Dopo l'identificazione delle informazioni, esse vengono estratte e visualizzate.

```
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) {
  }
}
```

Invio stringhe di comando

Con il metodo "write" le stringhe di comando da trasmettere vengono scritte sul socket. La scrittura avviene in due fasi. L'istruzione "write" trasferisce la stringa come stringa di byte al socket. L'istruzione "flush" invia i byte al ricevitore. Se durante questa procedura si presenta un errore, si verifica un'interruzione come exception.

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

11. Terminazione del collegamento

Le istruzioni per una terminazione ordinata del collegamento sono rilocate in un metodo separato che da un lato viene richiamato mediante l'interfaccia "Disconnect" e dall'altro in caso di interruzione del collegamento. Innanzitutto viene disattivato il pulsante "Disconnect", nella riga di stato viene visualizzato il messaggio sulla terminazione del collegamento e il timer del polling viene arrestato. Successivamente viene chiuso il socket. Se questa procedura riesce, tutti gli elementi di immissione e di visualizzazione vengono portati allo stato iniziale. Se la chiusura fallisce, l'interfaccia viene reimpostata per un collegamento esistente.

```
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());
}
```

Il programma esempio supporta tutte le comuni funzioni del Web-IO nella modalità stringa di comando, ottimizzata per il Web-IO 2x input digitale, 2x output digitale. Gli altri modelli Web-IO devono eventualmente essere adattati al programma. Ulteriori esempi di programma per la programmazione socket sono riportati nelle pagine dei tool per il Web-IO. Una descrizione dettagliata sull'interfaccia socket dei modelli Web-IO digitali è riportata nel manuale di riferimento.

Download esempio di programma

Prodotti

Web-IO 4.0 digitale
2x input, 2x output

All'occorrenza possibilità di alimentazione anche tramite PoE

Web-IO 4.0 digitale
12x input, 12x output

12x ingressi,
12x uscite

Altri web-IO

Tutti i Web-IO digitali W&T da 24 V

W&T
www.WuT.de

Saremo lieti di fornirvi una consulenza personalizzata!

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