Heimautomation unter Linux mit pvbrowser und C-CONTROL Station

Einleitung

pvbrowser ® ist eine Prozessvisualisierung, die Open Source (GPL Lizenz) ist und unter

http://pvbrowser.org

heruntergeladen werden kann. Die Software kann unter Linux und Windows betrieben werden.

Die C-CONTROL Station ist eine kostengünstige programmierbare Steuerung von

http://www.conrad.de

Artikel-Nr.: 125113 – 62

Falls Sie grundlegende Programmiererfahrung haben oder programmieren lernen möchten, können Sie mit dieser Kombination eine vollständige Heimautomation inklusive Visualisierung über das LAN oder das Internet realisieren. Das Design der Bildschirmmasken erfolgt mit einem graphischen Editor.

Download und Installation

pvbrowser downloaden und installieren:

```
wget http://pvbrowser.de/pvbrowser/tar/pvb.tar.gz
tar -zxf pvb.tar.gz
su -c "cd pvb; ./install.sh"
rm -rf pvb/
```

Beachten Sie, dass Sie die Qt4 Bibliotheken benötigen. Z.B. Paket „qt“ unter SuSE Linux 10.1

Programmiersoftware für die C-CONTROL Station downloaden und installieren:

```
wget http://pvbrowser.de/pvbrowser/tar/ccontrolSDK.tar.gz
tar -zxf ccontrolSDK.tar.gz
cd ccontrolSDK
./install.sh"
```

Die Programmiersoftware stammt eigentlich von
http://freshmeat.net/projects/cctrlin
Aber wir haben ein paar kleine Probleme fixen müssen, damit die Software unter modernen Linux Distributionen läuft. Deshalb stellen wir die modifizierte Software zur Verfügung.

Erste Schritte

Öffnen Sie einen Terminalemulator und geben folgendes ein:

```
mkdir neues_verzeichnis
cd neues_verzeichnis
cp -r ../ccontrolSDK/demo/ .
cd demo/pvbrowser
```

In diesem Verzeichnis finden Sie die Datei „basic_sps.bas“. Es handelt sich um ein Basic Programm, das auf der Steuerung laufen kann. Editieren Sie die Datei und sehen sich die Befehle an. Das Programm läuft in einer Endlosschleife. Falls ein Zeichen vom PC empfangen wird, gibt

die Steuerung Ihren Zustand auf die serielle Schnittstelle aus und die Visualisierung ist in der Lage den Zustand zu übernehmen. Sonst führt die Steuerung Ihr SPS-Programm aus.
Das von Conrad mitgelieferte Verbindungskabel überbrückt die Hardware- Handshake- Leitungen leider nicht. Deshalb sollten Sie RTS/CTS im Stecker überbrücken.
Siehe:
http://de.wikipedia.org/wiki/Nullmodem-Kabel

Alternativ können Sie die Schnittstelle des PC auch per shell script setzen. Beachten Sie auch, dass defaultmässig normale Benutzer eventuell keinen Zugriff auf die serielle Schnittstelle haben.

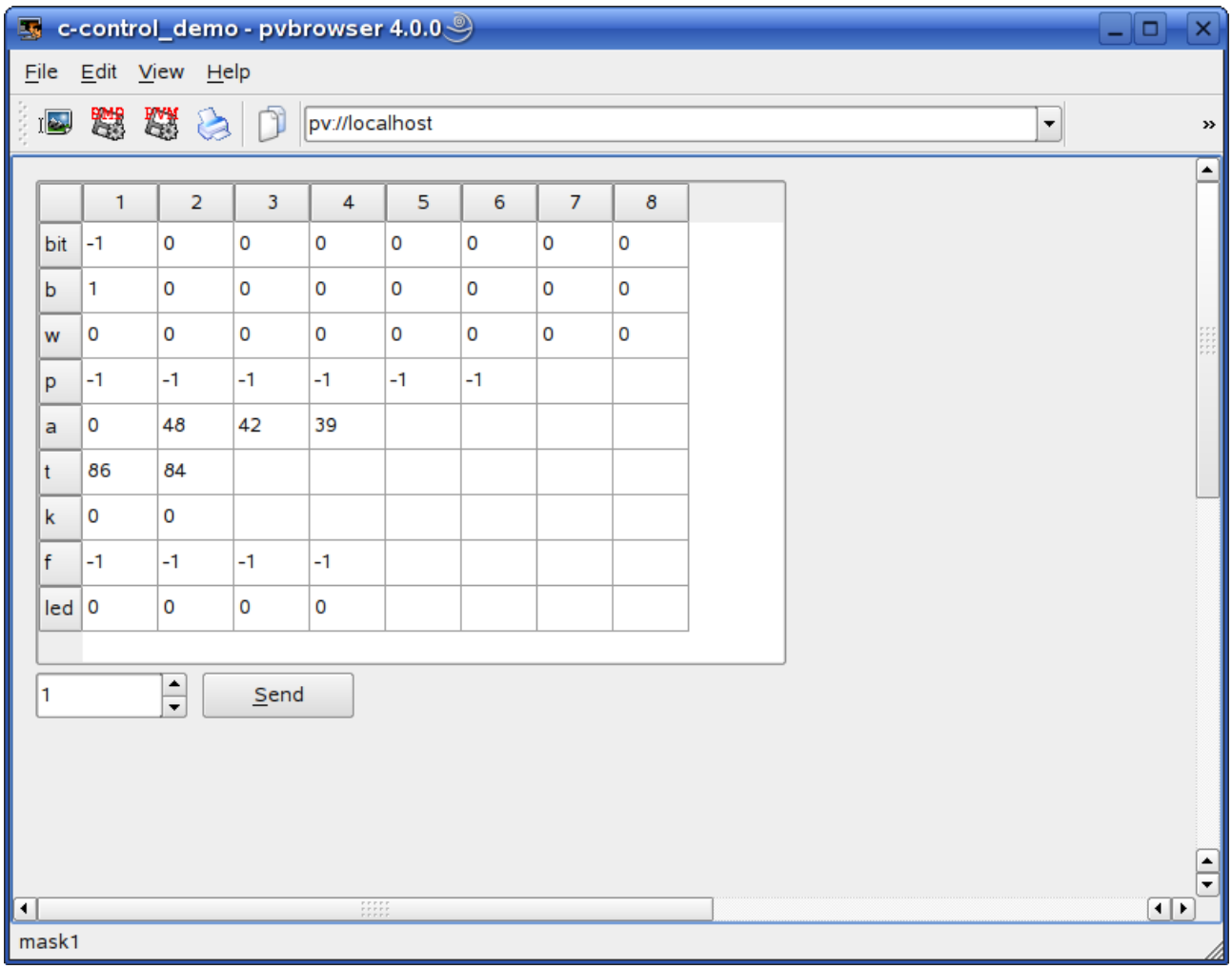
```
demo/pvbrowser> cat ./init_tty.sh
#!/bin/bash
echo "enter root password:"
su -c "chmod ugoa+rw /dev/ttyS0; stty 9600 -crtscts -F /dev/ttyS0"
echo "Parameters of /dev/ttyS0 set to:"
stty -a -F /dev/ttyS0
```

Das Basic Programm können Sie nun mit folgendem Befehl übersetzen und in die Steuerung laden. Dazu müssen Steuerung und PC verbunden sein und Sie müssen die RESET- Taste auf der Steuerung drücken.

```
ccontrol.sh basic_sps
```

Nachdem das Programm in die Steuerung geladen ist, betätigen Sie die START- Taste an der Steuerung. Damit beginnt die Steuerung das Programm auszuführen. Zum Testen können Sie das kleine Test Programm „./com“ auf dem PC aufrufen, um die Ausgaben der Steuerung zu verfolgen. Dabei sollte die LED TX auf der Steuerung flackern. Das Test Programm kann mit „Strg-C“ abgebrochen werden.
Nach diesem Schritten sind Sie sicher, dass die Steuerung korrekt läuft. Nun können Sie die Demo-Visualisierung starten.

```
xterm -e "./c-control_demo" &
pvbrowser &
```



In der Tabelle können Sie den Zustand der Steuerung verfolgen

Erweiterung der Visualisierung

Zur Entwicklung der pvbrowser Visualisierung wird die integrierte Entwicklungsumgebung pvdevelop verwendet.

```
pvdevelop c-control_demo.pvproject
```

Zur näheren Beschreibung von pvbrowser lesen Sie bitte:
http://pvbrowser.de/home/mediawiki/index.php/De:pvbrowser_manual

Sie sollten ebenfalls eine kurze Einführung in die Programmierung von ANSI-C lesen.

Mit diesem Rüstzeug sollten Sie in der Lage sein, den Sourcecode des c-control_demo zu verstehen und ihn Schritt für Schritt zu erweitern.

Zusammenfassung

Mit pvbrowser, der C-CONTROL Station und etwas Programmierkenntnissen sollten Sie in der Lage sein, eigene Visualisierungen zu erstellen. Beachten Sie auch, das Webseiten heute ebenfalls Programmierkenntnisse erfordern, denn viele Seiten werden mit Hilfe der Scriptsprache PHP erstellt.
Die Visualisierung lässt sich über Netzwerke (auch Internet) von anderen Computern unter Linux

als auch Windows bedienen. Sie müssen lediglich die URL Ihres selbst geschriebenen Servers in pvbrowser eingeben.

Anhang

Basic Programm in der Steuerung

```
'*****************************************************
'
' C-Control/BASIC       basic_sps.bat
'
' Systemvoraussetzungen:
'
' - serielle Verbindung zum PC
' - Verbindung mit zu pvbrowser
'
'*****************************************************

' Definitonen ********************************************************
' Ports
DEFINE P1 PORT[1]
DEFINE P2 PORT[2]
DEFINE P3 PORT[3]
DEFINE P4 PORT[4]
DEFINE P5 PORT[5]
DEFINE P6 PORT[6]

DEFINE K1 PORT[7]
DEFINE K2 PORT[8]

DEFINE F1 PORT[9]
DEFINE F2 PORT[10]
DEFINE F3 PORT[11]
DEFINE F4 PORT[12]

DEFINE LED1 PORT[13]
DEFINE LED2 PORT[14]
DEFINE LED3 PORT[15]
DEFINE LED4 PORT[16]

DEFINE A1 AD[1]
DEFINE A2 AD[2]
DEFINE A3 AD[3]
DEFINE A4 AD[4]

DEFINE T1 AD[5]
DEFINE T2 AD[6]

'DEFINE U1 AD[7]
'DEFINE U2 AD[8]

'Variablen
DEFINE bit1 BIT[1]
DEFINE bit2 BIT[2]
DEFINE bit3 BIT[3]
DEFINE bit4 BIT[4]
DEFINE bit5 BIT[5]
DEFINE bit6 BIT[6]
DEFINE bit7 BIT[7]
DEFINE bit8 BIT[8]

DEFINE b1  BYTE
DEFINE b2  BYTE
DEFINE b3  BYTE
DEFINE b4  BYTE
DEFINE b5  BYTE
DEFINE b6  BYTE
DEFINE b7  BYTE
DEFINE b8  BYTE

DEFINE w1  WORD
DEFINE w2  WORD
DEFINE w3  WORD
DEFINE w4  WORD
DEFINE w5  WORD
```

```
DEFINE w6  WORD
DEFINE w7  WORD
DEFINE w8  WORD

' Start des Programms ***************************************
#start
  b1  = 0
  if rxd then get b1
  if b1 = 0 then goto step
  ' if b1 <> 0 then alles ausgeben und eventuell b1 interpretieren
  print ">"
  print "bit=", bit1, bit2, bit3, bit4, bit5, bit6, bit7, bit8
  print "b=", b1, b2, b3, b4, b5, b6, b7, b8
  print "w=", w1, w2, w3, w4, w5, w6, w7, w8
  print "P=", P1, P2, P3, P4, P5, P6
  print "A=", A1, A2, A3, A4
  print "T=", T1, T2
  print "K=", K1, K2
  print "F=", F1, F2, F3, F4
  print "LED=", LED1, LED2, LED3, LED4

'************************************************************
' Hier kann man b1 interpretieren
'************************************************************
#interpret

'************************************************************
' Hier kommt das SPS Programm (Im Moment relativ nutzlos)
'************************************************************
#step
  if F1 = -1 then LED1 = 0
  if F2 = -1 then LED2 = 0
  if F3 = -1 then LED3 = 0
  if F4 = -1 then LED4 = 0
  if F1 = 0 then LED1 = 1
  if F2 = 0 then LED2 = 1
  if F3 = 0 then LED3 = 1
  if F4 = 0 then LED4 = 1
  goto start

' Ende des Programms *****************************************************
#ende
  end
```

Hauptprogramm der Visualisierung mit zwei zusätzlichen Threads, die parallel zum eigentlichen Hauptprogramm laufen

```
//***************************************************************************
//                          main.cpp  -  description
//                             ------------------
//  begin            : So Dez 10 10:23:09 2006
//  generated by     : pvdevelop (C) 2000-2006 by Lehrig Software Engineering
//  email            : lehrig@t-online.de
//***************************************************************************
#include "pvapp.h"
// todo: comment me out. you can insert these objects as extern in your masks.
//rlModbusClient
modbus(modbusdaemon_MAILBOX,modbusdaemon_SHARED_MEMORY,modbusdaemon_SHARED_MEMORY_SIZE);
//rlSiemensTCPClient
siemensTCP(siemensdaemon_MAILBOX,siemensdaemon_SHARED_MEMORY,siemensdaemon_SHARED_MEMORY_SIZE);
//rlPPIClient        ppi(ppidaemon_MAILBOX,ppidaemon_SHARED_MEMORY,ppidaemon_SHARED_MEMORY_SIZE);

rlThread      station;       // Dieser Thread empfaengt die Daten von der C-CONTROL Station
rlThread      poll;          // Dieser Thread pollt die C-CONTROL Station zyklisch
rlFifo        fifo(100);     // Der Fifo dient zur Kommunikation der Threads
StationStatus status;        // Hier wird das Prozessabbild der C-CONTROL Station gespeichert

void *poller(void *arg)      // Thread poller()
{
  char b = 1;
  while(1)                   // Endlosschleife
  {
    fifo.write(&b,1);        // Poll Kommando an get_station() schicken
    rlsleep(500);            // 500 Millisekunden schlafen
  }
  return arg;
}
```

```
void *get_station(void *arg) // Thread get_station()
{
  rlSerial tty;              // Serialle Schnittstelle
  int ret;
  unsigned char command;
  char line[132];

  memset(&status,0,sizeof(status));                                  // status = 0 setzen
  ret = tty.openDevice("/dev/ttyS0",B9600,1,0,8,1,rlSerial::NONE);  // Schnittstelle initialisieren
  printf("start ret=%d\n",ret);
  while(1)
  {
    fifo.read(&command,1);             // Auf Kommando von anderen Threads warten
    printf("command=%d\n",command);
    tty.writeChar(command);            // Kommando an C-CONTROL Station ausgeben
    while(1)
    {
      ret = tty.select(1000);          // maximal 1000 Millisekunden auf Zeichen warten
      if(ret == 1)                     // Daten sind verfuegbar
      {
        ret = tty.readChar();          // 1 Zeichen lesen
      }
      else                             // Timeout
      {
        ret = 0;
        while(fifo.poll() == rlFifo::DATA_AVAILABLE) fifo.read(&command,1); // Fifo leeren
        tty.writeChar(command);        // C-CONTROL Station pollen
        printf("timeout\n");
      }
      if(ret == 0x0a)                  // ret == Line Feed
      {
        tty.readLine((unsigned char *) line,sizeof(line)); // Zeile von Schnittstelle lesen
        //printf("%s\n",line);

        // Alle Informationen, die empfangen werden, in status einlesen
        if(strncmp(line,"bit=",4) == 0)
        {
          sscanf(line,"bit=\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d",
                      &status.bit[1],&status.bit[2],&status.bit[3],&status.bit[4],
                      &status.bit[5],&status.bit[6],&status.bit[7],&status.bit[8]);
        }
        else if(strncmp(line,"b=",2) == 0)
        {
          sscanf(line,"b=\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d",
                      &status.b[1],&status.b[2],&status.b[3],&status.b[4],
                      &status.b[5],&status.b[6],&status.b[7],&status.b[8]);
        }
        else if(strncmp(line,"w=",2) == 0)
        {
          sscanf(line,"w=\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d",
                      &status.w[1],&status.w[2],&status.w[3],&status.w[4],
                      &status.w[5],&status.w[6],&status.w[7],&status.w[8]);
        }
        else if(strncmp(line,"P=",2) == 0)
        {
          sscanf(line,"P=\t%d\t%d\t%d\t%d\t%d\t%d\t%d",
                      &status.p[1],&status.p[2],&status.p[3],&status.p[4],
                      &status.p[5],&status.p[6]);
        }
        else if(strncmp(line,"A=",2) == 0)
        {
          sscanf(line,"A=\t%d\t%d\t%d\t%d",
                      &status.a[1],&status.a[2],&status.a[3],&status.a[4]);
        }
        else if(strncmp(line,"T=",2) == 0)
        {
          sscanf(line,"T=\t%d\t%d",
                      &status.t[1],&status.t[2]);
        }
        else if(strncmp(line,"K=",2) == 0)
        {
          sscanf(line,"K=\t%d\t%d",
                      &status.k[1],&status.k[2]);
        }
        else if(strncmp(line,"F=",2) == 0)
        {
          sscanf(line,"F=\t%d\t%d\t%d\t%d",
                      &status.f[1],&status.f[2],&status.f[3],&status.f[4]);
        }
        else if(strncmp(line,"LED=",4) == 0)
```

```
        {
          sscanf(line,"LED=\t%d\t%d\t%d\t%d",
                      &status.led[1],&status.led[2],&status.led[3],&status.led[4]);
          break;
        }
      }
    }
    ret = tty.select(1000); // abschliessendes 0x0a Zeichen lesen
    if(ret == 1) ret = tty.readChar();
  }
  printf("exit\n");
  return arg;
}

int pvMain(PARAM *p)
{
int ret;

  pvSetCaption(p,"c-control_demo");
  pvResize(p,0,1280,1024);
  //pvScreenHint(p,1024,768); // this may be used to automatically set the zoomfactor
  ret = 1;
  pvGetInitialMask(p);
  if(strcmp(p->initial_mask,"mask1") == 0) ret = 1;

  while(1)
  {
    switch(ret)
    {
      case 1:
        pvStatusMessage(p,-1,-1,-1,"mask1");
        ret = show_mask1(p);
        break;
      default:
        return 0;
    }
  }
}

#ifdef USE_INETD
int main(int ac, char **av)
{
PARAM p;

  pvInit(ac,av,&p);
  /* here you may interpret ac,av and set p->user to your data */
  pvMain(&p);
  return 0;
}
#else  // multi threaded server
int main(int ac, char **av)
{
PARAM p;
int   s;

  pvInit(ac,av,&p);
  /* here you may interpret ac,av and set p->user to your data */
  station.create(get_station,NULL);    // Thread get_station() starten
  poll.create(poller,NULL);            // Thread poller() starten
  while(1)
  {
    s = pvAccept(&p);                  // Auf Clienten warten
    if(s != -1) pvCreateThread(&p,s);  // Thread zur Behandlung des Clienten starten
    else        break;
  }
  return 0;
}
#endif
```

Dieser Sourcecode muss nicht geändert werden, wenn Sie Ihre Visualisierung erweitern möchten.

„Slot“ Funktionen der Visualisierung die von Ihnen ausgefüllt werden könnnen

Die IDE pvdevelop wird Sie bei dieser Arbeit unterstützen.

```
//#################################################################
//# mask1_slots.h for ProcessViewServer created: So Dez 10 10:23:09 2006
//# please fill out these slots
//# here you find all possible events
//# Yours: Lehrig Software Engineering
//#################################################################

extern rlFifo        fifo;   // Fifo zur Kommunikation mit Thread get_station()
extern StationStatus status; // Prozessabbild der C-CONTROL Station

// todo: uncomment me if you want to use this data aquisiton
// also uncomment this classes in main.cpp and pvapp.h
// also remember to uncomment rllib in the project file
//extern rlModbusClient     modbus;
//extern rlSiemensTCPClient siemensTCP;
//extern rlPPIClient        ppi;

typedef struct // (todo: define your data structure here)
{
  unsigned char command; // Kommando, das zur C-CONTROL Station geschickt werden soll
}
DATA;

static int slotInit(PARAM *p, DATA *d) // Wird beim Anzeigen der Maske aufgerufen
{
  if(p == NULL || d == NULL) return -1;
  memset(d,0,sizeof(DATA));            // Locale Daten = 0 setzen
  d->command = 1;                      // command auf 1 setzen
  pvTablePrintf(p,table,-1,0,"bit");   // Beschriftung der Tabelle festlegen
  pvTablePrintf(p,table,-1,1,"b");
  pvTablePrintf(p,table,-1,2,"w");
  pvTablePrintf(p,table,-1,3,"p");
  pvTablePrintf(p,table,-1,4,"a");
  pvTablePrintf(p,table,-1,5,"t");
  pvTablePrintf(p,table,-1,6,"k");
  pvTablePrintf(p,table,-1,7,"f");
  pvTablePrintf(p,table,-1,8,"led");
  pvSetColumnWidth(p,table,0,50);      // Spaltenbreite in der Tabelle festlegen
  pvSetColumnWidth(p,table,1,50);
  pvSetColumnWidth(p,table,2,50);
  pvSetColumnWidth(p,table,3,50);
  pvSetColumnWidth(p,table,4,50);
  pvSetColumnWidth(p,table,5,50);
  pvSetColumnWidth(p,table,6,50);
  pvSetColumnWidth(p,table,7,50);
  return 0;
}

static int slotNullEvent(PARAM *p, DATA *d) // Wird zyklisch aufgerufen Siehe: Kommandozeilenoption
-sleep=
{
  int i,j;

  if(p == NULL || d == NULL) return -1;
  for(i=0; i<8; i++)                       // Zellen der Tabelle aktualisieren
  {
    j = i+1;
    pvTablePrintf(p,table,i,0,"%d",status.bit[j]);
  }
  for(i=0; i<8; i++)
  {
    j = i+1;
    pvTablePrintf(p,table,i,1,"%d",status.b[j]);
  }
  for(i=0; i<8; i++)
  {
    j = i+1;
    pvTablePrintf(p,table,i,2,"%d",status.w[j]);
  }
  for(i=0; i<6; i++)
  {
    j = i+1;
    pvTablePrintf(p,table,i,3,"%d",status.p[j]);
  }
  for(i=0; i<4; i++)
  {
    j = i+1;
    pvTablePrintf(p,table,i,4,"%d",status.a[j]);
  }
```

```
  for(i=0; i<2; i++)
  {
    j = i+1;
    pvTablePrintf(p,table,i,5,"%d",status.t[j]);
  }
  for(i=0; i<2; i++)
  {
    j = i+1;
    pvTablePrintf(p,table,i,6,"%d",status.k[j]);
  }
  for(i=0; i<4; i++)
  {
    j = i+1;
    pvTablePrintf(p,table,i,7,"%d",status.f[j]);
  }
  for(i=0; i<4; i++)
  {
    j = i+1;
    pvTablePrintf(p,table,i,8,"%d",status.led[j]);
  }
  return 0;
}

static int slotButtonEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  if(id == sendButton) // Falls sendButton gedrueckt ist, command an Thread get_station() schicken
  {
    unsigned char b = (unsigned char) d->command;
    fifo.write(&b,1);
  }
  return 0;
}

static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  return 0;
}

static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  return 0;
}

static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)
{
  if(p == NULL || id == 0 || d == NULL || text == NULL) return -1;
  return 0;
}

static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)
{
  if(p == NULL || id == 0 || d == NULL || val < -1000) return -1;
  if(id == spinBox) d->command = val; // command merken
  return 0;
}

static int slotCheckboxEvent(PARAM *p, int id, DATA *d, const char *text)
{
  if(p == NULL || id == 0 || d == NULL || text == NULL) return -1;
  return 0;
}

static int slotRadioButtonEvent(PARAM *p, int id, DATA *d, const char *text)
{
  if(p == NULL || id == 0 || d == NULL || text == NULL) return -1;
  return 0;
}

static int slotGlInitializeEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  return 0;
}

static int slotGlPaintEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
```

```
  return 0;
}

static int slotGlResizeEvent(PARAM *p, int id, DATA *d, int width, int height)
{
  if(p == NULL || id == 0 || d == NULL || width < 0 || height < 0) return -1;
  return 0;
}

static int slotGlIdleEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  return 0;
}

static int slotTabEvent(PARAM *p, int id, DATA *d, int val)
{
  if(p == NULL || id == 0 || d == NULL || val < -1000) return -1;
  return 0;
}

static int slotTableTextEvent(PARAM *p, int id, DATA *d, int x, int y, const char *text)
{
  if(p == NULL || id == 0 || d == NULL || x < -1000 || y < -1000 || text == NULL) return -1;
  return 0;
}

static int slotTableClickedEvent(PARAM *p, int id, DATA *d, int x, int y, int button)
{
  if(p == NULL || id == 0 || d == NULL || x < -1000 || y < -1000 || button < 0) return -1;
  return 0;
}

static int slotSelectionEvent(PARAM *p, int id, DATA *d, int val, const char *text)
{
  if(p == NULL || id == 0 || d == NULL || val < -1000 || text == NULL) return -1;
  return 0;
}

static int slotClipboardEvent(PARAM *p, int id, DATA *d, int val)
{
  if(p == NULL || id == 0 || d == NULL || val < -1000) return -1;
  return 0;
}

static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)
{
  if(p == NULL || id == 0 || d == NULL || text == NULL) return -1;
  //pvPopupMenu(p,-1,"Menu1,Menu2,,Menu3");
  return 0;
}

static int slotKeyboardEvent(PARAM *p, int id, DATA *d, int val, int modifier)
{
  if(p == NULL || id == 0 || d == NULL || val < -1000 || modifier < -1000) return -1;
  return 0;
}

static int slotMouseMovedEvent(PARAM *p, int id, DATA *d, float x, float y)
{
  if(p == NULL || id == 0 || d == NULL || x < -1000 || y < -1000) return -1;
  return 0;
}

static int slotMousePressedEvent(PARAM *p, int id, DATA *d, float x, float y)
{
  if(p == NULL || id == 0 || d == NULL || x < -1000 || y < -1000) return -1;
  return 0;
}

static int slotMouseReleasedEvent(PARAM *p, int id, DATA *d, float x, float y)
{
  if(p == NULL || id == 0 || d == NULL || x < -1000 || y < -1000) return -1;
  return 0;
}

static int slotUserEvent(PARAM *p, int id, DATA *d, const char *text)
{
  if(p == NULL || id == 0 || d == NULL || text == NULL) return -1;
  return 0;
```

```
}
```