pvbrowser handbuch

http://pvbrowser.org

19. Juli 2016

Inhaltsverzeichnis

Danksagung

Die Idee zu pvbrowser entstand bei der Firma SMS Siemag AG. Dort wollte man eine Prozessvisualisierung haben, bei der man den Quellcode zur Verfügung hat, um gegebenenfalls eigene Anpassungen zu machen. Es wurden Lösungen auf Java Basis und Qt Basis vorgeschlagen. Die Java Variante wurde von SMS Siemag AG bevorzugt. Aber die Qt Variante wurde als Open Source Projekt unter dem Namen pvbrowser ebenfalls angegangen.
Die besonderen Merkmale von pvbrowser sollten Plattformunabhängigkeit und konsequente Client/Server Architektur sein.
Die Java Variante ist nicht lange verfolgt worden, aber pvbrowser wurde sehr bald bei SMS Siemag AG eingesetzt. Hier muss den Mitarbeitern der Firma gedankt werden, die die ersten Versionen getestet und eingesetzt haben und das auch heute noch tun. Ohne deren Unterstützung wäre es kaum möglich gewesen, die Lizenzkosten für die kommerzielle Version von Qt aufzubringen.
Nach der Veröffentlichung von pvbrowser wurde der Kreis von Benutzern und Entwicklern, die zu pvbrowser beigetragen haben immer größer. Ohne die Benutzer und Entwickler wäre es kaum möglich gewesen, den heutigen Stand zu erreichen. Es wurden viele Ideen für neue Fähigkeiten von pvbrowser entwickelt und Fehler beseitigt.
Den Studenten und deren Professoren, die Diplomarbeiten in Zusammenhang mit unserem Projekt durchgeführt haben, gilt unser Dank für deren hervorragende Beiträge.
Der Firma Nokia/Trolltech gilt unser Dank, dass Qt unter LGPL Lizenz gestellt wurde. Der Support von Nokia/Trolltech hat uns auch sehr dabei geholfen, schwierige Probleme zu bewältigen. Außerdem sind wir sehr zufrieden mit der rasanten Entwicklung, die Qt genommen hat.
Von pvbrowser werden einige fremde Open Source Komponenten verwendet. Wir danken hiermit allen anderen Projekten aus der Open Source Szene, ohne deren Entwicklungen die Gesamtlösung unmöglich hätte realisiert werden können.
Unser besondere Dank gilt natürlich auch unseren Familien, die viel Geduld mit uns aufbringen mussten.

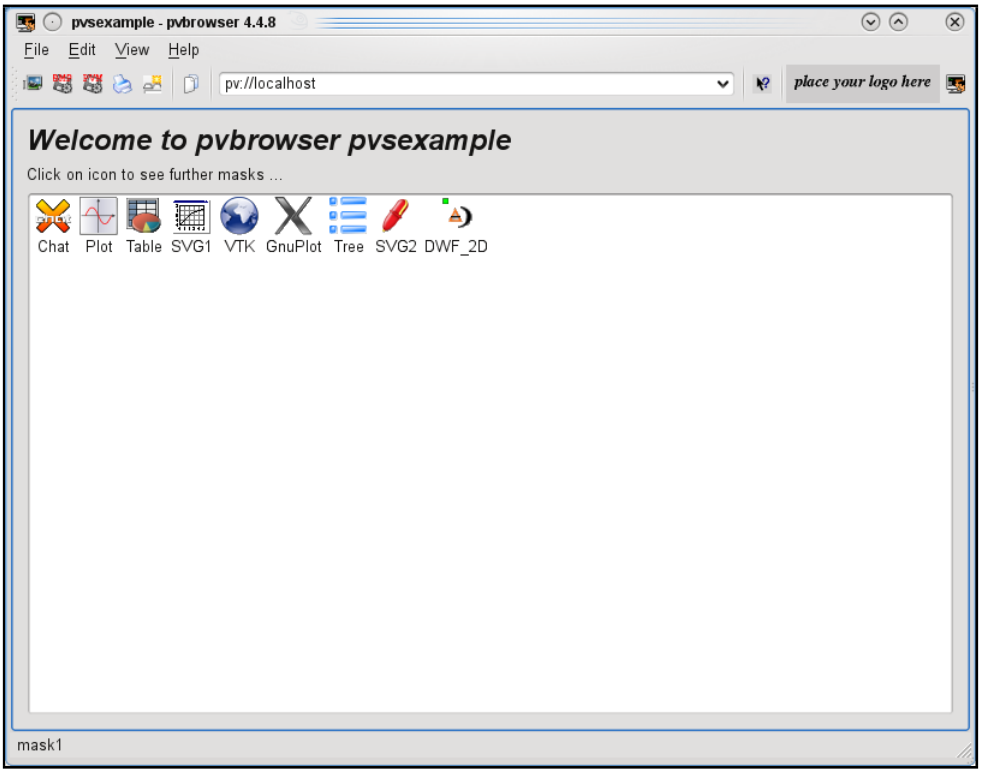

Abbildung 1: pvsexample Start-Maske

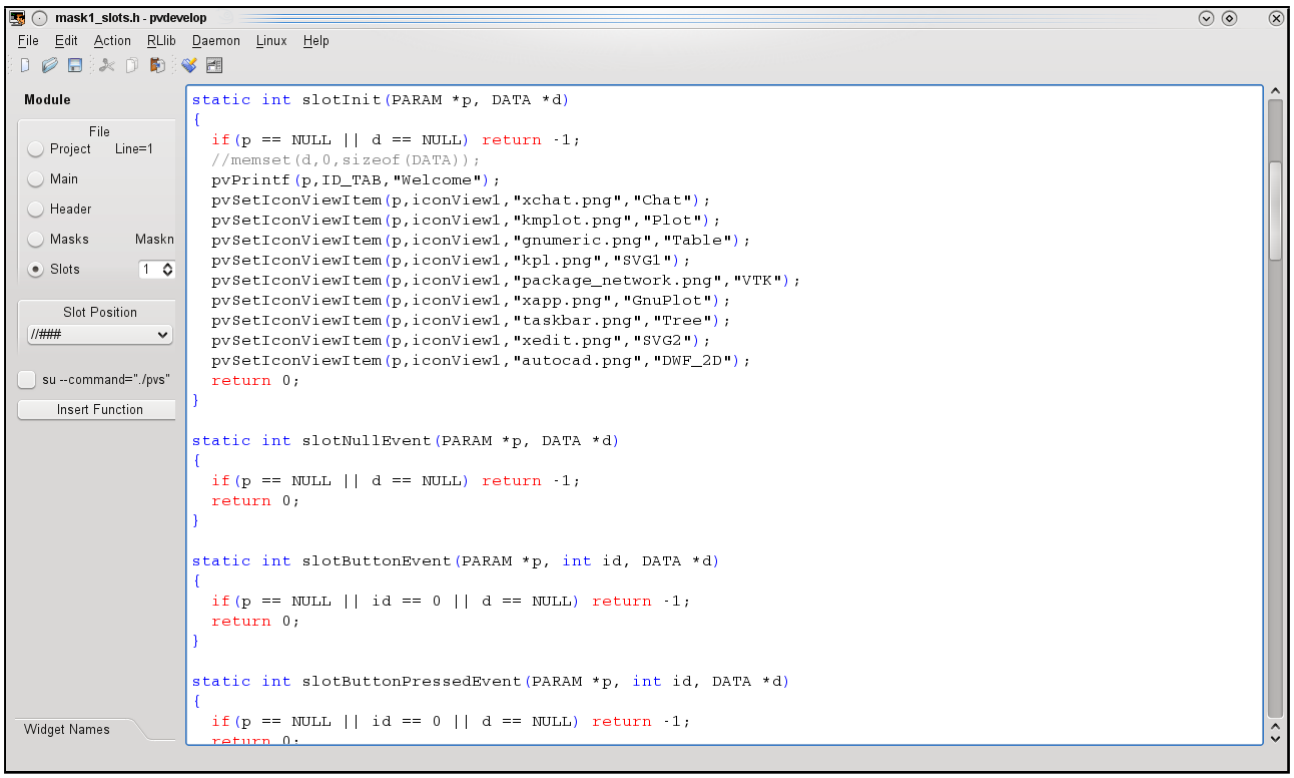

Abbildung 2: pvsexample Start-Maske in pvdevelop

Vorwort

Was ist SCADA ?

Zitat aus der Wikipedia: Automationen werden in mehrere Schichten unterteilt. Dabei ist das Level 1 die prozessnahe Schicht. Der Terminus SCADA bezieht sich gewöhnlich auf zentrale/dezentrale Systeme, die gesamte Installationen überwachen, visualisieren sowie steuern und regeln. Der größte Teil der Regelung wird automatisch durch Fernbedienungsterminals (RTU) oder durch Speicherprogrammierbare Steuerungen (SPS) beziehungsweise Level-1-Automationen durchgeführt. Die Aufgabe der Level-2-Automation ist es, die Funktion der Level-1-Automation zu optimieren, sowie Stellgrößen und Sollwerte auszugeben. Die Level-3-Automation dient hingegen der Planung, Qualitätssicherung und Dokumentation.
Die Datenerfassung beginnt gewöhnlich mit dem Level 1 und enthält die Koppelung an Messgeräte und Statusinformationen wie Schalterstellungen, die von dem SCADA-System erfasst werden. Die Daten werden dann in einer benutzerfreundlichen Darstellung präsentiert und ermöglichen es, steuernd in den Prozess einzugreifen. SCADA-Systeme implementieren typischerweise eine verteilte Datenbasis, die Datenpunkte beinhaltet. Ein Datenpunkt enthält einen Ein- oder Ausgangs- wert, der durch das System überwacht und gesteuert wird. Datenpunkte können physikalisch berechnet werden. Ein physikalischer Datenpunkt stellt einen Eingang oder Ausgang dar, während ein berechneter Punkt durch mathematische Operationen aus dem Zustand des Systems hervorgeht. Normalerweise werden Datenpunkte als eine Kombination von Werten mit Zeitstempel behandelt. Eine Serie von Datenpunkten ermöglicht die historische Auswertung.

Projektdatei für eine Visualisierung mit pvbrowser. Mittels qmake wird daraus ein Makefile erzeugt.

```
#############################################################################
# generated by pvdevelop at: Mi Nov 8 11:58:45 2006
#############################################################################

TEMPLATE = app
CONFIG += warn_on release console
CONFIG -= qt

# Input
HEADERS += pvapp.h    \
           mask11_slots.h \
           mask10_slots.h \
           mask9_slots.h \
           mask8_slots.h \
           mask7_slots.h \
           mask6_slots.h \
           mask5_slots.h \
           mask4_slots.h \
           mask3_slots.h \
           mask2_slots.h \
           mask1_slots.h
SOURCES += main.cpp   \
           mask11.cpp \
           mask10.cpp \
           mask9.cpp \
           mask8.cpp \
           mask7.cpp \
           mask6.cpp \
           mask5.cpp \
```

```
          mask4.cpp \
          mask3.cpp \
          mask2.cpp \
          mask1.cpp

!macx {
//unix:LIBS     += /usr/lib/libpvsmt.so -lpthread
unix:LIBS       += /opt/pvb/pvserver/libpvsmt.so -lpthread
#unix:LIBS      += /usr/lib/libpvsid.so
unix:INCLUDEPATH += /opt/pvb/pvserver
unix:LIBS       += /opt/pvb/rllib/lib/librllib.so
unix:INCLUDEPATH += /opt/pvb/rllib/lib
}

macx:LIBS        += /opt/pvb/pvserver/libpvsmt.a /usr/lib/libpthread.dylib
#macx:LIBS       += /opt/pvb/pvserver/libpvsid.a
macx:INCLUDEPATH += /opt/pvb/pvserver
macx:LIBS       += /usr/lib/librllib.dylib
macx:INCLUDEPATH += /opt/pvb/rllib/lib

#
# Attention:
# starting with mingw 4.8 we use mingw pthread and not our own mapping to windows threads
# you will have to adjust existing pro files
#
win32-g++ {
QMAKE_LFLAGS    += -static-libgcc
win32:LIBS      += $(PVBDIR)/win-mingw/bin/libserverlib.a
win32:LIBS      += $(PVBDIR)/win-mingw/bin/librllib.a
win32:LIBS      += -lws2_32 -ladvapi32 -lpthread
win32:INCLUDEPATH += $(PVBDIR)/pvserver
win32:INCLUDEPATH += $(PVBDIR)/rllib/lib
}

#DEFINES += USE_INETD
TARGET = pvsexample
```

Kapitel 1

Einleitung

pvbrowser ist eine SCADA Software unter GPL. Bei der GPL Lizenz müssen alle Beiträge unserem Projekt zwecks Veröffentlichung zur Verfügung gestellt werden.
pvbrowser stellt ein Framework für die Prozessvisualisierung zur Verfügung. Dies sind die Charakteristika von pvbrowser:

- Client/Server
- Qt Widgets
- Custom Widgets
- plattformunabhängig
- SVG Grafik
- xy Grafik
- 3D Grafik
- Webseiten mit WebKit
- IDE Unterstützung
- graphisches Design
- C/C++
- Lua, Python
- Multithreaded oder Inetd
- Unicode support (Chinesisch, Arabisch, Cyrillisch, ...)
- Unterstützung von ssh-urls
- Verbindung zu Feldbussen
- Verbindung zu SPS Systemen
- Kontrolle der Hintergrundprozesse
- Zentrales Event Log
- Eigene Benutzeridentifikation kann programmiert werden
- GPL Lizenz

Der pvbrowser Client kann mit einem Internet Browser verglichen werden. Es werden aber nicht im wesentlichen statische Internetseiten dargestellt, sondern dynamischer Inhalt, wie er bei SCADA Software und in anderen Bereichen notwendig ist. Mit pvbrowser können Sie beliebig viele Visualisierungen, die sich über die Anlage oder sogar das Internet verteilen über Hyperlinks browsen.
Der Rahmen des pvbrowser Fensters wird durch pvbrowser selbst festgelegt, der Inhalt des Fensters wird hingegen komplett durch den Entwickler der pvserver auf den Hintergrundrechnern festgelegt. Wenn die pvserver modifiziert werden, müssen bei den pvbrowser Clienten keine Anpassungen durchgeführt werden.
Die Benutzung des Fensters wird durch die kundenspezifische Implementierung festgelegt. Die Benutzung des Client Fensters sollte selbsterklärend sein. Mit einem Klick auf '?' neben der URL-Eingabe und einem Klick auf die Elemente der Werkzeugleiste erhält man eine kontextsensitive Hilfe. Die Hilfe im rechten Menu ist eine HTML Seite, die mit index.html beginnt. Diese Seite muss beim Starten der Visualisierung mit pvDownloadFile() geladen werden. Wenn index.html nicht existiert, ist keine Hilfe zu der kundenspezifischen Visualisierung verfügbar. Die Hilfe benutzt die von WebKit zur Verfügung gestellte Funktionalität.
pvbrowser kann mit einem Editor über Datei->Optionen angepasst werden. Achten Sie darauf, daß einige Einstellungen erst nach einem Neustart von pvbrowser Client wirksam werden.
Folgende Kommandozeilenoptionen sind verfügbar.

Kommandozeilenoptionen für pvbrowser

```
usage:  pvbrowser <-debug<=level>> <-log> <-ini=filename> <-font=name<:size>> <host<:port></mask>> <-
    disable> <-geometry=x:y:w:h> <-global_strut=width:height> <-delay=milliseconds>
    example: pvbrowser
    example: pvbrowser localhost
    example: pvbrowser localhost:5050
    example: pvbrowser -font=courier localhost
    example: pvbrowser -font=arial:14 localhost:5050 -disable
    example: pvbrowser -geometry=0:0:640:480
    example: pvbrowser -global_strut=50:50 # set minimum size for embedded systems
```

Der pvbrowser Client wird sich über TCP/IP mit einem pvserver verbinden. Der pvserver wird daraufhin ASCII Text zum pvbrowser Client schicken, der dort interpretiert und in Aufrufe der Qt Bibliothek umgesetzt wird. In umgekehrter Richtung wird der pvbrowser Client ASCII Text zum pvserver schicken, wenn der Benutzer Ereignisse auslöst. Das könnte z.B. das Drücken eines Buttons sein. Im pvserver ist eine Ereignisschleife enthalten in der dieser ASCII Text interpretiert wird und eine entsprechende 'slot Funktion' aufgerufen wird. Die Aufgabe des Entwicklers einer Visualisierung ist es nun die 'slot Funktionen' auszufüllen/programmieren. Das gesamte Gerüst des Servers wird aber mit unserer Entwicklungsumgebung pvdevelop erzeugt und verwaltet. Der Entwickler muss nur noch die 'slot Funktionen' betrachten. Das Layout der Masken wird graphisch in pvdevelop eingegeben.

Eine typische Funktion von pvserver, die einen ASCII Text an den pvbrowser Client schickt.

```
int pvSetValue(PARAM *p, int id, int value)
{
char buf[80];

  sprintf(buf,"setValue(%d,%d)\n",id,value);
  pvtcpsend(p, buf, strlen(buf));
  return 0;
}
```

Wie man an der Obigen Funktion erkennen kann, braucht der Entwickler der Visualisierung die ASCII Texte nicht kennen. Es steht Ihm eine Bibliothek zur Verfügung, die das kapselt. Der Parameter PARAM ist in all diesen Funktionen als erster Parameter enthalten und beschreibt die Verbindung zum pvbrowser Client. Im Besonderen ist darin der Socket gespeichert, der die TCP/IP Verbindung zum Client enthält.
pvbrowser arbeitet also mit einem eigenen Protokoll pv:// , unterstützt aber darüber hinaus auch das http:// Protokoll und kann damit normale Webseiten darstellen. Das wird über das integrierte WebKit realisiert. Zwischen normalen Internetseiten, pvservern und umgekehrt kann über Hyperlinks umgeschaltet werden. Internetseiten können ebenfalls auch als Teil einer pvbrowser Maske eingebunden werden.
Die Anbindung von pvserver an SPS oder Feldbussysteme ist über eine Vielzahl von Protokollen möglich, wobei auch ein gemischter Betrieb verschiedener Protokolle möglich ist.

Kapitel 2

Installation

pvbrowser ist unter Linux / Unix / Windows und Mac OS-X lauffähig. Die pvserver können darüber hinaus auch unter OpenVMS laufen.

Die Quellen von pvbrowser sind in dem Archiv `http://pvbrowser.de/pvbrowser/tar/pvb.tar.gz` enthalten. Laden Sie dieses Archiv herunter, wenn Sie pvbrowser selbst übersetzen möchten. Außerdem sind über unsere Homepage binäre Pakete für Windows und OS-X verfügbar. Binäre Pakete für diverse Linux Distributionen sind über den openSUSE Buildservice verfügbar. Auf den Buildservice wird im Download Bereich unserer Homepage verwiesen.

Zusätzlich gibt es das Paket `http://pvbrowser.de/pvbrowser/tar/pvbaddon.tar.gz` , in dem einige Demos und Vorlagen zu pvbrowser enthalten sind. Insbesondere sind darin Beispiele zur Datenerfassung über diverse Protokolle enthalten.

Auf die Installation der Binärpakete wird hier nicht näher eingegangen, da das nach dem Standardverfahren Ihres Betriebssystems gemacht wird. Es ist aber zu beachten, dass die Entwicklungsumgebung pvdevelop einige weitere Pakete erfordert. Der pvbrowser Client ist auch ohne diese Entwicklungswerkzeuge lauffähig. Allerdings ist unter OS-X zu beachten, dass die Qt Laufzeit Bibliotheken vorher installiert sein müssen. Unter Windows werden diese Bibliotheken von unserem Paket mitgeliefert. Unter Linux sollten diese Bibliotheken bereits standardmäßig installiert sein. Sie können aus den folgenden Boxen ersehen, welche Entwicklungswerkzeuge für Ihr Betriebssystem erforderlich sind.

Bauen und Testen der Software unter Linux und OS-X

```
#
# Linux:
#   Die Pakete qt4-devel inklusive WebKit , make und gcc muessen installiert sein
# OS-X:
#   xcode, X11User und das Qt SDK muessen installiert sein
#
wget http://pvbrowser.org/tar/pvb.tar.gz
tar -zxf pvb.tar.gz
cd pvb
./clean.sh
./build.sh
su
./install.sh
exit
pvbrowser pv://pvbrowser.de
```

build.sh enthält eine Abfrage, auf welchem Betriebssystem Sie bauen möchten. Sie müssen build.sh zunächst editieren und entsprechend den darin enthaltenen Anweisungen anpassen, damit die Qt Bibliotheken gefunden werden können.

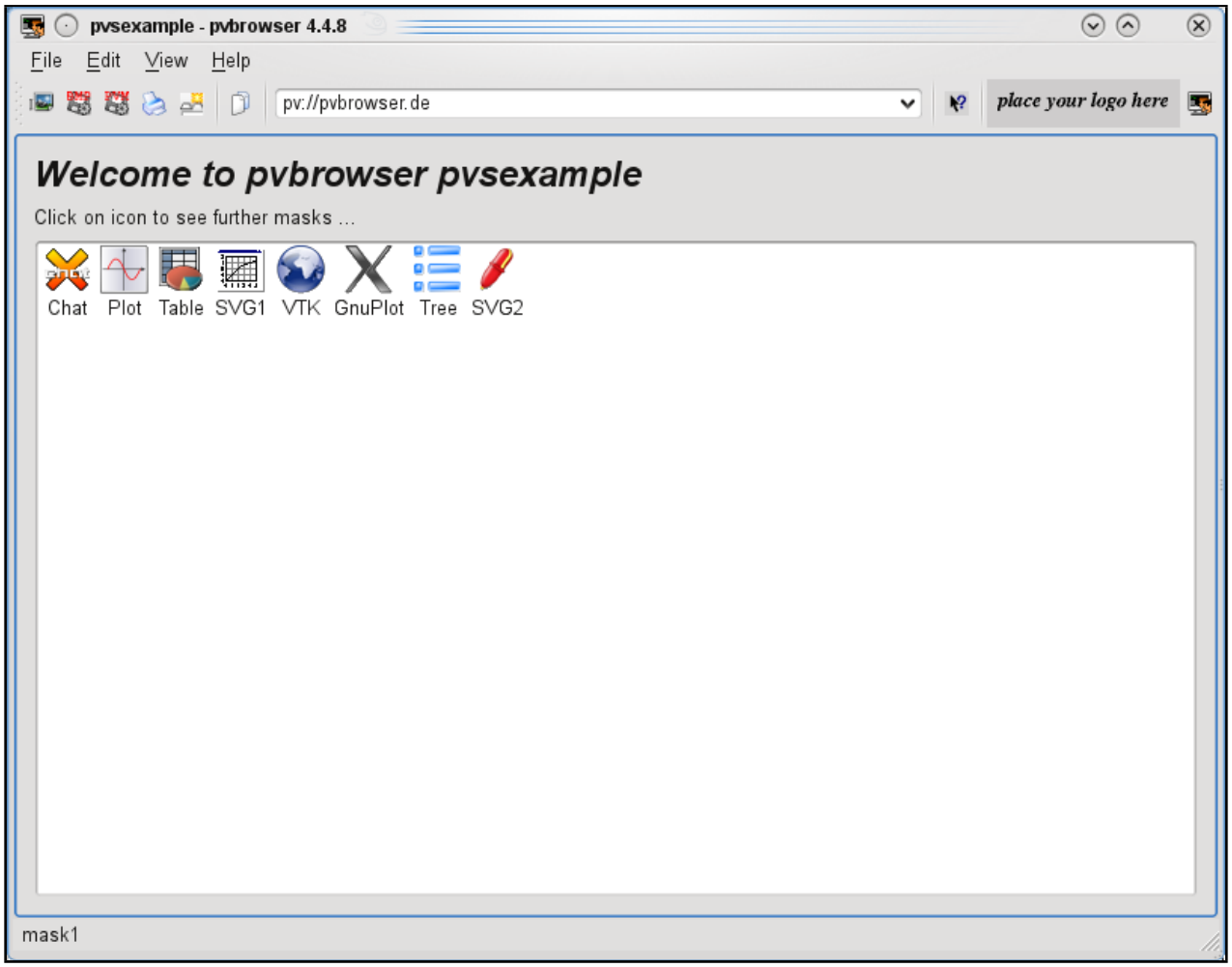

Abbildung 2.1: Der pvbrowser nach dem Aufruf von 'pvbrowser pv://pvbrowser.de'

Bauen und Testen unter Windows

```
#
# Das Qt SDK fuer Windows und MinGW muessen installiert sein
# Das Qt SDK fragt bei der Installation, ob MinGW ebenfalls installiert werden soll
# Die Environment Variablen QTDIR und MINGWDIR muessen korrekt gesetzt sein,
# damit diese Werkzeuge gefunden werden koennen.
#
http://pvbrowser.org/tar/pvb.tar.gz herunterladen und auspacken
DOS-Box oeffnen und in das Verzeichnis von pvb gehen.
cd win-ming
CreatePvbWithMinGW.bat
cd bin
pvbrowser pv://pvbrowser.de
```

Bauen der pvserver Bibliotheken unter OpenVMS

```
#
# Der CXX C++ Compiler muss installiert sein
#
http://pvbrowser.org/tar/pvb.tar.gz herunterladen und auspacken
In das Verzeichnis von pvb gehen.
@vms_build.com
```

Kapitel 3

pvbrowser Client

Der pvbrowser Client arbeitet im Prinzip wie ein Webbrowser, nur dass er neben dem http:// Protokoll hauptsächlich die eigenen Protokolle pv:// und pvssh:// benutzt. Die eigenen Protokolle sind verbindungsorientiert, d.h. die Verbindung zum Server bleibt so lange bestehen, bis die Sitzung beendet wird. Das http:// Protokoll ist hingegen verbindungslos, d.h. die Verbindung wird nach dem Abholen jeder Webseite beendet und erst nach dem Klicken eines Links wieder neu aufgebaut. Für die Zwecke der Prozessvisualisierung ist ein verbindungsorientiertes Protokoll besser geeignet, weil eine bessere Authentifikation möglich ist und der Overhead des http:// Protokolls entfällt.

Genau wie bei einem Webbrowser besitzt der pvbrowser Client eine Adresszeile, in die man eine URL (Internetadresse) eingeben kann. pvbrowser unterstützt die 3 oben genannten Protokolle. Das http:// Protokoll wird dabei durch das in Qt integrierte WebKit implementiert. Zwischen den einzelnen Protokollen kann über Hyperlinks von einer Seite zu einer anderen Seite verlinkt werden.

Verlinkung zwischen Masken und Webseiten

```
pvHyperlink(p,"pv://pvbrowser.de"); // Verlinkung aus einem pvserver auf unseren Testserver
pvHyperlink(p,"pvssh://benutzer@pvbrowser.de"); // Verlinkung ueber secure shell
pvHyperlink(p,"http://google.de"); // Verlinkung aus einem pvserver auf eine Suchmaschine

<a href="pv://pvbrowser.de">Verlinkung zu dem pvserver auf unseren Testserver</a>
<a href="pvssh://benutzer@pvbrowser.de">Verlinkung zu einem pvserver ueber secure shell</a>
```

Die Bedienung des Clients sollte selbsterklärend sein. Es sei jedoch auf die Einstellungsmöglichkeiten innerhalb des Clients hingewiesen. Die Einstellungen werden in einer Datei gespeichert, die im HOME Verzeichnis des Benutzers liegt. Über das Datei Menu gelangt man zu diesen Optionen und kann diese editieren. Beachten Sie, dass sich einige der Änderungen erst nach einem Neustart des Clients auswirken.

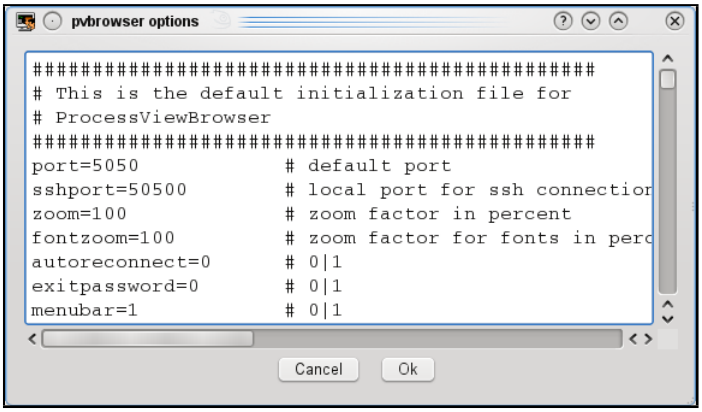

Abbildung 3.1: Dialog für die pvbrowser Optionen

Einstellungsmöglichkeiten für den pvbrowser Client

```
port=5050                    # Standard port
sshport=50500                # Lokaler Port fuer ssh Verbindungen
zoom=100                     # Zoom Faktor in Prozent
fontzoom=100                 # Zoom Faktor fuer Schriften in Prozent
autoreconnect=0              # 0|1 automatisches Neu verbinden, bei Verbindungsabbruch
exitpassword=0               # 0|1 gibt an, ob der Benutzer ein Passwort benoetigt,
                             # um den Browser zu verlassen
menubar=1                    # 0|1 schaltet die Menuzeile Ein/Aus
toolbar=1                    # 0|1 schaltet die Werkzeugleiste Ein/Aus
statusbar=1                  # 0|1 schaltet die Statuszeile Ein/Aus
scrollbars=1                 # 0|1 schaltet den Scroll bar Ein/Aus
fullscreen=0                 # 0|1 kann auf Vollbildmodus schalten
maximized=0                  # 0|1 das Fenster kann beim Start maximiert werden
cookies=1                    # 0=No 1=Yes 2=Ask Behandlung von Cookies
echo_table_updates=0         # 0|1 gibt an, ob auch bei der serverseitigen Aenderung
                             # von Tabelleninhalten Ereignisse generiert werden sollen
temp=/tmp                    # Verzeichnis, in dem temporaere Dateien von pvbrowser gespeichert
                             # werden sollen
customlogo=/opt/pvb/custom.png # durch den Benutzer definierbares Logo im Kopf des Fensters
newwindow=pvbrowser          # Befehl zum Starten weiterer Fenster aus pvbrowser
ssh=ssh                      # Dienstprogramm fuer secure shell. Unter Windows putty.exe
initialhost=pv://localhost   # Startseite mit der verbunden werden soll
# Der pvserver kann einige Dateien in das temporaere Verzeichnis auf dem Client laden und
# pvbrowser anweisen, diese mit einem bestimmten Programm zu oeffnen
# Dies kann nuetzlich sein, um auf dem Client Protokolle zu drucken oder eine Tabellen-
# Kalkulation mit einer CSV Datei aufzurufen, in der der Benutzer eigene Auswertungen
# machen kann.
view.pdf=okular              # Programm fuer PDF
view.img=gimp                # Programm fuer Bitmap Bilder
view.svg=inkscape            # Programm fuer SVG Bilder
view.txt=kwrite              # Programm fuer Text Dateien
view.csv=ooffice             # Programm fuer CSV Dateien (unter Windows auch Excel)
view.html=firefox            # Webbrowser
# Ausserdem folgt noch die Spracheinstellung
# Momentan sind darin Englisch, Deutsch, Franzoesisch und Spanisch enthalten
# Es koennen aber beliebige (auch nicht lateinische) Sprachen hinzugefuegt werden
```

URL's in pvbrowser

```
pv://server:portnummer/maske?parameter

Beispiele:
pv://pvbrowser.de
pv://pvbrowser.de:5050
pv://192.168.1.14:6000
pv://localhost/maske1?testwert=1
pvssh://benutzer@pvbrowser.de
http://www.google.de
```

Kapitel 4

pvdevelop IDE

pvdevelop ist eine IDE zur Entwicklung von pvservern unter der Benutzung von C C++ bzw. Lua oder Python. Mit pvdevelop muss man sich nicht um die Projektdatei, den Makefile und das Gerüst eines pvserver kümmern. Die Projektdatei, der Makefile und das Gerüst des pvserver wird automatisch generiert und aktualisiert. Die Bestandteile des pvserver kann man einfach browsen, editieren und die Masken der Visualisierung graphisch eingeben.
Im Editor Modus können Sie die Projektdatei und die Quelltexte einsehen und editieren. Auf der linken Seite des Fensters sehen Sie eine Tool box. Darin können Sie zwischen den verschiedenen Dateien, aus denen Ihr pvserver besteht umschalten. Wenn Sie auf 'Insert Function' klicken erscheint ein Auswahlbaum, aus dem Sie die Funktionen für den pvserver auswählen können. Beachten Sie, dass der Cursor im Textfenster vorher an der richtigen Stelle positioniert sein muss.
Wenn Sie die Tool box auf 'Widget Names' umschalten, sehen Sie eine Liste der von Ihnen entworfenen Widgets. Daraus können Sie auswählen und die am meisten benutzten Funktionen direkt einsetzen.
Im Designer Modus können Sie Ihre Widgets graphisch eingeben. Wählen Sie über das mit der rechten Maustaste erreichbare Popup Menu, die entsprechenden Funktionen aus. Nachdem ein Widget eingegeben wurde, können Sie es im Designer mit Hilfe der Maus Positionieren und die Größe verändern (rechte untere Ecke des Widgets). Über den Eigenschaften Dialog (Widget anklicken, rechte Maustaste) können Sie das Widget parametrieren.
Im Designer Modus wird die Maus eingefangen (grab mouse/release mouse). Wenn Sie 'release mouse' wählen, können Sie das Design testen. Es verhält sich dann, wie in der fertigen Anwendung. Zum weiteren Design schalten Sie dann wieder auf 'grab mouse' um.
Sie können in der Datei projektname.pvproject einstellen, in welchem Raster die Widgets positioniert werden können.
Falls Sie lieber mit Hilfe des Qt Designer arbeiten möchten, können Sie die Import/Export Funktion für UI Dateien aus dem 'Action' Menu im Editor Modus benutzen.
Die Entwicklung kann auch in anderen Entwicklungsumgebungen wie Eclipse durchgeführt werden. Besonders gut geeignet ist aber der Qt Creator von Nokia/Trolltech, da dieser, genau wie pvdevelop von hause aus mit qmake Projektdateien arbeitet.
Sie können sich voll auf die Programmierung der 'slot Funktionen' konzentrieren. Der Rest des Quelltextes, aus dem der pvserver besteht, wird vollständig von pvdevelop generiert und aktualisiert.
Da zum Übersetzen des pvserver ein Makefile generiert wird, kann man aber auch beliebige ASCII Editoren zur Entwicklung nutzen und mit make aus der Kommandozeile übersetzen.

Übersetzen aus der Kommandozeile

```
benutzer@rechner:~/pvb/pvsexample> make
make: Fuer das Ziel 'first' ist nichts zu tun.
benutzer@rechner:~/pvb/pvsexample>
```

Im Hilfe Menu von pvdevelop gelangen Sie zum Referenzmanual für die Bibliotheken zur Ansteuerung des pvbrowser und zur rllib für serverseitige Programmierung.

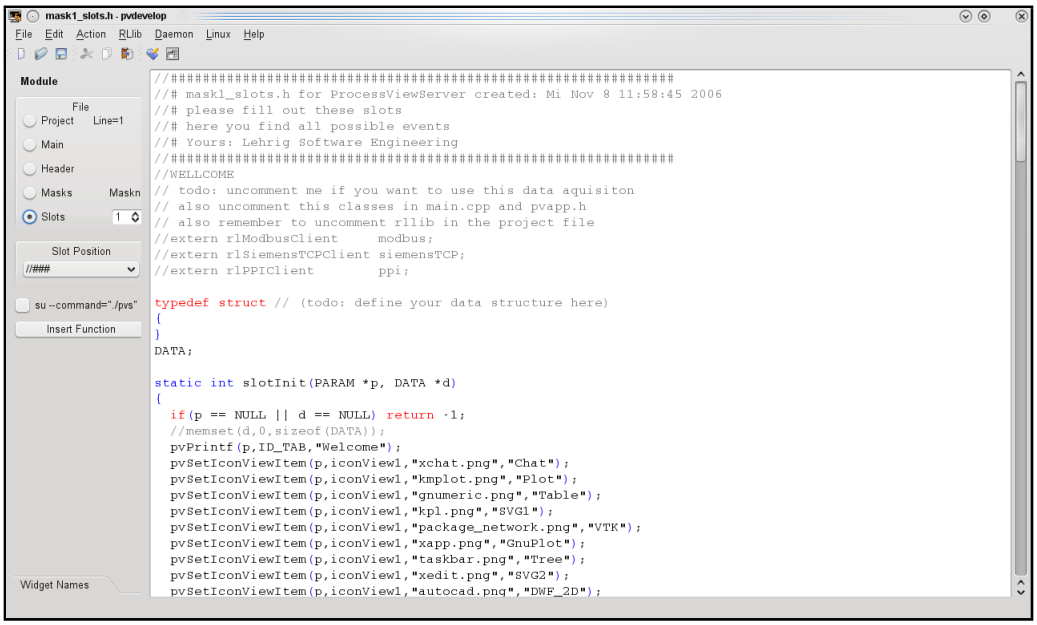

Abbildung 4.1: Das Hauptfenster von pvdevelop im Editor Modus

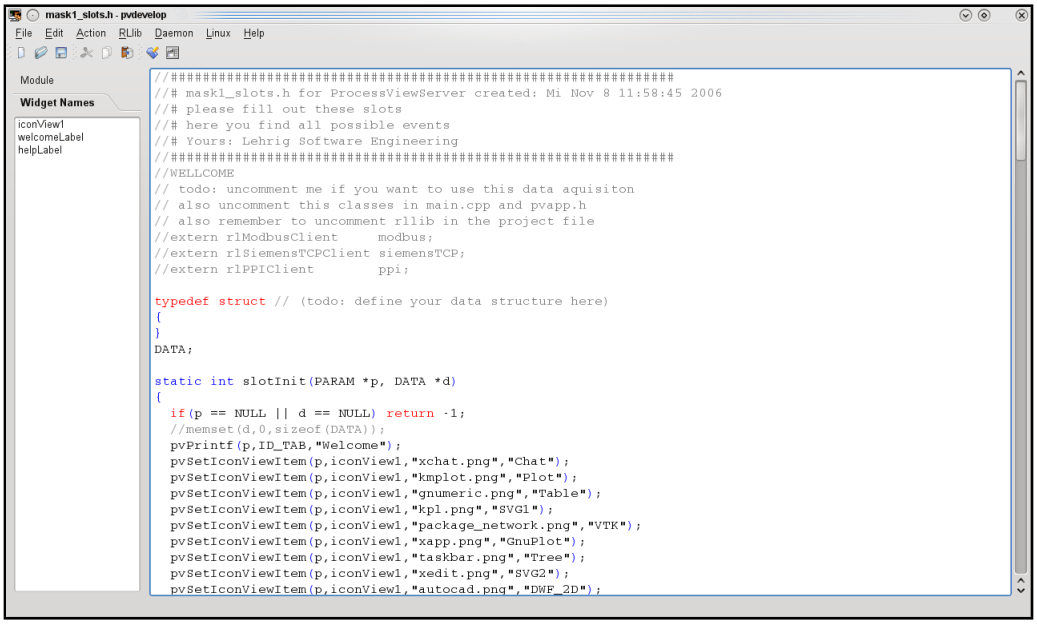

Abbildung 4.2: Das Hauptfenster von pvdevelop im Editor Modus mit ausgewählter Tool box

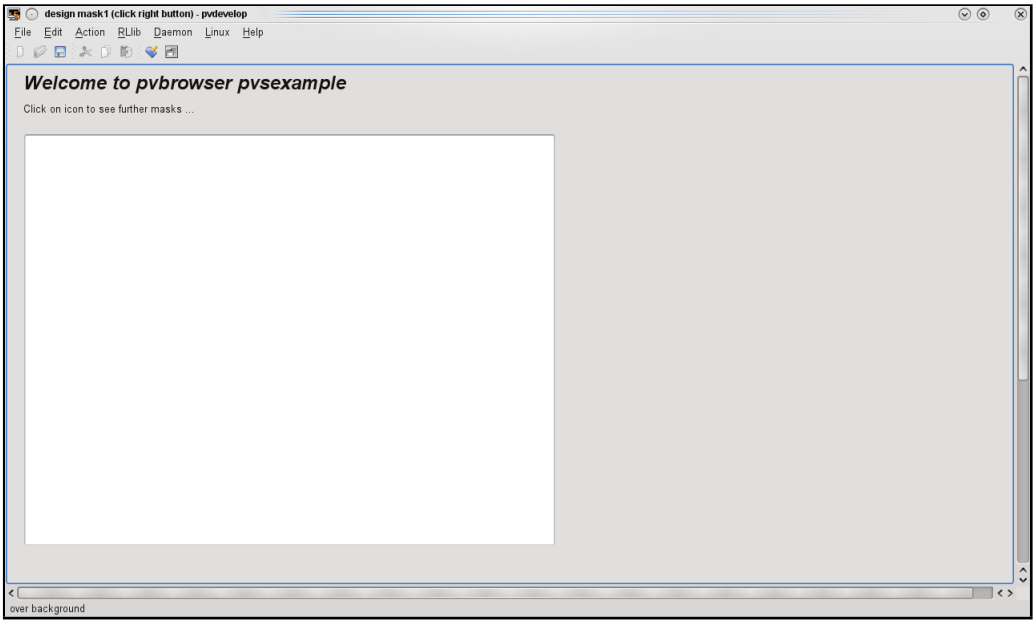

Abbildung 4.3: Das Hauptfenster von pvdevelop im Designer Modus

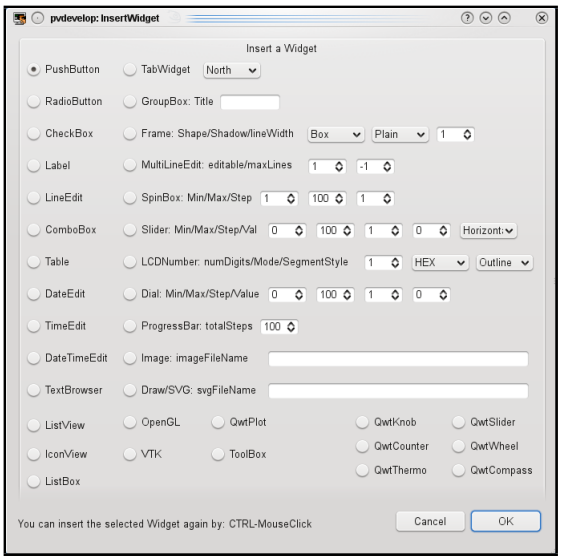

Abbildung 4.4: Dialog box zum Einfügen von Widgets

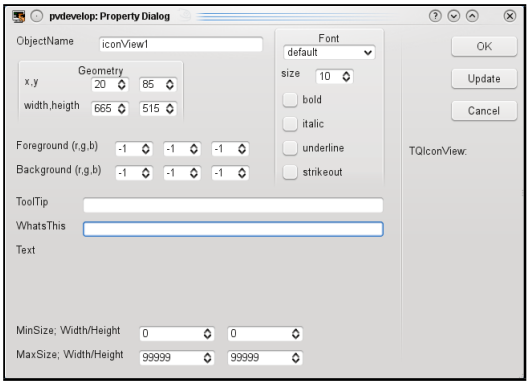

Abbildung 4.5: Dialog box für Widget Eigenschaften

Kapitel 5

Qt Creator als IDE verwenden

Qt Creator ist hervorragend geeignet, um pvserver zu erstellen, da dort die pro Dateien für qmake auch Verwendung finden.

Ein paar nützliche Shell Scripte Legen Sie sich unter dem Verzeichnis '~/bin' ein Paar Shell Scripte an, die Sie unterstützen.
Das erste Shell Script legt ein neues Projekt an und startet Qt Creator.

~/bin/pvnew

```
#!/bin/bash
pvdevelop -action=writeInitialProject
pvdevelop -action=uncommentRLLIB
pvdevelop -action=make
qtcreator pvs.pro &
```

Das zweite Shell Script importiert eine Maske 'maskN.ui' in den pvserver.

~/bin/pvimport

```
#!/bin/bash
if [ "x${1}" = "x" ]; then
  echo "usage: pvimport <masknumber>"
  exit
fi
pvdevelop -action=importUi:${1}
```

Das dritte Shell Script fügt eine neue Maske in ein bestehendes Projekt ein.

~/bin/pvinsert

```
#!/bin/bash
pvdevelop -action=insertMask
```

Legen Sie nun ein neues Verzeichnis an und geben den Befehl 'pvnew' ein. Damit wird ein neuer pvserver angelegt und Qt Creator gestartet.

Einen neuen pvserver anlegen und Qt Creator starten

```
me@mylinux:~/temp> mkdir pvserver
me@mylinux:~/temp> cd pvserver/
me@mylinux:~/temp/pvserver> pvnew
could not open pvs.pvproject
g++ -c -m64 -pipe -O2 -fmessage-length=0 -O2 -Wall -D_FORTIFY_SOURCE=2 -fstack-protector -funwind-
    tables -fasynchronous-unwind-tables -g -Wall -W -I/usr/share/qt4/mkspecs/default -I. -I/opt/pvb/
    pvserver -o main.o main.cpp
g++ -c -m64 -pipe -O2 -fmessage-length=0 -O2 -Wall -D_FORTIFY_SOURCE=2 -fstack-protector -funwind-
    tables -fasynchronous-unwind-tables -g -Wall -W -I/usr/share/qt4/mkspecs/default -I. -I/opt/pvb/
    pvserver -o mask1.o mask1.cpp
g++ -m64 -Wl,-O1 -o pvs main.o mask1.o /usr/lib/libpvsmt.so -pthread
me@mylinux:~/temp/pvserver>
```

In Qt Creator akzetieren Sie nun das Importieren der bestehenden Einstellungen.

Bei 'Projekte->Ausführung->Details zeigen' selektieren Sie bitte 'im Terminal ausführen', da die Ausgaben die der pvserver in der Konsole macht, sonst nicht verfolgt werden könnten. Diese Ausgaben sind aber eine wichtige Hilfe beim Debugging. Unter Windows sind die Schritte ähnlich, man muss die Shell Scripte nur durch entsprechende Batch Dateien ersetzen.

Plugins für Qt Designer Damit Sie die speziellen Widgets von pvbrowser in Qt Designer eingeben können, müssen Sie die Plugins in das Qt Designer Plugin Verzeichnis kopieren.

Das können Sie mit einem der folgenden Befehle machen, je nachdem, ob Sie ein 32 oder 64 Bit Linux verwenden.

Plugins kopieren

```
cp /opt/pvb/designer/plugins/* /usr/lib/qt4/plugins/
cp /opt/pvb/designer/plugins/* /usr/lib64/qt4/plugins/
Windows: %PVBDIR%/win-mingw/bin/plugins/designer/*.dll in Qt Designer Plugins Verzeichnis kopieren.
```

Hilfe Dateien für Qt Creator Unter `http://pvbrowser.org/pvbrowser/download.php?file=pvslib.qch` und `http://pvbrowser.org/pvbrowser/download.php?file=rllib.qch` finden Sie Hilfe Dateien für unsere Bibliotheken, die Sie aus Qt Creator ansprechen können.

pvdevelop als externes Werkzeug aus Qt Creator heraus nutzen In Qt Creator kann pvdevelop als externes Werkzeug definiert werden. Siehe die folgenden Bildschirmkopien, um die notwendigen Parameter ablesen zu können.

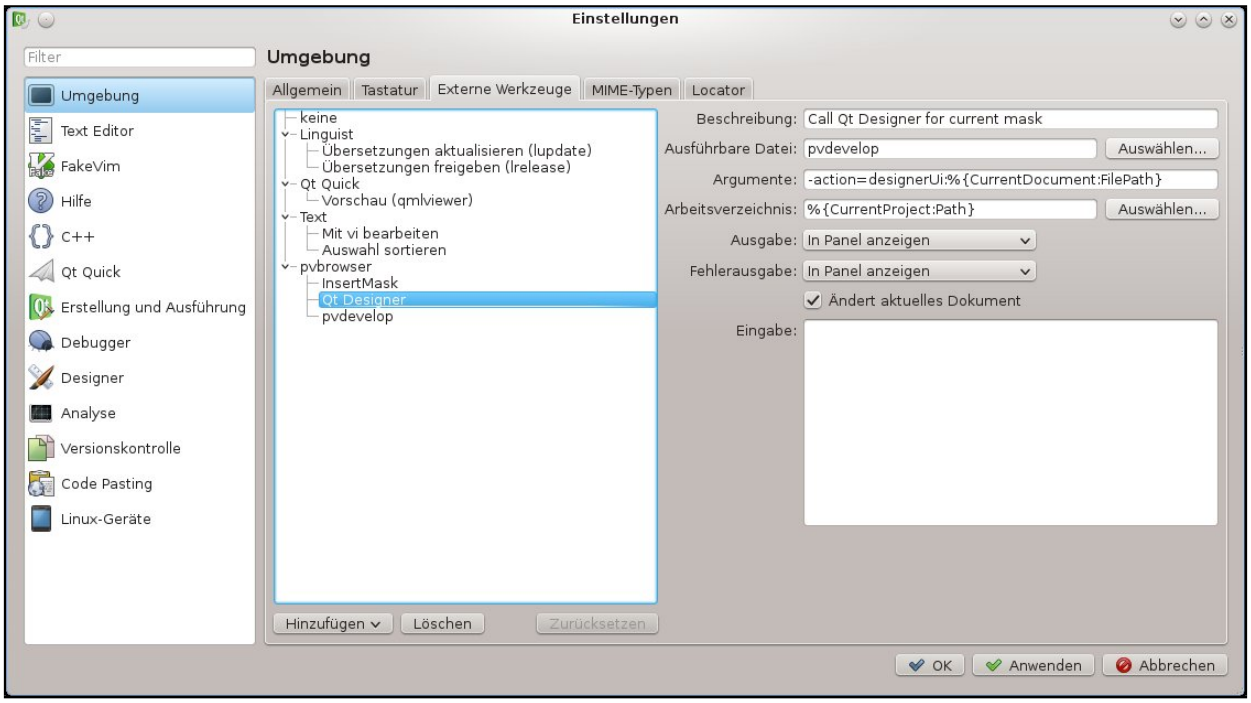

Abbildung 5.1: Aufruf von Qt Designer. In Qt Creator öffnen Sie bitte zunächst die entsprechende maskX.cpp oder maskX_slots.h Datei und selektieren dieses Menu. Nach dessen Aufruf kann man die entsprechende Maske in Qt Designer eingeben. pvdevelop exportiert/importiert die zur Maske gehörenden Definitionen aus der entsprechenden maskX.ui Datei.

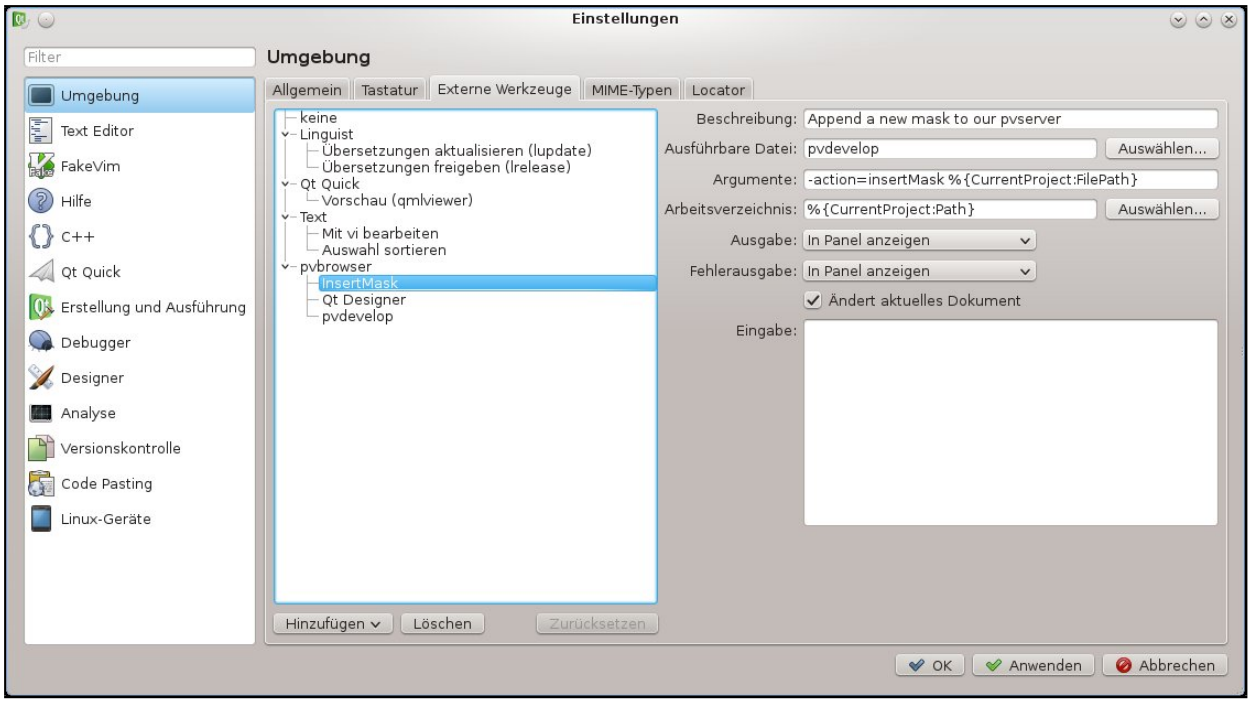

Abbildung 5.2: Mit den obigen Parametern lässt sich pvdevelop als externes Werkzeug für das Hinzufügen von neuen Masken verwenden.

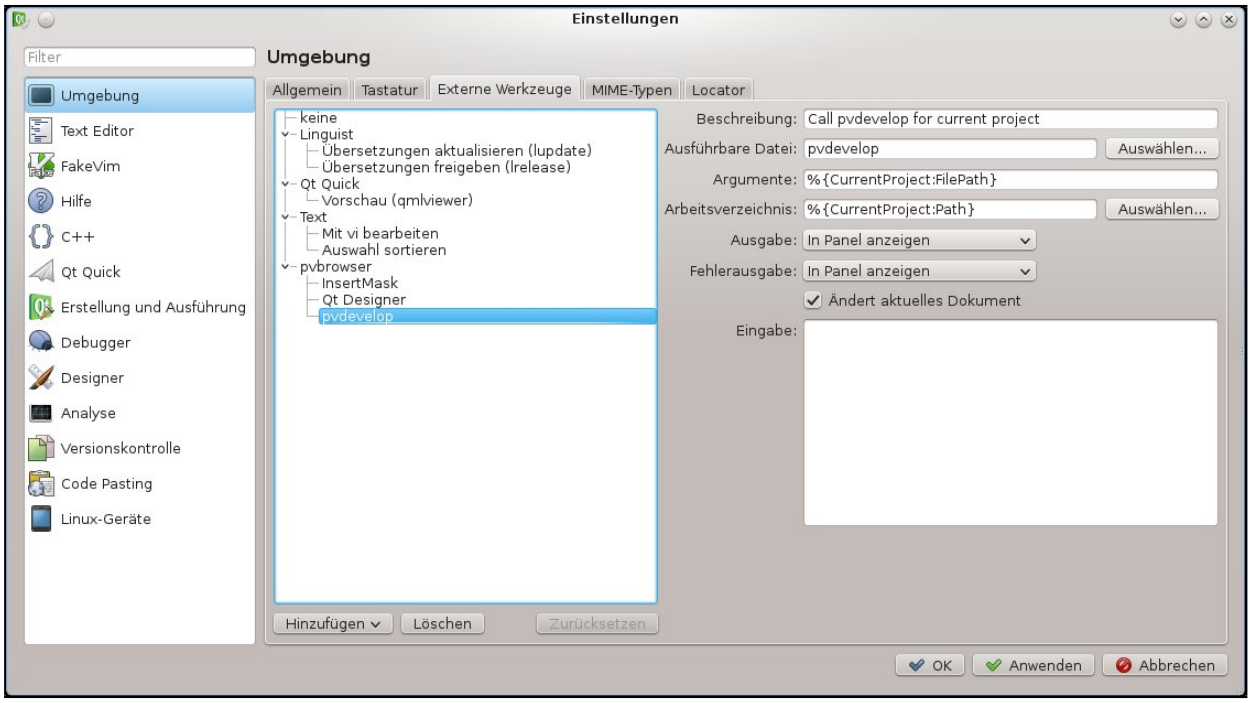

Abbildung 5.3: pvdevelop kann auch mit vollen graphischen Interface aufgerufen werden.

Das Layout der Maske kann nicht in Qt Creator eingegeben werden. Verwenden Sie hierfür den Editor oder pvdevelop.

Kapitel 6

Programmierung

Wenn Sie 'File->new pvserver' in pvdevelop wählen, werden Sie aufgefordert, den Namen und den Speicherort für ihren pvserver anzugeben. Daraufhin wird der integrierte Designer angezeigt. Die Bedienung erfolgt über das Popup Menu, erreichbar über die rechte Maustaste. Geben Sie das Layout mit dem graphischen Designer ein. Nun Speichern Sie das Layout und wechseln in den Editor Modus.
Der neue pvserver wird automatisch generiert. Bitte sehen Sie sich den Quelltext aus pvdevelop heraus an, um seine Struktur zu verstehen.
Nachdem Sie diese Initialen Schritte durchgeführt haben, können Sie die ursprüngliche Visualisierung starten. 'Action->start server' und 'Action->start browser' sind die entsprechenden Befehle.
Ihre Aufgabe ist es, die Datenstruktur DATA und mask∗_slots.h zu kodieren, um die Logik Ihres pvserver festzulegen.

6.1 Aufbau eines pvserver

Es gibt eine Projektdatei, die den verwendeten Quelltext und die Bibliotheken definiert. Es gibt eine main Datei. In dieser Datei kann man die unterschiedlichen main() Funktionen für INETD und MULTI THREADED pvserver sehen, die über ein #ifdef Präprozessorsymbol unterschieden werden. Standardmäßig wird der MULTI THREADED Server genommen. Wenn Sie den pvserver lieber über den Superserver xinetd starten möchten, müssen Sie in der Projektdatei das USE_INETD auskommentieren und die 'libpvsid' anstatt der 'libpvsmt' Variante der pvbrowser Bibliothek auswählen.
pvMain ist die Hauptroutine zur Behandlung eines Clienten. In diesem Unterprogramm sieht man eine Schleife, in der die verfügbaren Masken aufgerufen werden. Der Rückgabewert einer Maske legt fest, welche Maske als nächstes aufgerufen werden soll.
Es gibt eine Header Datei, die in alle Quelltexte des pvserver aufgenommen wird.
Es gibt so viele Masken, wie sie festlegen. Der Quelltext wird automatisch generiert. Normalerweise braucht man sich nicht darum zu kümmern.
Es gibt so viele mask_slots, wie Sie Masken definieren. Diese Header Datei wird in der zugehörigen Maske inkludiert. Dort wird die lokale Datenstruktur DATA und die slot Unterprogramme definiert.

6.1.1 Projekt Datei für qmake

Aus der Projektdatei wird mit Hilfe des aus dem Qt SDK stammenden qmake ein Makefile generiert. In der Projektdatei werden alle Quelltexte und Bibliotheken spezifiziert, aus denen Ihr pvserver besteht. Beachten Sie, dass die Projektdatei automatisch die Bibliotheken auswählt, die für das von Ihnen verwendete Betriebssystem relevant sind. Falls Sie weitere Quelltexte oder Bibliotheken in Ihrem pvserver verwenden möchten, können Sie das in der Projektdatei angeben.

Projektdatei

```
#############################################################################
# generated by pvdevelop at: Mi Nov 8 11:58:45 2006
#############################################################################

TEMPLATE = app
CONFIG += warn_on release console
```

```
CONFIG -= qt

# Input
HEADERS += pvapp.h    \
          mask11_slots.h \
          mask10_slots.h \
          mask9_slots.h \
          mask8_slots.h \
          mask7_slots.h \
          mask6_slots.h \
          mask5_slots.h \
          mask4_slots.h \
          mask3_slots.h \
          mask2_slots.h \
          mask1_slots.h
SOURCES += main.cpp   \
          mask11.cpp \
          mask10.cpp \
          mask9.cpp \
          mask8.cpp \
          mask7.cpp \
          mask6.cpp \
          mask5.cpp \
          mask4.cpp \
          mask3.cpp \
          mask2.cpp \
          mask1.cpp

!macx {
//unix:LIBS     += /usr/lib/libpvsmt.so -lpthread
unix:LIBS       += /opt/pvb/pvserver/libpvsmt.so -lpthread
#unix:LIBS      += /usr/lib/libpvsid.so
unix:INCLUDEPATH += /opt/pvb/pvserver
unix:LIBS       += /opt/pvb/rllib/lib/librllib.so
unix:INCLUDEPATH += /opt/pvb/rllib/lib
}

macx:LIBS        += /opt/pvb/pvserver/libpvsmt.a /usr/lib/libpthread.dylib
#macx:LIBS       += /opt/pvb/pvserver/libpvsid.a
macx:INCLUDEPATH += /opt/pvb/pvserver
macx:LIBS       += /usr/lib/librllib.dylib
macx:INCLUDEPATH += /opt/pvb/rllib/lib

#
# Attention:
# starting with mingw 4.8 we use mingw pthread and not our own mapping to windows threads
# you will have to adjust existing pro files
#
win32-g++ {
QMAKE_LFLAGS    += -static-libgcc
win32:LIBS      += $(PVBDIR)/win-mingw/bin/libserverlib.a
win32:LIBS      += $(PVBDIR)/win-mingw/bin/librllib.a
win32:LIBS      += -lws2_32 -ladvapi32 -lpthread
win32:INCLUDEPATH += $(PVBDIR)/pvserver
win32:INCLUDEPATH += $(PVBDIR)/rllib/lib
}

#DEFINES += USE_INETD
TARGET = pvsexample
```

6.1.2 Main Funktion

Die main() Funktion unterscheidet sich, je nachdem ob man die MULTI THEADED Variante oder die INETD Variante auswählt. Das wird über das Präprozessorsymbol USE_INETD unterschieden.
Wenn sich ein neuer Client mit pvserver verbindet wird in main() entweder ein neuer Thread kreiert und daraus dann pvMain() aufgerufen, um den Client zu bedienen oder im Falle von USE_INETD wird pvMain() direkt aufgerufen.
In pvMain() kann man zunächst die vom Client verwendete URL interpretieren. Dann geht pvMain() in eine Endlosschleife in der die verschiedenen Masken aufgerufen werden, die in pvserver definiert worden sind. Der Return Wert einer Maske gibt dann an, welche Maske als nächstes aufgerufen werden soll. Wenn der Client die Verbindung beendet, wird der Thread mit pvMain() automatisch beendet. Es ist möglich in der Datenstruktur PARAM einen Zeiger auf eine cleanup() Routine zu setzen, falls Sie selber noch Ressourcen freigeben müssen, wenn der pvserver beendet wird. Im Normalfall ist das aber nicht notwendig.
Die im gezeigten Quelltext enthaltene Funktion getParams() wurde vom Entwickler von pvsexample eingeführt und ist nicht generiert worden. Das restliche Gerüst wurde aber ursprünglich von pvdevelop generiert.

Main Funktion

```
//****************************************************************************
//                        main.cpp -  description
//                           -------------------
// begin          : Mi Nov 8 11:58:45 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);

static int getParams(const char *ini, POPUP_DATA *popup)
{
  const char *cptr;

  cptr = strstr(ini, "popup=");
  if (cptr != NULL)
  {
    if (cptr[6] == 't') popup->popup = true;
    else                popup->popup = false;
  }

  cptr = strstr(ini, "x1=");
  if (cptr != NULL)
  {
    sscanf(cptr, "x1=%f", &popup->x1);
  }

  cptr = strstr(ini, "y1=");
  if (cptr != NULL)
  {
    sscanf(cptr, "y1=%f", &popup->y1);
  }

  cptr = strstr(ini, "x0=");
  if (cptr != NULL)
  {
    sscanf(cptr, "x0=%f", &popup->x0);
  }

  cptr = strstr(ini, "y0=");
```

```
  if (cptr != NULL)
  {
    sscanf(cptr, "y0=%f", &popup->y0);
  }

  cptr = strstr(ini, "scale=");
  if (cptr != NULL)
  {
    sscanf(cptr, "scale=%f", &popup->scale);
    printf("scale main:%f\n", popup->scale);
  }

  cptr = strstr(ini, "svgx0=");
  if (cptr != NULL)
  {
    sscanf(cptr, "svgx0=%f", &popup->svgx0);
  }

  cptr = strstr(ini, "svgy0=");
  if (cptr != NULL)
  {
    sscanf(cptr, "svgy0=%f", &popup->svgy0);
  }

  return 0;
}

int pvMain(PARAM *p)
{
int ret;

  POPUP_DATA popup_data;
  memset(&popup_data, 0, sizeof(popup_data));
  p->user = &popup_data;

  pvSetCaption(p,"pvsexample");
  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;
  }
  else if(strncmp(p->initial_mask,"mask10",6) == 0)
  {
    getParams(p->initial_mask, &popup_data);
    ret = 10;
  }
  if(trace) printf("initial_mask=%s\n",p->initial_mask);
  if(trace) printf("url=%s\n",p->url);
  pvDownloadFile(p,"index.html"); // provide help for the user
                                  // you can also download pages linked within index.html
  pvDownloadFile(p,"page.html"); // provide help for the user
                                  // you can also download pages linked within index.html

  while(1)
  {
    switch(ret)
    {
      case 11:
        pvStatusMessage(p,-1,-1,-1,"mask11");
        ret = show_mask11(p);
```

```
        break;
      case 10:
        pvStatusMessage(p,-1,-1,-1,"mask10");
        ret = show_mask10(p);
        break;
      case 9:
        pvStatusMessage(p,-1,-1,-1,"mask9");
        ret = show_mask9(p);
        break;
      case 8:
        pvStatusMessage(p,-1,-1,-1,"mask8");
        ret = show_mask8(p);
        break;
      case 7:
        pvStatusMessage(p,-1,-1,-1,"mask7");
        ret = show_mask7(p);
        break;
      case 6:
        pvStatusMessage(p,-1,-1,-1,"mask6");
        ret = show_mask6(p);
        break;
      case 5:
        pvStatusMessage(p,-1,-1,-1,"mask5");
        ret = show_mask5(p);
        break;
      case 4:
        pvStatusMessage(p,-1,-1,-1,"mask4");
        ret = show_mask4(p);
        break;
      case 3:
        pvStatusMessage(p,-1,-1,-1,"mask3");
        ret = show_mask3(p);
        break;
      case 2:
        pvStatusMessage(p,-1,-1,-1,"mask2");
        ret = show_mask2(p);
        break;
      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 */
  while(1)
  {
    s = pvAccept(&p);
    if(s != -1) pvCreateThread(&p,s);
    else       break;
  }
  return 0;
}
#endif
```

6.1.3 Masken

Aus der Endlosschleife in pvMain() werden die einzelnen Masken aufgerufen, die in Ihrer Visualisierung implementiert sind. Die Start Maske aus pvsexample wird nun gezeigt.
In show_mask1() ist eine Ereignisschleife enthalten. Die einzelnen Ereignisse werden dort aufbereitet und es wird die zugehörige 'slot Funktion' aufgerufen, in der Sie Ihren Code eingeben können. Die Datei selber brauchen sie niemals selber ändern. Das wird Ihnen von pvdevelop abgenommen.
In generated_defineMask() werden die von Ihnen graphisch entworfenen Widgets dem pvbrowser Client bekannt gemacht. pvbrowser wird die entsprechenden Konstruktoren für die Widgets aufrufen.
Alle 'pv-Funktionen' Funktionen haben als ersten Parameter die Datenstruktur PARAM. Diese beinhaltet u.a. den Socket, über den mit dem Client kommuniziert wird. Die 'pv-Funktionen' senden eine ASCII Text an den Client, der diesen interpretiert und in Aufrufe von Qt umsetzt.
Der enum am Anfang der Datei listet die Widgetnamen auf, die von Ihnen eingegeben wurden. Diese Namen entsprechen der id in den 'pv-Funktionen' und adressieren das entsprechende Widget im Client. Der Client besitzt dazu ein Feld mit Zeigern auf Widgets. Das enum gibt den Index in dieses Feld an.
Die 'slot Funktionen' werden mit einem #include in die Maske eingebunden.

Quelltext zum vollständig generierten Teil einer Maske

```
////////////////////////////////////////////////////////////////////////////
//
// show_mask1 for ProcessViewServer created: Mi Nov 8 11:58:45 2006
//
////////////////////////////////////////////////////////////////////////////
#include "pvapp.h"

// _begin_of_generated_area_ (do not edit -> use ui2pvc) -------------------

// our mask contains the following objects
enum {
  ID_MAIN_WIDGET = 0,
  iconView1,
  welcomeLabel,
  helpLabel,
  buttonRestroom,
  ID_END_OF_WIDGETS
};

  static const char *toolTip[] = {
  "",
  "",
  "",
  "",
  "",
  ""};

  static const char *whatsThis[] = {
  "",
  "",
  "",
```

```
  "",
  "",
  ""};

  static const int widgetType[ID_END_OF_WIDGETS+1] = {
  0,
  TQIconView,
  TQLabel,
  TQLabel,
  TQPushButton,
  -1 };

static int generated_defineMask(PARAM *p)
{
  int w,h,depth;

  if(p == NULL) return 1;
  w = h = depth = strcmp(toolTip[0],whatsThis[0]);
  if(widgetType[0] == -1) return 1;
  if(w==h) depth=0; // fool the compiler
  pvStartDefinition(p,ID_END_OF_WIDGETS);

  pvQIconView(p,iconView1,0);
  pvSetGeometry(p,iconView1,20,85,665,515);

  pvQLabel(p,welcomeLabel,0);
  pvSetGeometry(p,welcomeLabel,20,10,500,25);
  pvSetText(p,welcomeLabel,pvtr("Welcome␣to␣pvbrowser␣pvsexample"));
  pvSetFont(p,welcomeLabel,"Sans␣Serif",18,1,1,0,0);

  pvQLabel(p,helpLabel,0);
  pvSetGeometry(p,helpLabel,20,40,265,30);
  pvSetText(p,helpLabel,pvtr("Click␣on␣icon␣to␣see␣further␣masks␣..."));

  pvQPushButton(p,buttonRestroom,0);
  pvSetGeometry(p,buttonRestroom,545,15,130,40);
  pvSetText(p,buttonRestroom,pvtr("Goto␣restroom"));
  pvSetFont(p,buttonRestroom,"Sans␣Serif",10,0,0,0,0);
  pvSetMinSize(p,buttonRestroom,0,40);

  pvQLayoutVbox(p,ID_MAIN_WIDGET,-1);

  pvAddWidgetOrLayout(p,ID_MAIN_WIDGET,welcomeLabel,-1,-1);
  pvAddWidgetOrLayout(p,ID_MAIN_WIDGET,buttonRestroom,-1,-1);
  pvAddWidgetOrLayout(p,ID_MAIN_WIDGET,helpLabel,-1,-1);
  pvAddWidgetOrLayout(p,ID_MAIN_WIDGET,iconView1,-1,-1);

  pvEndDefinition(p);
  return 0;
}

// _end_of_generated_area_ (do not edit -> use ui2pvc) ---------------------

#include "mask1_slots.h"

static int defineMask(PARAM *p)
{
  if(p == NULL) return 1;
  generated_defineMask(p);
  // (todo: add your code here)
  return 0;
}
```

```
static int showData(PARAM *p, DATA *d)
{
  if(p == NULL) return 1;
  if(d == NULL) return 1;
  return 0;
}

static int readData(DATA *d) // from shared memory, database or something else
{
  if(d == NULL) return 1;
  // (todo: add your code here)
  return 0;
}


int show_mask1(PARAM *p)
{
  DATA d;
  char event[MAX_EVENT_LENGTH];
  char text[MAX_EVENT_LENGTH];
  char str1[MAX_EVENT_LENGTH];
  int  i,w,h,val,x,y,button,ret;
  float xval, yval;

  defineMask(p);
  //rlSetDebugPrintf(1);
  if((ret=slotInit(p,&d)) != 0) return ret;
  readData(&d); // from shared memory, database or something else
  showData(p,&d);
  pvClearMessageQueue(p);
  while(1)
  {
    pvPollEvent(p,event);
    switch(pvParseEvent(event, &i, text))
    {
      case NULL_EVENT:
        readData(&d); // from shared memory, database or something else
        showData(p,&d);
        if((ret=slotNullEvent(p,&d)) != 0) return ret;
        break;
      case BUTTON_EVENT:
        if(trace) printf("BUTTON_EVENT␣id=%d\n",i);
        if((ret=slotButtonEvent(p,i,&d)) != 0) return ret;
        break;
      case BUTTON_PRESSED_EVENT:
        if(trace) printf("BUTTON_PRESSED_EVENT␣id=%d\n",i);
        if((ret=slotButtonPressedEvent(p,i,&d)) != 0) return ret;
        break;
      case BUTTON_RELEASED_EVENT:
        if(trace) printf("BUTTON_RELEASED_EVENT␣id=%d\n",i);
        if((ret=slotButtonReleasedEvent(p,i,&d)) != 0) return ret;
        break;
      case TEXT_EVENT:
        if(trace) printf("TEXT_EVENT␣id=%d␣%s\n",i,text);
        if((ret=slotTextEvent(p,i,&d,text)) != 0) return ret;
        break;
      case SLIDER_EVENT:
        sscanf(text,"(%d)",&val);
        if(trace) printf("SLIDER_EVENT␣val=%d\n",val);
        if((ret=slotSliderEvent(p,i,&d,val)) != 0) return ret;
        break;
      case CHECKBOX_EVENT:
```

```
      if(trace) printf("CHECKBOX_EVENT id=%d %s\n",i,text);
      if((ret=slotCheckboxEvent(p,i,&d,text)) != 0) return ret;
      break;
    case RADIOBUTTON_EVENT:
      if(trace) printf("RADIOBUTTON_EVENT id=%d %s\n",i,text);
      if((ret=slotRadioButtonEvent(p,i,&d,text)) != 0) return ret;
      break;
    case GL_INITIALIZE_EVENT:
      if(trace) printf("you have to call initializeGL()\n");
      if((ret=slotGlInitializeEvent(p,i,&d)) != 0) return ret;
      break;
    case GL_PAINT_EVENT:
      if(trace) printf("you have to call paintGL()\n");
      if((ret=slotGlPaintEvent(p,i,&d)) != 0) return ret;
      break;
    case GL_RESIZE_EVENT:
      sscanf(text,"(%d,%d)",&w,&h);
      if(trace) printf("you have to call resizeGL(w,h)\n");
      if((ret=slotGlResizeEvent(p,i,&d,w,h)) != 0) return ret;
      break;
    case GL_IDLE_EVENT:
      if((ret=slotGlIdleEvent(p,i,&d)) != 0) return ret;
      break;
    case TAB_EVENT:
      sscanf(text,"(%d)",&val);
      if(trace) printf("TAB_EVENT(%d,page=%d)\n",i,val);
      if((ret=slotTabEvent(p,i,&d,val)) != 0) return ret;
      break;
    case TABLE_TEXT_EVENT:
      sscanf(text,"(%d,%d,",&x,&y);
      pvGetText(text,str1);
      if(trace) printf("TABLE_TEXT_EVENT(%d,%d,\"%s\")\n",x,y,str1);
      if((ret=slotTableTextEvent(p,i,&d,x,y,str1)) != 0) return ret;
      break;
    case TABLE_CLICKED_EVENT:
      sscanf(text,"(%d,%d,%d)",&x,&y,&button);
      if(trace) printf("TABLE_CLICKED_EVENT(%d,%d,button=%d)\n",x,y,button);
      if((ret=slotTableClickedEvent(p,i,&d,x,y,button)) != 0) return ret;
      break;
    case SELECTION_EVENT:
      sscanf(text,"(%d,",&val);
      pvGetText(text,str1);
      if(trace) printf("SELECTION_EVENT(column=%d,\"%s\")\n",val,str1);
      if((ret=slotSelectionEvent(p,i,&d,val,str1)) != 0) return ret;
      break;
    case CLIPBOARD_EVENT:
      sscanf(text,"(%d",&val);
      if(trace) printf("CLIPBOARD_EVENT(id=%d)\n",val);
      if(trace) printf("clipboard = \n%s\n",p->clipboard);
      if((ret=slotClipboardEvent(p,i,&d,val)) != 0) return ret;
      break;
    case RIGHT_MOUSE_EVENT:
      if(trace) printf("RIGHT_MOUSE_EVENT id=%d text=%s\n",i,text);
      if((ret=slotRightMouseEvent(p,i,&d,text)) != 0) return ret;
      break;
    case KEYBOARD_EVENT:
      sscanf(text,"(%d",&val);
      if(trace) printf("KEYBOARD_EVENT modifier=%d key=%d\n",i,val);
      if((ret=slotKeyboardEvent(p,i,&d,val)) != 0) return ret;
      break;
    case PLOT_MOUSE_MOVED_EVENT:
      sscanf(text,"(%f,%f)",&xval,&yval);
      if(trace) printf("PLOT_MOUSE_MOVE %f %f\n",xval,yval);
```

```
      if((ret=slotMouseMovedEvent(p,i,&d,xval,yval)) != 0) return ret;
      break;
    case PLOT_MOUSE_PRESSED_EVENT:
      sscanf(text,"(%f,%f)",&xval,&yval);
      if(trace) printf("PLOT_MOUSE_PRESSED␣%f␣%f\n",xval,yval);
      if((ret=slotMousePressedEvent(p,i,&d,xval,yval)) != 0) return ret;
      break;
    case PLOT_MOUSE_RELEASED_EVENT:
      sscanf(text,"(%f,%f)",&xval,&yval);
      if(trace) printf("PLOT_MOUSE_RELEASED␣%f␣%f\n",xval,yval);
      if((ret=slotMouseReleasedEvent(p,i,&d,xval,yval)) != 0) return ret;
      break;
    case MOUSE_OVER_EVENT:
      sscanf(text,"%d",&val);
      if(trace) printf("MOUSE_OVER_EVENT␣id=%d␣%d\n",i,val);
      if((ret=slotMouseOverEvent(p,i,&d,val)) != 0) return ret;
      break;
    case USER_EVENT:
      if(trace) printf("USER_EVENT␣id=%d␣%s\n",i,text);
      if((ret=slotUserEvent(p,i,&d,text)) != 0) return ret;
      break;
    default:
      if(trace) printf("UNKNOWN_EVENT␣id=%d␣%s\n",i,text);
      break;
  }
 }
}
```

6.1.4 Slot Funktionen

Die 'slot Funktionen' müssen vom Entwickler einer Visualisierung programmiert werden. Am Anfang der Datei steht die Definition einer Datenstruktur DATA. Darin kann der Entwickler Daten definieren, die privat für diese Maske sind. Das kann mit einer C++ Klasse verglichen werden, obwohl das an dieser Stelle in ANSI C ausgedrückt wird.

Die slot Funktionen, die der Entwickler einer Visualisierung ausfüllen muss

```
//################################################################
//# mask1_slots.h for ProcessViewServer created: Mi Nov 8 11:58:45 2006
//# please fill out these slots
//# here you find all possible events
//# Yours: Lehrig Software Engineering
//################################################################
//WELLCOME
// 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)
{
}
DATA;

static int slotInit(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  //memset(d,0,sizeof(DATA));
  if(1)
  {
    pvHide(p,buttonRestroom); // switch on/off restroom support
```

```
  }
  else
  {
    pvSetPixmap(p,buttonRestroom,"restroom.png");
  }
  pvPrintf(p,ID_TAB,"Welcome");
  pvSetIconViewItem(p,iconView1,"xchat.png","Chat");
  pvSetIconViewItem(p,iconView1,"kmplot.png","Plot");
  pvSetIconViewItem(p,iconView1,"gnumeric.png","Table");
  pvSetIconViewItem(p,iconView1,"kpl.png","SVG1");
  pvSetIconViewItem(p,iconView1,"package_network.png","VTK");
  pvSetIconViewItem(p,iconView1,"xapp.png","GnuPlot");
  pvSetIconViewItem(p,iconView1,"taskbar.png","Tree");
  pvSetIconViewItem(p,iconView1,"xedit.png","SVG2");
  pvSetIconViewItem(p,iconView1,"autocad.png","DWF_2D");
  return 0;
}

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

static int slotButtonEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  if(id == buttonRestroom)
  {
    pvHyperlink(p,"pv://localhost:5051");
  }
  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;
  if(id == iconView1)
  {
    if    (strcmp(text,"Chat")  == 0) return CHAT1;
    else if(strcmp(text,"Plot") == 0) return PLOT1;
    else if(strcmp(text,"Table") == 0) return TABLE1;
    else if(strcmp(text,"SVG1") == 0) return SVG1;
    else if(strcmp(text,"VTK")  == 0) return VTK1;
    else if(strcmp(text,"GnuPlot") == 0) return GNUPLOT1;
    else if(strcmp(text,"Tree") == 0) return TREE1;
    else if(strcmp(text,"SVG2") == 0) return SVG2;
    else if(strcmp(text,"DWF_2D") == 0) return DWF2GL;
    else pvPrintf(p,helpLabel,"%s␣not␣implemented␣jet",text);
  }
  return 0;
```

```
}

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

6.1.5 Header Datei

Die Header Datei pvapp.h wird in jede Maske inkludiert. Darin kann man seine eigenen Definitionen unterbringen. Über die Variable trace können die Ausgaben des Servers abgeschaltet werden. Bei der Entwicklung sind diese Ausgaben aber hilfreich und sollten eingeschaltet sein. Wenn man INETD nutzt, müssen diese Ausgaben abgeschaltet werden, da diese sonst an den pvbrowser Client gehen würden und diesen stören könnten.

Header Datei pvapp.h für alle Masken

```
//***************************************************************************
//                       pvapp.h -  description
//                          -------------------
// begin          : Mi Nov 8 11:58:45 2006
// generated by   : pvdevelop (C) 2000-2006 by Lehrig Software Engineering
// email          : lehrig@t-online.de
//***************************************************************************
#ifndef _PVAPP_H_
#define _PVAPP_H_

#include "processviewserver.h"
#include "rleventlogserver.h"
#include "rltime.h"
#include "rlcutil.h"
#include "rlsvganimator.h"
#include <math.h>

static int trace = 1;

#ifdef unix
#define LOGFILE "/var/log/pvbchat.log"
#else
#define LOGFILE NULL
#endif

#define ESC_KEY 16777216
#define PI 3.141592f
// todo: comment me out
//#include "rlmodbusclient.h"
//#include "rlsiemenstcpclient.h"
//#include "rlppiclient.h"
//#include "modbusdaemon.h"         // this is generated
//#include "siemensdaemon.h"        // this is generated
//#include "ppidaemon.h"            // this is generated

// these are our masks
enum {
  WELLCOME = 1,
  CHAT1    = 2,
  PLOT1    = 3,
  MODAL1   = 4,
  TABLE1   = 5,
  SVG1     = 6,
  VTK1     = 7,
  GNUPLOT1 = 8,
  TREE1    = 9,
  SVG2     = 10,
  DWF2GL   = 11
};

// this is for SVG2 begin
typedef struct
{
  float x0, y0, x1, y1, scale, svgx0, svgy0;
  bool popup;
}
POPUP_DATA;

typedef struct
{
  float lifter_height;
}
SIMULATION_DATA;
```

```
// this is for SVG2 end

int initializeGL(PARAM *p);
int resizeGL(PARAM *p, int width, int height);

int show_mask11(PARAM *p);
int show_mask10(PARAM *p);
int show_mask9(PARAM *p);
int show_mask8(PARAM *p);
int show_mask7(PARAM *p);
int show_mask6(PARAM *p);
int show_mask5(PARAM *p);
int show_mask4(PARAM *p);
int show_mask3(PARAM *p);
int show_mask2(PARAM *p);
int show_mask1(PARAM *p);

#endif
```

6.1.6 Datenstruktur PARAM

Die Datenstruktur PARAM wird bei allen 'pv-Funktionen' als 1 Parameter angegeben. Sehen Sie sich bitte an, welche Informationen darin verfügbar sind.

Die Datenstruktur PARAM beschreibt die Verbindung zum Client

```
typedef struct _PARAM_
{
  int  s;                          /* socket                          */
  int  os;                         /* original socket                 */
  int  port;                       /* our port                        */
  int  language;                   /* language or DEFAULT_LANGUAGE    */
  int  convert_units;              /* 1 if units must be converted    */
  FILE *fp;                        /* filepointer                     */
  int  sleep;                      /* sleep time in milliseconds      */
  int  (*cleanup)(void *);         /* cleanup for user code           */
  void *app_data;                  /* application data for cleanup    */
  void *user;                      /* pointer to user data            */
  char *clipboard;                 /* pointer to clipboard text | NULL */
  long clipboard_length;           /* sizeof clipboard contents       */
  int  modal;                      /* modal dialog                    */
  int  (*readData)(void *d);       /* modal dialog                    */
  int  (*showData)(_PARAM_ *p, void *d); /* modal dialog              */
  void *modal_d;                   /* modal dialog                    */
  void *modalUserData;             /* modal dialog                    */
  PARSE_EVENT_STRUCT parse_event_struct;
  float *x;                        /* array buffer for script language */
  float *y;                        /* array buffer for script language */
  int   nxy;                       /* number of elements in arry      */
  char  url[MAX_PRINTF_LENGTH];    /* url the client is using         */
  char  initial_mask[MAX_PRINTF_LENGTH]; /* initial mask user wants to see */
  char  file_prefix[32];           /* prefix for temporary files      */
                                   /* files with this prefix will be */
                                   /* deleted on connection lost      */
  int   free;                      /* free structure                  */
  char  version[32];               /* pvbrowser VERSION of client     */
  char  pvserver_version[32];      /* pvserver VERSION                */
  int   exit_on_bind_error;        /* exit if we can not bind on port */
  int   hello_counter;             /* for thread timeout if no @hello */
  int   local_milliseconds;        /* time of last call to select()   */
  int   force_null_event;          /* force null_event for better update */
                                   /* if the user has tabs within his */
                                   /* client the invisible tab are    */
                                   /* paused by default               */
  int   allow_pause;               /* 0 not allowed else allowed      */
  int   pause;                     /* pause=1 if tab invisible else 0 */
                                   /* you can test pause in NULL_EVENT */
  int   my_pvlock_count;           /* used to avoid deadlock by repeated */
                                   /* call of pvlock                  */
  int   num_additional_widgets;    /* additional widgets after        */
                                   /* ID_END_OF_WIDGETS               */
  int   mouse_x, mouse_y;          /* last mouse pos when pressed     */
  char  *mytext;                   /* buffer for internal use only    */
  const char *communication_plugin; /* pointer to commandline arg or NULL */
  int   use_communication_plugin;  /* can also be set at runtime      */
  char  lang_section[32];          /* use pvSelectLanguage()          */
  char  *mytext2;                  /* temp used in language translation */
  int   http;                      /* 0|1 talk http                   */
  FILE  *fptmp;                    /* temporary file pointer          */
  int   fhdltmp;                   /* temporary file handle           */
}PARAM;
```

6.2 Slot Programmierung

In dem vorangegangenen Kapitel haben Sie die 'slot Funktionen' bereits gesehen. Das Gerüst ist von pvdevelop erzeugt worden. Jetzt müssen Sie diese Funktionen ausfüllen, um die Logik Ihrer Visualisierung zu programmieren.
Der slotInit() ist für die Initialisierung Ihrer Variablen in DATA verantwortlich.
Der slotNullEvent() wird zyklisch im Intervall von (PARAM *) p->sleep Millisekunden aufgerufen. Darin können Sie laufende Updates des pvbrowser Fenster einbauen.
Die anderen 'slot Funktionen' werden aufgerufen, wenn der Benutzer ein Ereignis in pvbrowser auslöst, wie z.B. das Drücken eines Buttons.

Beispiel zu slot Funktionen

```
typedef struct // (todo: define your data structure here)
{
  rlSvgAnimator svgAnimator;
}
DATA;

static int drawSVG1(PARAM *p, int id, DATA *d)
{
  if(d == NULL) return -1;
  if(d->svgAnimator.isModified == 0) return 0;
  printf("writeSocket\n");
  gBeginDraw(p,id);
  d->svgAnimator.writeSocket();
  gEndDraw(p);
  return 0;
}

static int slotInit(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  //memset(d,0,sizeof(DATA));

  // load HTML
  pvDownloadFile(p,"icon32x32.png");
  pvDownloadFile(p,"upperWidget.html");
  pvDownloadFile(p,"leftWidget.html");
  pvSetSource(p,upperWidget,"upperWidget.html");
  pvSetSource(p,leftWidget,"leftWidget.html");

  // load SVG
  d->svgAnimator.setSocket(&p->s);
  d->svgAnimator.setId(centerWidget);
  d->svgAnimator.read("test.svg");

  // keep aspect ratio of SVG
  pvSetZoomX(p, centerWidget, -1.0f);
  pvSetZoomY(p, centerWidget, -1.0f);

  // draw SVG
  drawSVG1(p,centerWidget,d);

  // download icons
  pvDownloadFile(p,"1center.png");
  pvDownloadFile(p,"1uparrow.png");
  pvDownloadFile(p,"1downarrow.png");
  pvDownloadFile(p,"1leftarrow.png");
  pvDownloadFile(p,"1rightarrow.png");
  pvDownloadFile(p,"1center2.png");
  pvDownloadFile(p,"1uparrow2.png");
  pvDownloadFile(p,"1downarrow2.png");
  pvDownloadFile(p,"1leftarrow2.png");
```

```
  pvDownloadFile(p,"1rightarrow2.png");

  // set sliderZoom to 100 percent
  pvSetValue(p,sliderZoom,100);

  //pvSetSource(p,upperWidget,"de_total.html");
  return 0;
}

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

static int slotButtonEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  if    (id == iCenter)
  {
    pvSetImage(p,iCenter,"1center2.png");
    d->svgAnimator.zoomCenter(1.0f);
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.setXY0(0.0f,0.0f);
    d->svgAnimator.moveMainObject(0,0);
    drawSVG1(p,centerWidget,d);
    pvSetValue(p,sliderZoom,100);
  }
  else if(id == iUp)
  {
    pvSetImage(p,iUp,"1uparrow2.png");
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.moveMainObject(0,-DELTA);
    drawSVG1(p,centerWidget,d);
  }
  else if(id == iDown)
  {
    pvSetImage(p,iDown,"1downarrow2.png");
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.moveMainObject(0,DELTA);
    drawSVG1(p,centerWidget,d);
  }
  else if(id == iLeft)
  {
    pvSetImage(p,iLeft,"1leftarrow2.png");
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.moveMainObject(-DELTA,0);
    drawSVG1(p,centerWidget,d);
  }
  else if(id == iRight)
  {
    pvSetImage(p,iRight,"1rightarrow2.png");
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.moveMainObject(DELTA,0);
    drawSVG1(p,centerWidget,d);
  }
  else if(id == pbPrintHtml)
  {
    pvPrintHtmlOnPrinter(p,upperWidget);
  }
  return 0;
}
```

6.3 Util Funktionen

Die 'Util Funktionen' dienen zur Ansteuerung des pvbrowser Fensters vom pvserver. Diese Funktionen senden ASCII-Texte zum pvbrowser Client, die dort interpretiert und in Aufrufe von Qt umgesetzt werden. In den von pvdevelop generierten mask∗.cpp Dateien werden die Konstruktoren für die von Ihnen entworfenen Widgets aufgerufen 'generated_defineMask()'. Der Name der 'Util Funktionen' beginnt mit 'pv'. Der erste Parameter der 'Util Funktionen' ist der (PARAM ∗p), der die Verbindung zu pvbrowser Client beschreibt. Der zweite Parameter enthält bei den meisten 'Util Funktionen' die 'int id', welche das jeweilige Widget adressiert. Die id ist einer der Namen, die in dem enum am Anfang von mask∗.cpp aus Ihrem Design generiert wurde.
Das Referenzmanual der 'Util Funktionen' finden Sie auf unserer Homepage und im Hilfe-Manual innerhalb von pvdevelop. Sehen Sie sich dort besonders im Unterpunkt 'Construction' um, da dort alle Funktionen aufgelistet werden, die sich auf ein spezielles Widget beziehen.

6.4 rllib

Die rllib ermöglicht die plattformunabhängige Programmierung von vielen Systemfunktionen auf der Serverseite. Darin sind auch Klassen zur Unterstützung diverser SPS/Feldbus-Protokolle enthalten. Um die rllib in Ihren pvserver mit einzubinden wählen Sie bitte das Menu 'RLlib->Uncomment rllib' in pvdevelop.

- *rl3964R* implementiert das Siemens 3964R (Dust) Protokoll
- *rlbussignaldatabase* Klasse zum Speichern/Lesen von Prozessvariablen in einer MySQL Datenbank.
- *rlController* Diese Klasse implementiert Übertragungsfunktionen für Regelungen. entsprechend 'F. Doerrscheid/W. Latzel, Grundlagen der Regelungstechnik, B.G. Teubner Stuttgart, Regelalgorithmen mit der Trapezregel'
- *rlCorbaClient* dient zur Kapselung von Corba Kommunikationen.
- *rlCorbaServer* dient zur Kapselung von Corba Kommunikationen.
- *rlcutil* enthält einige einfache C Funktionen.
- *rlDataProvider* Container für Prozessvariablen.
- *rlDataProviderClient* Client für den Zugriff auf Prozessvariablen, die über einen Server mit *rlDataProviderThreads* zur über das Netzwerk zur Verfügung gestellt werden.
- *rlDataProviderThreads* Stellt *rlDataProvider* Container über ein Netzwerk zur Verfügung. Dazu wird ein Thread gestartet, der beliebig viele *rlDataProviderClient* bedienen kann.
- *rlDataAcquisition* dient zur Datenerfassung nach dem 'pvbrowser Prinzip'. Aus pvserver oder anderen Anwendungen kann man damit auf das Shared Memory und die Mailbox zugreifen, mit der der Datenaustausch mit dem Daemon stattfindet. Diese Klasse repräsentiert die Prozessvariablen als lesbaren ASCII-Text. Siehe auch *rlDataAquisitionProvider*.
- *rlDataAcquisitionProvider* Diese Klasse wird auf der Seite eines Daemons verwendet, um die Prozessvariablen in das Shared Memory zu schreiben und Ausgabewerte von der Mailbox zu empfangen. Siehe auch *rlDataAquisition*
- *rlEIBnetIp* implementiert EIBnetIp (European Installation Bus) über Netzwerk.
- *rlEventLogServer* dient zur Implementierung eines Servers, der Event Log Botschaften von anderen Prozessen (auch über Netzwerk) empfangen kann. Ein solcher Server ist bereits als pvserver in 'pcontrol' (siehe pvbaddon) implementiert.
- *rlEventLogServerThreads* dient zur Implementierung eines Servers, der Event Log Botschaften von anderen Prozessen (auch über Netzwerk) empfangen kann. Ein solcher Server ist bereits als pvserver in 'pcontrol' (siehe pvbaddon) implementiert.
- *rlFifo* First In First out Puffer zur Kommunikation unter verschiedenen Threads.
- *rlFileLoad* lädt einen ASCII-File, der dann innerhalb des Arbeitsspeichers iteriert werden kann.

- *rlHilscherCIF* Wrapper für den Gerätetreiber für Hilscher CIF Karten (Profibus, CAN, ...).
- *rlHistoryLogger* dient zur Aufzeichnung einer History in ASCII-Dateien und im Arbeitsspeicher.
- *rlHistoryReader* dient zur zum Lesen einer mit *rlHistoryLogger* aufgezeichneten History.
- *rlIniFile* implementiert INI Dateien, wie man sie von Windows kennt. Damit können INI Dateien gelesen, geschrieben und manipuliert werden. Ausserdem kann man mit Hilfe der Methode i18n Programmtexte Internationalisieren.
- *rlInterpreter* dienst zur Implementierung eines Interpreters für Kommandozeilen.
- *rlIpAdr* IP Adresse für die Verwendung zusammen mit *rlUdpSocket*
- *rlMailbox* implementiert eine Mailbox, d.h. einen Mechanismus zur Kommunikation zwischen Prozessen auf einem Rechner, bei dem mehrere Prozesse auf die Mailbox schreiben dürfen, aber nur ein Prozess liest.
- *rlModbus* implementiert das Modbus Protokoll. Es werden sowohl Modbus RTU, Modbus ASCII über die serielle Schnittstelle, als auch Modbus TCP unterstützt.
- *rlModbusclient* Klasse zur Verwendung in einem pvserver, mit dem man auf das Shared Memory und die Mailbox zugreift, die von einem aus pvdevelop generierten modbusdaemon bedient werden. Die Prozessdaten werden binär kodiert repräsentiert.
- *rlMutex* implementiert einen Mutex.
- *rlOpcXmlDa* implementiert einen Client für das OPC XML/DA Protokoll (HTTP/SOAP/XML). In pvbaddon ist ein Daemon enthalten, der damit arbeitet.
- *rlPcontrol* Bietet eine Klasse mit der Prozesse gestartet und überwacht werden können.
- *rlPlcMem* Implementiert Variablen für eine Soft-SPS.
- *rlPlcState* Implementiert Felder von Variablen für eine Soft-SPS.
- *rlPPIClient* implementiert das Siemens PPI Protokoll mit Hilfe von libnodave.
- *rlSerial* Klasse für serielle Schnittstellen (RS232/RS485)
- *rlSharedMemory* Implementiert ein Shared Memory, d.h. RAM, das von mehreren Prozessen gleichzeitig gesehen werden kann.
- *rlSiemensTCP* Implementiert das Siemens SPS Protokoll für S7-Serie und S5.
- *rlSiemensTCPClient* Klasse mit der aus einem pvserver auf das Shared Memory und die Mailbox zugegriffen werden kann, die von einem aus pvdevelop generierten siemensdaemon bereitgestellt werden. Die Daten werden binär kodiert.
- *rlSocket* Socket zur Kommunikation über TCP, sowohl mit IPv4 als auch IPv6.
- *rlSpawn* Startet einen anderen Prozess und verbindet dessen STDIN und STDOUT über eine Pipe mit dieser Klasse. Das ist beispielsweise von Nutzen, wenn man eine graphische Oberfläche für Kommandozeilen orientierte Anwendungen erstellen möchte. ACHTUNG: Nur unter unixartigen Betriebssystemen verfügbar.
- *rlSpreadsheetCell* Zelle für eine Tabelle.
- *rlSpreadsheetRow* Zeile für eine Tabelle.
- *rlSpreadsheetTable* Tabelle. Tabellen können CSV Dateien lesen und schreiben.
- *rlSpreadsheetWorkbook* Mehrere Tabellen. Tabellen können als Serie von CSV Dateien gelesen und geschrieben werden.
- *rlString* Einfache String Klasse.

- *rlSvgAnimator* Klasse zur Animation von SVG Grafiken aus einem pvserver.
- *rlSvgCat* Klasse zur 'Normalisierung' von SVG bzw. XML Code. Die einzelnen Tags werden auf einzelne Zeilen aufgeteilt und linksbündig auf STDOUT geschrieben.
- *rlSvgPosition* Wird zusammen mit *rlSvgAnimator* verwendet und stellt die Position eines graphischen Objektes einer SVG Grafik dar.
- *rlThread* Wrapper für Threads mit Hilfe von pthread bzw. den Windows Thread Routinen.
- *rlTime* implementiert Uhrzeit und Datum.
- *rlUdpSocket* Netzwerkkommunikation über UDP.
- *rlWebcam* implementiert einen Client, der Motion JPEG Webcams über http anbinden kann.
- *rlwthread* C Funktionen zur Kapselung von Threads.

Das Referenzmanual der rllib finden Sie auf unserer Homepage und im Hilfe-Manual innerhalb von pvdevelop.

6.5 Lua

Lua `http://www.lua.org/` ist eine Skriptsprache zur Einbettung in ein C/C++ Hauptprogramm. In einem pvserver wird für jeden sich neu verbindenden Client ein neuer Thread eröffnet, der mit der Funktion pvMain() beginnt den Client zu bedienen. Bei der Lua Variante eines pvserver wird dort das Lua Script 'main.Lua' mit der Funktion 'luaMain()' aufgerufen. Die komplette Behandlung des Client kann nun in Lua kodiert werden. Unsere Bibliotheken sind dazu mit Hilfe von Swig `http://swig.org/` mit einer Sprachenanbindung an Lua versehen worden.
Ein Vorteil von Lua ist die geringe Größe der Bibliothek, die Lua implementiert. Das erlaubt es Lua statisch in Gastprogramme einzubinden. Der Endbenutzer muss Lua daher gar nicht installieren. Man kann daher unsere pvserver in Lua entwickeln ohne zusätzliche Programmpakete installieren zu müssen. Es wird weder ein C/C++ Compiler, noch das Qt SDK benötigt.
Ein weiterer Vorteil dieser Lösung ist, dass man den pvserver während des Betriebes ohne Compilieren zu müssen ändern kann. Jeder sich neu anmeldende Benutzer wird von dem aktuell vorhandenen Lua Code bedient ohne dass es notwendig wäre den pvserver neu zu starten. Dies ist vergleichbar zur Verwendung von PHP bei Webservern.
In pvdevelop muss man beim Anlegen eines neuen Projektes auswählen, in welcher Sprache es programmiert werden soll. Wenn man dort 'Lua' wählt, erstellt pvdevelop das Gerüst des Lua Skriptes. Der für das Design der Masken verantwortliche Teil des Codes wird komplett von pvdevelop generiert. Die 'slotFunktionen' werden beim Anlegen einer neuen Maske generiert. Der Programmierer der Visualisierung muss dann nur noch diese 'slotFunktionen' codieren.
Im pvbaddon Verzeichnis 'pvbaddon/demos/lua/' findet man Beispielprojekte in Lua. Hier soll ein 'Hallo Welt' pvserver in Lua gezeigt werden, der Modbus und eine SQL Datenbank anbindet.

6.5.1 main.lua

Das Lua Hauptprogramm

```
--------------------------------------------------------------------------------
-- pvserver in lua   run: pvslua -port=5050 -cd=/your/directory/with/your/lua/code
--------------------------------------------------------------------------------
trace = 1            -- here you may put variables global for all your masks
-- declare the data acquisition class for connecting to modbus
-- this class communicates with the modus_daemon via a shared memeory and a mailbox
--                                        Mailbox                       Shared Memory ShmSize
mb = rllib.rlDataAcquisition("/srv/automation/mbx/modbus1.mbx","/srv/automation/shm/modbus1.shm"
    ,65536)

qtdb = pv.qtDatabase() -- declare a Qt Database

dofile("mask1.lua")  -- include your masks here
```

```
--------------------------------------------------------------------------------
function luaMain(ptr) -- pvserver Lua Main Program

  p = pv.getParam(ptr) -- get the PARAM structure

  pv.pvSetCaption(p,string.format("Hello␣Modbus␣from␣Lua␣pvserver␣%3.1f",0.1))
  pv.pvResize(p,0,1280,1024)
  pv.pvGetInitialMask(p)
  print("Inital␣mask␣=␣", p.initial_mask)

  -- open the database
  ret = qtdb.open(qtdb,"QMYSQL","localhost","information_schema","","")
  print("qtdb.open()␣ret=", ret)

  print(string.format("Shared␣Memory␣%s:␣key=%x␣(hex)␣id=%d␣(dec)","/srv/automation/shm/modbus1.shm",
                    mb.shmKey(mb), mb.shmId(mb)))

  -- show the masks
  ret = 1
  while 1 do          -- show your masks
    if    (ret==1) then
      ret = showMask1(p)
    else
      ret = 1
    end
  end

  pv.pvThreadFatal(p,"Lua␣calling␣ThreadFatal")
  return 0

end
```

Das Lua Hauptprogramm wird von 'pvslua' (Unserem C/C++ Hauptprogramm) aufgerufen, wenn sich ein neuer Client mit 'pvslua' verbindet. Darin beginnt nun die Behandlung eines Client.
Wie man in dem Code sehen kann, werden dort Funktionen unserer C/C++ Bibliotheken von Lua aus verwendet. Als globale Klassen werden eine Datenanbindung an unser Shared Memory und die Mailbox sowie eine MySQL Datenbank definiert. Die Datenbank Klasse basiert auf den Qt Klassen für Datenbanken. Es werden also viele Typen von Datenbanken unterstützt.

Unterstützte Datenbanken

```
dbtype := Description
"QDB2"    IBM DB2, v7.1 and higher
"QIBASE"  Borland InterBase Driver
"QMYSQL"  MySQL Driver
"QOCI"    Oracle Call Interface Driver
"QODBC"   ODBC Driver (includes Microsoft SQL Server)
"QPSQL"   PostgreSQL v6.x and v7.x Driver
"QSQLITE" SQLite version 3 or above
"QSQLITE2" SQLite version 2
"QTDS"    Sybase Adaptive Server
```

In einer Schleife werden dann die Masken der Visualisierung angezeigt. Momentan ist dort nur eine Maske eingetragen.

6.5.2 maskN.lua

Code für eine Maske der Visualisierung

```
--------------------------------------------------------------------------------
-- this file is generated by pvdevelop. DO NOT EDIT !!!
--------------------------------------------------------------------------------
```

```
function showMask1(p)
  --- begin variables that are private to this mask ---------------------------------
  iarray = pv.IntegerArray()              -- see pv.getIntegers(text,iarray) below
  farray = pv.FloatArray()                -- see pv.getFloats(text,farray) below
  --- begin construction of our mask ------------------------------------------------
  ID_MAIN_WIDGET = 0
  button1 = 1
  button2 = 2
  button3 = 3
  button4 = 4
  label1 = 5
  label2 = 6
  label3 = 7
  label4 = 8
  svg1 = 9
  table1 = 10
  ID_END_OF_WIDGETS = 11

  toolTip = {}
  toolTip[0] = ""
  toolTip[1] = ""
  toolTip[2] = ""
  toolTip[3] = ""
  toolTip[4] = ""
  toolTip[5] = ""
  toolTip[6] = ""
  toolTip[7] = ""
  toolTip[8] = ""
  toolTip[9] = ""
  toolTip[10] = ""

  whatsThis = {}
  whatsThis[0] = ""
  whatsThis[1] = ""
  whatsThis[2] = ""
  whatsThis[3] = ""
  whatsThis[4] = ""
  whatsThis[5] = ""
  whatsThis[6] = ""
  whatsThis[7] = ""
  whatsThis[8] = ""
  whatsThis[9] = "test1.svg"
  whatsThis[10] = ""

  widgetType = {}
  widgetType[0] = pv.TQWidget
  widgetType[1] = pv.TQPushButton
  widgetType[2] = pv.TQPushButton
  widgetType[3] = pv.TQPushButton
  widgetType[4] = pv.TQPushButton
  widgetType[5] = pv.TQLabel
  widgetType[6] = pv.TQLabel
  widgetType[7] = pv.TQLabel
  widgetType[8] = pv.TQLabel
  widgetType[9] = pv.TQDraw
  widgetType[10] = pv.TQTable

  pv.pvStartDefinition(p,ID_END_OF_WIDGETS)

  pv.pvQPushButton(p,button1,0)
  pv.pvSetGeometry(p,button1,15,25,100,30)
  pv.pvSetText(p,button1,"Out␣1")
```

```
pv.pvSetFont(p,button1,"Sans␣Serif",10,0,0,0,0)

pv.pvQPushButton(p,button2,0)
pv.pvSetGeometry(p,button2,15,60,100,30)
pv.pvSetText(p,button2,"Out␣2")
pv.pvSetFont(p,button2,"Sans␣Serif",10,0,0,0,0)

pv.pvQPushButton(p,button3,0)
pv.pvSetGeometry(p,button3,15,95,100,30)
pv.pvSetText(p,button3,"Out␣3")
pv.pvSetFont(p,button3,"Sans␣Serif",10,0,0,0,0)

pv.pvQPushButton(p,button4,0)
pv.pvSetGeometry(p,button4,15,130,100,30)
pv.pvSetText(p,button4,"Out␣4")
pv.pvSetFont(p,button4,"Sans␣Serif",10,0,0,0,0)

pv.pvQLabel(p,label1,0)
pv.pvSetGeometry(p,label1,135,25,100,30)
pv.pvSetText(p,label1,"bit4")
pv.pvSetFont(p,label1,"Sans␣Serif",10,0,0,0,0)

pv.pvQLabel(p,label2,0)
pv.pvSetGeometry(p,label2,135,60,100,30)
pv.pvSetText(p,label2,"bit5")
pv.pvSetFont(p,label2,"Sans␣Serif",10,0,0,0,0)

pv.pvQLabel(p,label3,0)
pv.pvSetGeometry(p,label3,135,95,100,30)
pv.pvSetText(p,label3,"bit6")
pv.pvSetFont(p,label3,"Sans␣Serif",10,0,0,0,0)

pv.pvQLabel(p,label4,0)
pv.pvSetGeometry(p,label4,135,130,100,30)
pv.pvSetText(p,label4,"bit7")
pv.pvSetFont(p,label4,"Sans␣Serif",10,0,0,0,0)

pv.pvQDraw(p,svg1,0)
pv.pvSetGeometry(p,svg1,275,10,635,450)
pv.pvSetFont(p,svg1,"Sans␣Serif",10,0,0,0,0)
pv.pvSetWhatsThis(p,svg1,"test1.svg")

pv.pvQTable(p,table1,0,2,2)
pv.pvSetGeometry(p,table1,5,170,265,290)
pv.pvSetFont(p,table1,"Sans␣Serif",10,0,0,0,0)


pv.pvEndDefinition(p);
--- end construction of our mask ------------------------------------------------
--- end variables that are private to this mask ---------------------------------
dofile("mask1_slots.lua")               -- include our slot functions

if trace == 1 then print("show␣mask1") end
pv.pvClearMessageQueue(p)               -- clear all pending events
ret = slotInit(p)                       -- intitialize our variables
if ret ~= 0 then return ret end         -- return number of next mask to call
while(1)                                -- event loop
do
  event  = pv.pvGetEvent(p)             -- get the next event
  result = pv.pvParseEventStruct(p,event) -- parse the event
  id     = result.event
  i      = result.i
  text   = result.text
```

```
                                    -- now call the according slot function
if    id == pv.NULL_EVENT then
    ret = slotNullEvent(p)
elseif id == pv.BUTTON_EVENT then
    if trace==1 then print("BUTTON_EVENT id=", i) end
    ret = slotButtonEvent(p,i)
elseif id == pv.BUTTON_PRESSED_EVENT then
    if trace == 1 then print("BUTTON_PRESSED_EVENT id=",i) end
    ret=slotButtonPressedEvent(p,i)
elseif id == pv.BUTTON_RELEASED_EVENT then
    if trace == 1 then print("BUTTON_RELEASED_EVENT id=",i) end
    ret=slotButtonReleasedEvent(p,i)
elseif id == pv.TEXT_EVENT then
    if trace == 1 then print("TEXT_EVENT id=",i," text=",text) end
    ret=slotTextEvent(p,i,text)
elseif id == pv.SLIDER_EVENT then
    pv.getIntegers(text,iarray)
    if trace == 1 then print("SLIDER_EVENT val=",iarray.i0) end
    ret=slotSliderEvent(p,i,iarray.i0)
elseif id == pv.CHECKBOX_EVENT then
    if trace == 1 then print("CHECKBOX_EVENT id=",i," text=",text) end
    ret=slotCheckboxEvent(p,i,text)
elseif id == pv.RADIOBUTTON_EVENT then
    if trace == 1 then print("RADIOBUTTON_EVENT id=",i," text=",text) end
    ret=slotRadioButtonEvent(p,i,text)
elseif id == pv.GL_INITIALIZE_EVENT then
    if trace == 1 then print("you have to call initializeGL()") end
    ret=slotGlInitializeEvent(p,i)
elseif id == pv.GL_PAINT_EVENT then
    if trace == 1 then print("you have to call paintGL()") end
    ret=slotGlPaintEvent(p,i)
elseif id == pv.GL_RESIZE_EVENT then
    pv.getIntegers(text,iarray)
    if trace == 1 then print("you have to call resizeGL(w,h)") end
    ret=slotGlResizeEvent(p,i,iarray.i0,iarray.i1)
elseif id == pv.GL_IDLE_EVENT then
    ret=slotGlIdleEvent(p,i)
elseif id == pv.TAB_EVENT then
    pv.getIntegers(text,iarray)
    if trace == 1 then print("TAB_EVENT id=",i,"page=",iarray.i0) end
    ret=slotTabEvent(p,i,iarray.i0)
elseif id == pv.TABLE_TEXT_EVENT then
    pv.getIntegers(text,iarray)
    pv.pvlock(p)
    str1 = pv.getTextFromText(text)
    pv.pvunlock(p)
    if trace == 1 then print("TABLE_TEXT_EVENT id=",i," x=",iarray.i0," y=",iarray.i1," text=",
        str1) end
    ret=slotTableTextEvent(p,i,iarray.i0,iarray.i1,str1)
elseif id == pv.TABLE_CLICKED_EVENT then
    pv.getIntegers(text,iarray)
    if trace == 1 then print("TABLE_CLICKED_EVENT id=",i," x=",iarray.i0," y=",iarray.i1," button=
        ",iarray.i2) end
    ret=slotTableClickedEvent(p,i,iarray.i0,iarray.i1,iarray.i2)
elseif id == pv.SELECTION_EVENT then
    pv.getIntegers(text,iarray)
    pv.pvlock(p)
    str1 = pv.getTextFromText(text)
    pv.pvunlock(p)
    if trace == 1 then print("SELECTION_EVENT id=",i," column=",iarray.i0," text=",str1) end
    ret=slotSelectionEvent(p,i,iarray.i0,str1)
elseif id == pv.CLIPBOARD_EVENT then
    pv.getIntegers(text,iarray)
```

```
        if trace == 1 then print("CLIPBOARD_EVENT␣id=",iarray.i0) end
        if trace == 1 then print("clipboard␣=␣",p.clipboard) end
        ret=slotClipboardEvent(p,i,iarray.i0)
    elseif id == pv.RIGHT_MOUSE_EVENT then
        if trace == 1 then print("RIGHT_MOUSE_EVENT␣id=",i,"␣text=",text) end
        ret=slotRightMouseEvent(p,i,text)
    elseif id == pv.KEYBOARD_EVENT then
        pv.getIntegers(text,iarray)
        if trace == 1 then print("KEYBOARD_EVENT␣modifier=",i,"␣key=",iarray.i0) end
        ret=slotKeyboardEvent(p,i,iarray.i0,i)
    elseif id == pv.PLOT_MOUSE_MOVED_EVENT then
        pv.getFloats(text,farray)
        if trace == 1 then print("PLOT_MOUSE_MOVE␣",farray.f0,farray.f1) end
        ret=slotMouseMovedEvent(p,i,farray.f0,farray.f1)
    elseif id == pv.PLOT_MOUSE_PRESSED_EVENT then
        pv.getFloats(text,farray)
        if trace == 1 then print("PLOT_MOUSE_PRESSED␣",farray.f0,farray.f1) end
        ret=slotMousePressedEvent(p,i,farray.f0,farray.f1)
    elseif id == pv.PLOT_MOUSE_RELEASED_EVENT then
        pv.getFloats(text,farray)
        if trace == 1 then print("PLOT_MOUSE_RELEASED␣",farray.f0,farray.f1) end
        ret=slotMouseReleasedEvent(p,i,farray.f0,farray.f1)
    elseif id == pv.MOUSE_OVER_EVENT then
        pv.getIntegers(text,iarray)
        if trace == 1 then print("MOUSE_OVER_EVENT␣",iarray.i0) end
        ret=slotMouseOverEvent(p,i,iarray.i0)
    elseif id == pv.USER_EVENT then
        if trace == 1 then print("USER_EVENT␣id=",i,"␣text=",text) end
        ret=slotUserEvent(p,i,text)
    else
        if trace == 1 then print("UNKNOWN_EVENT␣id=",i,"␣text=",text) end
        ret = 0
    end
    if ret ~= 0 then return ret end           -- return number of next mask to call
  end                                         -- end of event loop
  return 0                                    -- never come here
end
```

Der Code für die Masken der Visualisierung wird von pvdevelop generiert. In der dort enthaltenen Ereignisschleife werden die 'slotFunktionen' aufgerufen, die vom Programmierer der Visualisierung kodiert werden müssen.

6.5.3 maskN_slots.lua

slotFunktionen für eine Maske der Visualisierung

```
-------------------------------------------------------------------------------
-- mask1_slots.lua    Please edit this file in order to define your logic
-------------------------------------------------------------------------------
                       -- here you may define variables local for your mask
                       -- also see the variables in the generated maskX.lua
inp = {}               -- declare some memory for inputs
inp[1] = rllib.rlPlcMem() -- these values are read in slotNullEvent
inp[2] = rllib.rlPlcMem() -- and can be used in any (other) slot

ani = rllib.rlSvgAnimator() -- class for handling a SVG

function drawSvg1(p)     -- helper function for drawing the SVG
  pv.gBeginDraw(p,svg1)
  ani.writeSocket(ani)
  pv.gEndDraw(p)
end
```

```
function slotInit(p)   -- this function will be called before the event loop starts
  pv.pvSetAlignment(p,label1,pv.AlignCenter)    -- set label text alignment
  pv.pvSetAlignment(p,label2,pv.AlignCenter)
  pv.pvSetAlignment(p,label3,pv.AlignCenter)
  pv.pvSetAlignment(p,label4,pv.AlignCenter)
  inp[1].i = mb.intValue(mb,"coilStatus(1,0)")  -- read modbus values
  inp[2].i = mb.intValue(mb,"coilStatus(1,8)")
  if inp[1].isSet(inp[1],rllib.BIT4) == 1       then -- init label1
    pv.pvSetPaletteBackgroundColor(p,label1,255,0,0)
  else
    pv.pvSetPaletteBackgroundColor(p,label1,0,255,0)
  end
  if inp[1].isSet(inp[1],rllib.BIT5) == 1       then -- init label2
    pv.pvSetPaletteBackgroundColor(p,label2,255,0,0)
  else
    pv.pvSetPaletteBackgroundColor(p,label2,0,255,0)
  end
  if inp[1].isSet(inp[1],rllib.BIT6) == 1       then -- init label3
    pv.pvSetPaletteBackgroundColor(p,label3,255,0,0)
  else
    pv.pvSetPaletteBackgroundColor(p,label3,0,255,0)
  end
  if inp[1].isSet(inp[1],rllib.BIT7) == 1       then -- init label4
    pv.pvSetPaletteBackgroundColor(p,label4,255,0,0)
  else
    pv.pvSetPaletteBackgroundColor(p,label4,0,255,0)
  end
  pv.pvSetPaletteBackgroundColor(p,button1,0,255,0) -- show all button in green
  pv.pvSetPaletteBackgroundColor(p,button2,0,255,0)
  pv.pvSetPaletteBackgroundColor(p,button3,0,255,0)
  pv.pvSetPaletteBackgroundColor(p,button4,0,255,0)
  ani.setId(ani,svg1)                           -- load and draw a test SVG
  ani.setSocket(ani,pv.pvGetSocketPointer(p))
  ani.read(ani,"test1.svg")
  drawSvg1(p)
  -- read a mysql table and show it on screen
  qtdb.query(qtdb,p,"select␣*from␣tables")
  qtdb.populateTable(qtdb,p,table1)
  return 0
end

function slotNullEvent(p)
  inp[1].i_old = inp[1].i -- read new input values from modbus slave=1
  inp[2].i_old = inp[2].i -- inp may be used in all slot functions
  inp[1].i = mb.intValue(mb,"coilStatus(1,0)")
  inp[2].i = mb.intValue(mb,"coilStatus(1,8)")

  -- update color of label if input value changes
  -- and do some outputs within the SVG
  if inp[1].hasBeenSet(inp[1],rllib.BIT4) == 1      then
     pv.pvSetPaletteBackgroundColor(p,label1,255,0,0)
     ani.svgTextPrintf(ani,"text1", "bit4=1")       end
  if inp[1].hasBeenCleared(inp[1],rllib.BIT4) == 1  then
     pv.pvSetPaletteBackgroundColor(p,label1,0,255,0)
     ani.svgTextPrintf(ani,"text1", "bit4=0")       end

  if inp[1].hasBeenSet(inp[1],rllib.BIT5) == 1      then
     pv.pvSetPaletteBackgroundColor(p,label2,255,0,0)
     ani.svgTextPrintf(ani,"text1", "bit5=1")       end
  if inp[1].hasBeenCleared(inp[1],rllib.BIT5) == 1  then
     pv.pvSetPaletteBackgroundColor(p,label2,0,255,0)
     ani.svgTextPrintf(ani,"text1", "bit5=0")       end
```

```
  if inp[1].hasBeenSet(inp[1],rllib.BIT6) == 1      then
     pv.pvSetPaletteBackgroundColor(p,label3,255,0,0)
     ani.svgTextPrintf(ani,"text1", "bit6=1")       end
  if inp[1].hasBeenCleared(inp[1],rllib.BIT6) == 1  then
     pv.pvSetPaletteBackgroundColor(p,label3,0,255,0)
     ani.svgTextPrintf(ani,"text1", "bit6=0")       end

  if inp[1].hasBeenSet(inp[1],rllib.BIT7) == 1      then
     pv.pvSetPaletteBackgroundColor(p,label4,255,0,0)
     ani.svgTextPrintf(ani,"text1", "bit7=1")       end
  if inp[1].hasBeenCleared(inp[1],rllib.BIT7) == 1  then
     pv.pvSetPaletteBackgroundColor(p,label4,0,255,0)
     ani.svgTextPrintf(ani,"text1", "bit7=0")       end

  if inp[1].intChanged(inp[1]) == 1 or inp[2].intChanged(inp[2]) == 1 then
     drawSvg1(p)
     end

  return 0
end

function slotButtonEvent(p,id)
  return 0
end

function slotButtonPressedEvent(p,id)
  -- write some outputs to modbus
  -- and do some outputs within the SVG
  if    (id == button1) then
    pv.pvSetPaletteBackgroundColor(p,button1,255,0,0)
    mb.writeIntValue(mb,"coil(1,0)",1)
    ani.show(ani,"PV.circle1",0)
    drawSvg1(p)
  elseif(id == button2) then
    pv.pvSetPaletteBackgroundColor(p,button2,255,0,0)
    mb.writeIntValue(mb,"coil(1,1)",1)
    ani.show(ani,"pv.monitor1",0)
    drawSvg1(p)
  elseif(id == button3) then
    pv.pvSetPaletteBackgroundColor(p,button3,255,0,0)
    mb.writeIntValue(mb,"coil(1,2)",1)
    ani.svgTextPrintf(ani,"text1", "Hello")
    drawSvg1(p)
  elseif(id == button4) then
    pv.pvSetPaletteBackgroundColor(p,button4,255,0,0)
    mb.writeIntValue(mb,"coil(1,3)",1)
    ani.svgTextPrintf(ani,"text1", "World")
    drawSvg1(p)
  end
  return 0
end

function slotButtonReleasedEvent(p,id)
  -- write some outputs to modbus
  -- and do some outputs within the SVG
  if    (id == button1) then
    pv.pvSetPaletteBackgroundColor(p,button1,0,255,0)
    mb.writeIntValue(mb,"coil(1,0)",0)
    ani.show(ani,"PV.circle1",1)
    drawSvg1(p)
  elseif(id == button2) then
    pv.pvSetPaletteBackgroundColor(p,button2,0,255,0)
    mb.writeIntValue(mb,"coil(1,1)",0)
```

```
    ani.show(ani,"pv.monitor1",1)
    drawSvg1(p)
  elseif(id == button3) then
    pv.pvSetPaletteBackgroundColor(p,button3,0,255,0)
    mb.writeIntValue(mb,"coil(1,2)",0)
  elseif(id == button4) then
    pv.pvSetPaletteBackgroundColor(p,button4,0,255,0)
    mb.writeIntValue(mb,"coil(1,3)",0)
  end
  return 0
end

function slotTextEvent(p,id,text)
  return 0
end

function slotSliderEvent(p,id,val)
  return 0
end

function slotCheckboxEvent(p,id,text)
  return 0
end

function slotRadioButtonEvent(p,id,text)
  return 0
end

function slotGlInitializeEvent(p,id)
  return 0
end

function slotGlPaintEvent(p,id)
  return 0
end

function slotGlResizeEvent(p,id,width,height)
  return 0
end

function slotGlIdleEvent(p,id)
  return 0
end

function slotTabEvent(p,id,val)
  return 0
end

function slotTableTextEvent(p,id,x,y,text)
  return 0
end

function slotTableClickedEvent(p,id,x,y,button)
  return 0
end

function slotSelectionEvent(p,id,val,text)
  return 0
end

function slotClipboardEvent(p,id,val)
  return 0
end
```

```
function slotRightMouseEvent(p,id,text)
  return 0
end

function slotKeyboardEvent(p,id,val,modifier)
  return 0
end

function slotMouseMovedEvent(p,id,x,y)
  return 0
end

function slotMousePressedEvent(p,id,x,y)
  return 0
end

function slotMouseReleasedEvent(p,id,x,y)
  return 0
end

function slotMouseOverEvent(p,id,enter)
  return 0
end

function slotUserEvent(p,id,text)
  return 0
end
```

In den 'slotFunktionen' kodiert der Programmierer der Visualisierung die Logik des pvserver.
Im Beispiel wird eine Anbindung an Modbus über unser Shared Memory und die Mailbox gezeigt. Ausserdem ist dort eine Datenbank abfrage enthalten. Die Kommentare im Programmcode sollten ausreichend sein, um den Code zu verstehen.

6.5.4 modbus.ini

INI Datei für den modbus_client daemon

```
# ini file for modbus_client
#
# USE_SOCKET := 1 | 0 # if 0 then USE_TTY
# DEBUG     := 1 | 0
# BAUDRATE  := 300   |
#              600   |
#              1200  |
#              1800  |
#              2400  |
#              4800  |
#              9600  |
#              19200 |
#              38400 |
#              57600 |
#              115200
# PARITY    := NONE | ODD | EVEN
# CYCLE<N>  := <count>,<name>
# name      := coilStatus(slave,adr)    |
#              inputStatus(slave,adr)   |
#              holdingRegisters(slave,adr) |
#              inputRegisters(slave,adr)
# CYCLETIME in milliseconds
# SHARED_MEMORY_SIZE must be equal to SHARED_MEMORY_SIZE of pvserver
# MAX_NAME_LENGTH is maximum length of variable name in shared memory
#
```

```
[GLOBAL]
USE_SOCKET=0
DEBUG=0
CYCLETIME=100

[SOCKET]
IP=192.168.1.103
PORT=502

[TTY]
DEVICENAME=/dev/ttyUSB0
BAUDRATE=9600
RTSCTS=1
PARITY=NONE

[RLLIB]
MAX_NAME_LENGTH=30
SHARED_MEMORY=/srv/automation/shm/modbus1.shm
SHARED_MEMORY_SIZE=65536
MAILBOX=/srv/automation/mbx/modbus1.mbx

[CYCLES]
NUM_CYCLES=1
CYCLE1=16,coilStatus(1,0)
```

Der daemon 'modbus_client' benötigt eine INI Datei, in der definiert wird, was von Modbus zyklisch gelesen werden soll, wie Shared Memory und Mailbox heißen und über welche Schnittstelle kommuniziert werden soll.

modbus_client starten

```
modbus_client modbus.ini
```

6.6 Python

Für Python wird eine Sprachenanbindung zu den 'Util Funktionen' und zur rllib, die mit Hilfe von Swig `http://swig.org` generiert wurde, zur Verfügung gestellt. Damit können auch Python Programmierer diese Bibliotheken verwenden.
In pvdevelop kann man beim Anlegen eines neuen pvserver auswählen, dass man ihn in Python programmieren möchte. Dann wird pvdevelop ein Rahmenprogramm in C++ erstellen, das die 'slot Funktionen' als Python Klasse einbindet. Der Programmierer kann diese generierten Python Klassen mit den 'slot Funktionen' dann ausfüllen und dabei die oben genannten Bibliotheken nutzen.

slot Funktionen in Python

```
### pvbrowser slots written in python ##################
### begin of generated area, do not edit ###############
import pv, rllib

class mask1:

  p = pv.PARAM()
  # our mask contains the following objects
  ID_MAIN_WIDGET = 0
  obj1 = 1
  ID_END_OF_WIDGETS = 2

  toolTip = [
    '',
    '',
    '' ]

  whatsThis = [
```

```
  '',
  '',
  '' ]

widgetType = [
  '',
  'TQPushButton',
  '' ]

### end of generated area, do not edit ###############

I = 0

def slotInit(self, s):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotNullEvent(self, s):
  self.p.s = self.p.os = s # set socket must be the first command
  ret = pv.pvPrintf(self.p,self.obj1,'hello'+str(self.I))
  self.I = self.I + 1
  return 0

def slotButtonEvent(self, s, id):
  self.p.s = self.p.os = s # set socket must be the first command
  if id == self.obj1:
    self.I = 0
    ret = pv.pvPrintf(self.p,self.obj1,'reset'+str(self.I))
    print 'reset␣I␣=␣0'
  return 0

def slotButtonPressedEvent(self, s, id):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotButtonReleasedEvent(self, s, id):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotTextEvent(self, s, id, text):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotSliderEvent(self, s, id, val):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotCheckboxEvent(self, s, id, text):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotRadioButtonEvent(self, s, id, text):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotGlInitializeEvent(self, s, id):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotGlPaintEvent(self, s, id):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0
```

```
def slotGlResizeEvent(self, s, id, width, height):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotGlIdleEvent(self, s, id):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotTabEvent(self, s, id, val):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotTableTextEvent(self, s, id, x, y, text):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotTableClickedEvent(self, s, id, x, y, button):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotSelectionEvent(self, s, id, val, text):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotClipboardEvent(self, s, id, val):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotRightMouseEvent(self, s, id, text):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotKeyboardEvent(self, s, id, val, modifier):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotMouseMovedEvent(self, s, id, x, y):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotMousePressedEvent(self, s, id, x, y):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotMouseReleasedEvent(self, s, id, x, y):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotMouseOverEvent(self, s, id, enter):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0

def slotUserEvent(self, s, id, text):
  self.p.s = self.p.os = s # set socket must be the first command
  return 0
```

6.7 Widgets

In den folgenden Unterpunkten werden wir auf die Programmierung der einzelnen Widgets eingehen. Die folgenden 'Util Funktionen' gelten für alle Arten von Widgets.

- *int pvSetWhatsThis(PARAM *p, int id, const char *text);*

- *int pvWhatsThisPrintf(PARAM *p, int id, const char *format, ...);*
- *int pvToolTip(PARAM *p, int id, const char *text);*
- *int pvSetGeometry(PARAM *p, int id, int x, int y, int w, int h);*
- *int pvSetMinSize(PARAM *p, int id, int w, int h);*
- *int pvSetMaxSize(PARAM *p, int id, int w, int h);*
- *int pvMoveCursor(PARAM *p, int id, int cursor);*
- *int pvResize(PARAM *p, int id, int w, int h);*
- *int pvHide(PARAM *p, int id);*
- *int pvShow(PARAM *p, int id);*
- *int pvSetPaletteBackgroundColor(PARAM *p, int id, int r, int g, int b);*
- *int pvSetPaletteForegroundColor(PARAM *p, int id, int r, int g, int b);*
- *int pvSetFontColor(PARAM *p, int id, int r, int g, int b);*
- *int pvSetFont(PARAM *p, int id, const char *family, int pointsize, int bold, int italic , int underline, int strikeout);*
- *int pvSetEnabled(PARAM *p, int id, int enabled);*
- *int pvCopyToClipboard(PARAM *p, int id);*
- *int pvSaveAsBmp(PARAM *p, int id, const char *filename);*
- *int pvSetAlignment(PARAM *p, int id, int alignment);*
- *int pvSetText(PARAM *p, int id, const char *text);*
- *int pvPrintf(PARAM *p, int id, const char *format, ...);*
- *int pvSetBackgroundColor(PARAM *p, int id, int r, int g, int b);*
- *int pvText(PARAM *p, int id);*

Die weiteren 'Util Funktionen' beziehen sich dann jeweils auf spezielle Widgets. Diese finden Sie unter 'Construction' in der Hilfe von pvslib unter jedem Widget.

6.7.1 PushButton

PushButtons können ein optionales Icon enthalten.

Abbildung 6.1: PushButton

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.2 RadioButton

Mehrere RadioButtons schalten exklusiv, wenn sie ein gemeinsames Eltern Widget haben.

Abbildung 6.2: RadioButton

Sie liefern Ereignisse in folgenden Slots.

- *static int slotRadioButtonEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.3 CheckBox

CheckBoxen können angehakt werden.

Abbildung 6.3: CheckBox

Sie liefern Ereignisse in folgenden Slots.

- *static int slotCheckboxEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.4 Label

Labels dienen zur Ausgabe einer Zeile Text.

Abbildung 6.4: Label

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.5 LineEdit

LineEdits dienen zur Eingabe einer Zeile Text.

Abbildung 6.5: LineEdit

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.6 MultiLineEdit

MultiLineEdits dienen zur Eingabe mehrerer Zeilen Text. Das MultiLineEdit kann auch auf 'read only' gesetzt werden.

Abbildung 6.6: MultiLineEdit

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotClipboardEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.7 ComboBox

ComboBoxen dienen zur Auswahl einer Option aus mehreren. ComboBoxen können auch editierbar sein.

Abbildung 6.7: ComboBox

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*

- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.8 LCDNumber

Mit LCDNumber können Zahlenwerte als LCD Anzeige ausgegeben werden.

Abbildung 6.8: LCDNumber

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.9 Slider

Slider dienen zur Eingabe von Werten über einen Schieberegler.

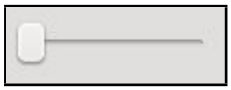

Abbildung 6.9: Slider

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.10 Frame

Mit einem Frame können andere Widgets mit Rahmen umgeben und gruppiert werden.

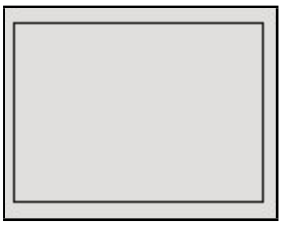

Abbildung 6.10: Frame

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.11 GroupBox

Eine GroupBox gruppiert mehrere Widgets und versieht sie mit einem Titel.

Abbildung 6.11: GroupBox

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.12 ToolBox

Eine ToolBox besteht aus mehreren übers Tabs auswählbaren Bereichen, die vom Benutzer umgeschaltet werden können. In den verschiedenen Bereichen können dann thematisch zugeordnete Widgets eingefügt werden.

Abbildung 6.12: ToolBox

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTabEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.13 TabWidget

Ein TabWidget stellt mehrere Reiter zur Organisation von Unterfenstern zur Verfügung, über die der Benutzer die Unterfenster auswählen kann.

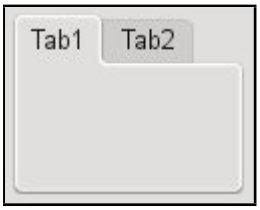

Abbildung 6.13: TabWidget

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTabEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.14 ListBox

Eine ListBox stellt eine Auswahl mehrerer Einträge zur Verfügung.

Abbildung 6.14: ListBox

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotSelectionEvent(PARAM *p, int id, DATA *d, int val, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.15 Table

Eine Table dient zur Ein- und Ausgabe von tabellarischen Daten.

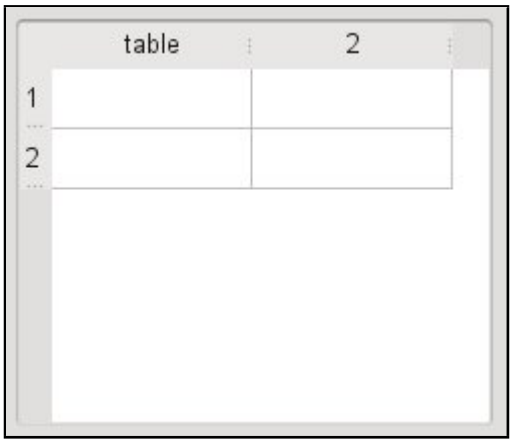

Abbildung 6.15: Table

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTableTextEvent(PARAM *p, int id, DATA *d, int x, int y, const char *text)*
- *static int slotTableClickedEvent(PARAM *p, int id, DATA *d, int x, int y, int button)*
- *static int slotKeyboardEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.16 SpinBox

Eine SpinBox dient zur Ein- und Ausgabe von Werten mit Begrenzung auf minimal und maximal Wert.

Abbildung 6.16: SpinBox

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.17 Dial

Ein Dial dient zur Ein- und Ausgabe von Werten als Drehknopf.

Abbildung 6.17: Dial

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.18 Line

Eine Line dient zur graphischen Organisation.

Abbildung 6.18: Line

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.19 ProgressBar

Ein ProgressBar dient als Fortschrittsanzeige.

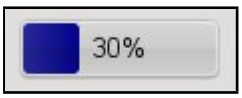

Abbildung 6.19: ProgressBar

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.20 ListView

Ein ListView implementiert eine baumartige Anzeige.

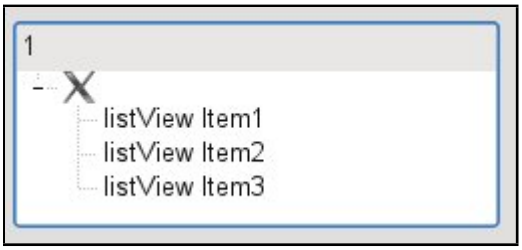

Abbildung 6.20: ListView

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSelectionEvent(PARAM *p, int id, DATA *d, int val, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.21 IconView

Ein IconView erlaubt die Auswahl über Icons.

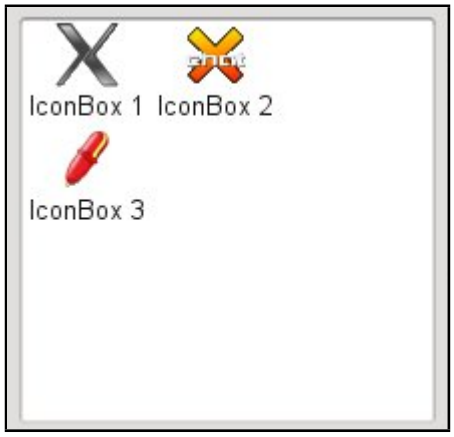

Abbildung 6.21: IconView

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.22 TextBrowser/WebKit

TextBrowser erlaubt den Einbau beliebiger html Dateien oder Webseiten über eine http URL mit Hilfe von WebKit. Beim Anklicken von Hyperlinks wird ein TextEvent mit der URL geliefert.

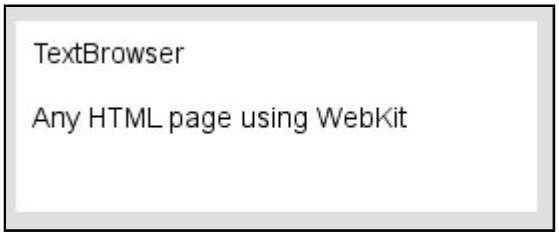

Abbildung 6.22: TextBrowser/WebKit

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*

- *static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.23 DateTimeEdit

Über DateTimeEdit können Datum und Uhrzeit eingegeben werden.

Abbildung 6.23: DateTimeEdit

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.24 DateEdit

Über DateEdit kann das Datum eingegeben werden.

Abbildung 6.24: DateEdit

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.25 TimeEdit

Über TimeEdit kann die Uhrzeit eingegeben werden.

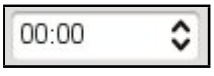

Abbildung 6.25: TimeEdit

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.26 QwtThermo

Ein QwtThermo kann als Analog anzeige verwendet werden.

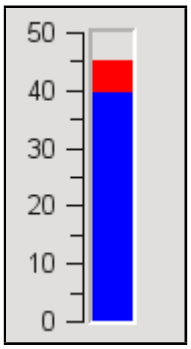

Abbildung 6.26: QwtThermo

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

Die Qwt Widgets liefern float Werte. Um diese auswerten zu können, muss man die Ereignisschleife der Maske erweitern.

```
case SLIDER_EVENT:
  sscanf(text,"(%d)",&val);
  if(trace) printf("SLIDER_EVENT␣val=%d\n",val);
  if((ret=slotSliderEvent(p,i,&d,val)) != 0) return ret;
    // Im slotSliderEvent wuerden Sie nur die Vorkommastelle erhalten.
  if((ret=slotTextEvent(p,i,&d,text)) != 0) return ret;
    // event auch an slotTextEvent weiterreichen und
    // darin den float Wert auslesen
    // sscanf(text,"(%f)",&fval);
  break;
```

6.7.27 QwtKnob

Ein QwtKnob dient zur Ein- und Ausgabe von Werten als Drehknopf.

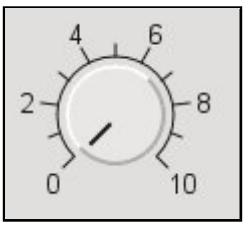

Abbildung 6.27: QwtKnob

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.28 QwtCounter

Ein QwtCounter dient zur Ein- und Ausgabe von Werten mit minimal und maximal Wert.

Abbildung 6.28: QwtCounter

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.29 QwtWheel

Ein QwtWheel dient zur Eingabe von Werten.

Abbildung 6.29: QwtWheel

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.30 QwtSlider

Ein QwtSlider dient zur Ein- und Ausgabe von Werten mit minimal und maximal Wert.

Abbildung 6.30: QwtSlider

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.31 QwtDial

Ein QwtDial dient als analoge Anzeige

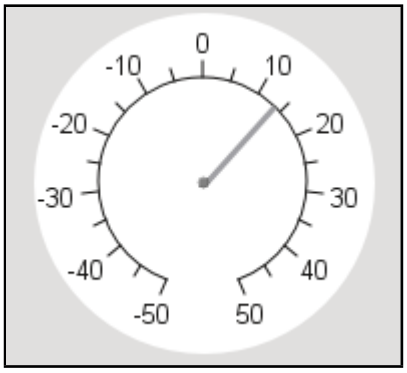

Abbildung 6.31: QwtDial

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.32 QwtAnalogClock

Ein QwtAnalogClock dient als Uhrzeitanzeige.

Abbildung 6.32: QwtAnalogClock

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.33 QwtCompass

Ein QwtCompass dient zur Ein- und Ausgabe von Werten über eine Kompass Nadel.

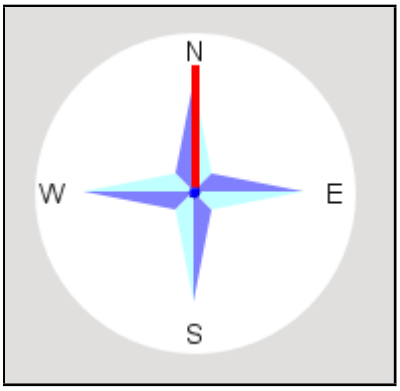

Abbildung 6.33: QwtCompass

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.7.34 Custom Widgets in pvbrowser

pvbrowser bietet einen reichhaltigen Satz von Standard Widgets. Es ist aber ebenfalls möglich eigene Widgets hinzuzufügen. Diese Custom Widgets werden in Shared Object Bibliotheken definiert. Auf Unix ähnlichen Systemen werden Shared Object Bibliotheken mit libNAME.so benannt. Auf OS-X lautet der Name libNAME.dylib. Auf Windows heißen sie NAME.dll.

Ein Beispiel zu solchen Custom Widgets kann in pvbaddon.tar.gz gefunden werden.

Verzeichnis: pvbaddon/custom_widgets/embedded_widgets/embedded_widgets_1

Abbildung 6.34: Schnappschuss der ewidgets

Wie Benutzer Custom Widgets Plugins erhalten und installieren können

Die obigen Custom Widgets können in Binärform aus dem Dowload Bereich unserer Webseite geladen werden. Bitte speichern sie die Datei dann unter folgendem Verzeichnis auf ihrem Computer / Mobilgerät.

Verzeichnis wo die Plugins gespeichert werden

```
Unix hnliche Systeme: /opt/pvbrowser/libewidgets.so
OS-X:                 /opt/pvbrowser/libewidgets.dylib
Windows:              \$PVBDIR\pvbrowser\ewidgets.dll
Android:              /sdcard/pvbrowser/libewidgets.so
Symbian:              \sys\bin\ewidgets.dll # Bitte achten Sie auf den exakten Ort fr die DLL.
                                            # Das sys Verzeichnis ist versteckt und oder geschtzt.
```

Der oben angegebene Pfad wird unter den Optionen von pvbrowser definiert.
pvb_widget_plugindir=/opt/pvbrowser
Damit ist es auch möglich einen anderen Pfad für das plugindir zu verwenden.
Zur Installation lädt der Benutzer die binäre Datei herunter und speichert sie in dem oben angegebenen Verzeichnis. Wenn Sie einen pvserver aufrufen, der ein solches Custom Widget verwendet, Ihr pvbrowser diese Widgets aber noch nicht kennt, erhalten sie Eine Meldung mit entsprechenden Anweisungen.

Wie die Widget Plugins benutzt werden

Im Designer Modus von pvdevelop wählen Sie “Custom Widget” zum Einfügen und setzen die whatsThis Eigenschaft auf “/libname/widgettype:arguments”.

Im Falle des obigen ewidgets Plugin sind folgende Kombinationen möglich

```
"/ewidgets/myQtSvgDialGaugeThermometer:arguments"
"/ewidgets/myQtScrollWheel:arg1,arg2"
"/ewidgets/myQtScrollDial:arguments"
"/ewidgets/myQtMultiSlider:arguments"
"/ewidgets/myQtScrollDial:arguments"
"/ewidgets/myQtSvgDialGaugeTachometer:arguments"
"/ewidgets/myQtSvgDialGaugeAmperemeter:arguments"
"/ewidgets/myQtScrollDial:arguments"
"/ewidgets/myQtSvgSlideSwitch:arguments"
"/ewidgets/myQtSvgToggleSwitch:arguments"
"/ewidgets/myQtSvgButtonBerylSquare:arguments"
"/ewidgets/myQtSvgButtonBeryl:"
"/ewidgets/myQt5WayButton"
"/ewidgets/myQtScrollDial"
```

Beachten Sie, dass libname weder das führende „lib“ noch die abschließende Dateierweiterung “.so ‖*.dylib*‖*.dll*” enthält.
Die vom Plugin zur Verfügung gestellten widgettypes müssen aus der Dokumentation des Plugins hervorgehen (siehe oben).
Im obigen Beispiel werden keine sinnvollen arguments verwendet. Ewidgets benutzt keine arguments. Andere Custom Widgets könnten dies aber verlangen.
Die Standard Funktionen wie z.B. pvSetGeometry() können auf alle Custom Widgets angewandt werden. Funktionen, die speziell für ein Custom Widget sind könnten mit Hilfe von pvSetText() umgesetzt werden. Die Custom Widgets würden die Konvention “falls der text mit einem @ beginnt”, dann interpretiere den folgenden text als ein spezielles Kommando für das Custom Widget.
Einen Beispiel pvserver, der die ewidgets Plugins verwendet findet man unter
“pvbaddon/custom_widgets/embedded_widgets/pvsEmbeddedWidgetsSample”

Wie man eigene Widget Plugins entwickelt

Zunächst entwickelt man ganz normale Qt Widgets, wie das in der Qt Dokumentation beschrieben wird und testet sie als normale Bibliothek. Dann benötigt man etwas Klebstoff (Code), um aus den Widgets ein Plugin zu machen, dass zur Laufzeit von pvbrowser geladen werden kann.
Siehe:
pvbaddon/custom_widgets/embedded_widgets/embedded_widgets_1/mywidgets.h
pvbaddon/custom_widgets/embedded_widgets/embedded_widgets_1/mywidgets.cpp
Das “myQtMultiSlider” Widget Plugin wird z.B. von dem normalen “QtMultiSlider” Widget abgeleitet. Das sieht folgendermaßen aus.

Klasse zur Definition eines Custom Widgets

```
#include <QtMultiSlider>
class myQtMultiSlider: public QtMultiSlider
{
  Q_OBJECT
  Q_PROPERTY(int value READ value WRITE setValue)
  public:
    myQtMultiSlider(const char * name, int *_sock, int _ident, QWidget * parent, const char *arg);
    ~myQtMultiSlider();
    virtual bool event(QEvent *event);
  public slots:
    void onTopChanged(int newval);
    void onBottomChanged(int newval);
  protected:
    int *sock;
    int ident;
};
```

Wir haben die protected “int *sock” und “int ident” Variablen hinzugefügt, die mit dem von pvbrowser verwendeten Socket und der id (ident) korrespondieren. Dann haben wir eine “virtual bool event(QEvent *event)” Methode und ein paar Slot Funktionen hinzugefügt.

Definition eines pvb Ereignisses

```
const QEvent::Type pvbEventNumber = (QEvent::Type) (QEvent::User + 1);

class PvbEvent : public QEvent
{
public:
  PvbEvent(const char *_command, QString _param, QEvent::Type _event=pvbEventNumber);
  virtual ~PvbEvent();
  const char *command;
  QString   param;
};
```

Immer, wenn ein Kommando vom pvserver an Ihr Custom Widget gesendet wird, enthält die Botschaft 2 Texte, die man interpretieren kann. Wir sehen uns dazu die Implementierung von “myQtMultiSlider” an, um zu sehen, wie das gemacht wird.

Implementierung des Konstruktors

```
myQtMultiSlider::myQtMultiSlider(const char * name, int *_sock, int _ident, QWidget * parent, const
    char *arg )
              :QtMultiSlider(parent)
{
  Q_UNUSED(arg);
  sock = _sock;
  ident = _ident;
  if(name) setObjectName(name);
  //default skin
  setSkin("Beryl");
  connect(topSlider(),SIGNAL(sliderMoved(int)),this,SLOT(onTopChanged(int)));
  connect(bottomSlider(),SIGNAL(sliderMoved(int)),this,SLOT(onBottomChanged(int)));
}
```

Dem Konstruktor wird der Objekt name des Widget, der Netzweck sock von pvbrowser und die ident (id), die in pvbrowser verwendet wird, übergeben. Optional kann das arg welches Sie bei der Eingabe Ihres Widgets im graphischen Designer von pvdevelop angegeben haben, ausgewertet werden. In diesem Beispiel wird das arg ignoriert. Dann verbinden wir die slotFunktionen, die wir verwenden wollen, mit den entsprechenden Signalen.

Slot Funktionen

```
void myQtMultiSlider::onTopChanged(int v)
{
  sprintf(tmp,"user(%d,\"%d;%s\")\n",ident,v,"TOP");
  tcp_send(sock,tmp,strlen(tmp));
}

void myQtMultiSlider::onBottomChanged(int v)
{
  sprintf(tmp,"user(%d,\"%d;%s\")\n",ident,v,"BOTTOM");
  tcp_send(sock,tmp,strlen(tmp));
}
```

In den Slot Funktionen senden wir den text für das entsprechende Ereignis an den Server. In unserem Beispiel verwenden wir ein „user“ Ereignis.

Event handling

```
bool myQtMultiSlider::event(QEvent *event)
{
  if(event->type() == pvbEventNumber)
  {
    int i, val;
```

```
    PvbEvent *e=(PvbEvent *)event;
    //printf("event->command=%s event->param=%s\n", e->command, (const char *) e->param.toUtf8());
    if(strncmp(e->command,"setValue(",9) == 0)
    {
      sscanf(e->command, "setValue(%d,%d)", &i, &val);
      //printf("%s\n", e->command);
      setValue(val);
    }
    else if(strncmp(e->command, "setMinValue(", 12) == 0)
    {
      sscanf(e->command, "setMinValue(%d,%d)", &i, &val);
      //printf("%s\n", e->command);
      setMinimum(val);
    }
    else if(strncmp(e->command, "setMaxValue(", 12) == 0)
    {
      sscanf(e->command, "setMaxValue(%d,%d)", &i, &val);
      //printf("%s\n", e->command);
      setMaximum(val);
    }
    else
    {
      printf("TODO: interpret (command,param) and call the method\n");
      printf("event->command=%s event->param=%s\n", e->command, (const char *) e->param.toUtf8());
    }
    return true;
  }
  else
  {
    return QtMultiSlider::event(event);
  }
}
```

Die Methode event behandelt alle Ereignisse.
Entweder werden die Ereignisse durch Benutzerinteraktionen ausgelöst oder indem eine Botschaft vom pvserver an den pvbrowser Client gesendet wurde. Falls das Ereignis vom pvserver gesendet wurde, hat es den Typ pvbEventNumber. Hier interpretieren wir den Text, welcher das Ereignis beschreibt und rufen die gewünschte Funktion auf. Es ist lediglich notwendig die Ereignisse zu behandeln, die speziell für unser Widget sind. Alle Ereignisse, die auf alle Widgets anwendbar sind, die von QWidget abgeleitet wurden, benötigen keine spezielle Behandlung, da dies bereits in pvbrowser ausgewertet wird. Alle Ereignisse, die durch Benutzerinteraktionen ausgelöst werden, werden im „else“ Zweig behandelt.
Die Export Funktionen der dynamisch ladbaren Bibliothek sehen wie folgt aus.

Export Funktionen

```
//for pvdevelop, when load this library
// TODO: add icons, info, link
extern "C" const char *listWidgets(){
    return "myButton"
           ",myQt5WayButton"
           ",myQtBasicDialGauge"
           ",myQtSvgDialGaugeTachometer"
           ",myQtSvgDialGaugeAmperemeter"
           ",myQtSvgDialGaugeThermometer"
           ",myQtBasicGraph"
           ",myQtMultiSlider"
           ",myQtScrollDial"
           ",myQtScrollWheel"
           ",myQtSvgButtonBeryl"
           ",myQtSvgButtonBerylSquare"
           ",myQtSvgSlideSwitch"
           ",myQtSvgToggleSwitch";
}
```

```
//for pvbrowser
//set tcp send pointer to app function
//when load library
extern "C" void setTcpSend(int (*_tcp_send)(int *s, const char *msg, int len)){
  if(_tcp_send)
    tcp_send=_tcp_send;
}

//for pvbrowser and pvdevelop
// This our export function
extern "C" QWidget *new_pvbCustomWidget(const char *name, int *_sock,int _ident, QWidget *parent,
    const char *arg)
{
  //widget name is class_name + '_' + ident
  char wname[40];
  sprintf(wname,"%s_%d",name,_ident);

  if(strcmp(name,"myButton") == 0)
  {
    return new myButton(wname, _sock,_ident, parent);
  }
  // TODO: add more custom widgets
  else if(strcmp(name,"myQtSvgDialGauge") == 0)
    return new myQtSvgDialGauge(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtSvgDialGaugeTachometer") == 0)
    return new myQtSvgDialGaugeTachometer(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtSvgDialGaugeAmperemeter") == 0)
    return new myQtSvgDialGaugeAmperemeter(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtSvgDialGaugeThermometer") == 0)
    return new myQtSvgDialGaugeThermometer(wname, _sock,_ident, parent, arg);

  else if(strcmp(name,"myQt5WayButton") == 0)
    return new myQt5WayButton(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtBasicDialGauge") == 0)
    return new myQtBasicDialGauge(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtBasicGraph") == 0)
    return new myQtBasicGraph(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtMultiSlider") == 0)
    return new myQtMultiSlider(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtScrollDial") == 0)
    return new myQtScrollDial(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtScrollWheel") == 0)
    return new myQtScrollWheel(wname, _sock,_ident, parent, arg);

  //removed and added 2 inherited
  //else if(strcmp(name,"myQtSvgButton") == 0)
  //  return new myQtSvgButton(wname, _sock,_ident, parent, arg);

  else if(strcmp(name,"myQtSvgButtonBeryl") == 0)
    return new myQtSvgButtonBeryl(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtSvgButtonBerylSquare") == 0)
    return new myQtSvgButtonBerylSquare(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtSvgButtonMetallicBrush") == 0)
    return new myQtSvgButtonMetallicBrush(wname, _sock,_ident, parent, arg);

  else if(strcmp(name,"myQtSvgSlideSwitch") == 0)
    return new myQtSvgSlideSwitch(wname, _sock,_ident, parent, arg);
  else if(strcmp(name,"myQtSvgToggleSwitch") == 0)
    return new myQtSvgToggleSwitch(wname, _sock,_ident, parent, arg);
  else
    return NULL;
}
```

Die Export Funktionen sind von Type extern “C”.
Wenn pvbrowser das Widget Plugin lädt, erwartet er, dass sich die obigen Export Funktionen auflösen lassen.

6.8 Grafiken

pvbrowser beherrscht mehrere Möglichkeiten zur Ausgabe von Grafiken.
Zunächst können einfache Bitmap Grafiken in den gängigen Dateiformaten ausgegeben werden, die von Qt unterstützt werden. Es können XY Grafiken erstellt werden, mit denen man z.B. Messkurven darstellen kann. Vektor orientierte Grafiken werden über das SVG Format eingebunden. SVG kann in pvbrowser auch dynamisch animiert werden und es können Ereignisse generiert werden, wenn der Benutzer graphische Objekte in SVG anklickt. Externe Plot Werkzeuge, wie z.B. gnuplot können in pvbrowser einfach eingebunden werden, wenn sie in der Lage sind Bitmap Grafiken als Ausgabe zu erstellen. Es ist sogar möglich 3D Grafiken in pvbrowser zu integrieren. Dazu besteht einerseits die Möglichkeit OpenGL zu verwenden. Damit können z.B. von Autocad erzeugte DWF Dateien in pvbrowser eingebunden und animiert werden. Andererseits ist auch das Visualization Toolkit VTK `http://vtk.org/` in pvbrowser benutzbar. Dabei werden die Scripte für die VTK Visualisierung in TCL an den pvbrowser Client geschickt. Beachten Sie bitte, dass VTK standardmäßig nicht in pvbrowser mit compiliert wird. Wenn man VTK benutzen möchte, muss man VTK installieren, die Datei pvb/pvbrowser/pvbrowser.pro editieren, darin den Kommentar vor CONFIG += USE_VTK entfernen und neu compilieren.

6.8.1 Bitmap Grafiken

Bitmap Grafiken können mit JPG, PNG, GIF, TIFF und BMP Format eingebunden werden.

Abbildung 6.35: Bitmap Grafik

Setzen eines Bildes in ein Image Widget

```
pvDownloadFile(p,"filename.jpg");
pvSetImage(p,id,"filename.jpg");
```

Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

6.8.2 xy Grafiken

xy Grafiken können alternativ mit dem Draw oder QwtPlot Widget erstellt werden.
Sie liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseMovedEvent(PARAM *p, int id, DATA *d, float x, float y)*

- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

Beispiele für xy Grafiken mit dem Draw und dem QwtPlot Widget

```
typedef struct // (todo: define your data structure here)
{
  int pause, display;
  int step1, step2;
  float phi1, phi2;
  double x1[100], sin_x[100];
  double x2[100], cos_x[100];
  int tab;
  int modalInput;
}
DATA;

static int drawGraphic1(PARAM *p, int id, DATA *d)
{
  int fontsize;
  int i;
  float x1[100], sin_x[100];
  float x2[100], cos_x[100];

  for(i=0;i<100;i++)
  {
    x1[i] = (float) d->x1[i]; sin_x[i]= (float) d->sin_x[i];
    x2[i] = (float) d->x2[i]; cos_x[i]= (float) d->cos_x[i];
  }

  fontsize = 12;

  gBeginDraw   (p, id);

  gSetColor    (p, BLACK);
  gSetFont     (p, TIMES, fontsize, Normal, 0);
  gBoxWithText (p, 50, 50, 1050, 550, fontsize, "x/radiant", "sin(x),␣cos(x)", NULL);
  gXAxis       (p, 0, 1.0f, 2.0f*PI, 1);
  gYAxis       (p, -1.5f, 0.5f, 1.5f, 1);

  gSetStyle    (p, 2);
  gXGrid       (p);
  gYGrid       (p);

  gSetWidth    (p, 3);
  gSetStyle    (p, 1);
  gSetColor    (p, RED);
  gLine        (p, x1, sin_x, 100);
  gSetColor    (p, GREEN);
  gLine        (p, x2, cos_x, 100);

  fontsize = 18;
  gSetColor    (p, BLUE);
  gSetFont     (p, TIMES, fontsize, Bold, 0);
  gText        (p, 50, 50-fontsize*2, "This␣is␣a␣Diagram", ALIGN_LEFT);

  gEndDraw     (p);
  return 0;
}

static int slotInit(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  memset(d,0,sizeof(DATA));
  pvResize(p,0,1280,1024);
```

```
  pvPrintf(p,ID_TAB,"Plot");
  pvSetPixmap(p,back,"back.png");
  pvSetChecked(p,radioButton1,1);

  d->step1=1;
  d->step2=1;
  d->tab=0;
  pvDisplayFloat(p,lCDNumber1,1);
  pvSetValue(p,progressBar1,1);

  // qwt plot begin ----------------------------------------------
  qpwSetCanvasBackground(p,qwtPlot1,239,239,239);
  qpwEnableAxis(p,qwtPlot1,yLeft);
  qpwEnableAxis(p,qwtPlot1,xBottom);
  qpwSetTitle(p,qwtPlot1,"Trigonometric");

  qpwEnableOutline(p,qwtPlot1,1);
  qpwSetOutlinePen(p,qwtPlot1,GREEN);

  // legend
  qpwSetAutoLegend(p,qwtPlot1,1);
  qpwEnableLegend(p,qwtPlot1,1);
  qpwSetLegendPos(p,qwtPlot1,BottomLegend);
  qpwSetLegendFrameStyle(p,qwtPlot1,Box|Sunken);

  // axes
  qpwSetAxisTitle(p,qwtPlot1,xBottom, "Alpha");
  // qpwSetAxisScaleDraw(p,qwtPlot1,xBottom, "hh:mm:ss");
  qpwSetAxisTitle(p,qwtPlot1,yLeft, "f(Alpha)");

  // curves
  qpwInsertCurve(p,qwtPlot1, 0, "Sinus(Alpha)");
  qpwSetCurvePen(p,qwtPlot1, 0, BLUE, 3, DashDotLine);
  qpwSetCurveYAxis(p,qwtPlot1, 0, yLeft);

  qpwInsertCurve(p, qwtPlot1, 1, "Cosinus(Alpha)");
  qpwSetCurvePen(p, qwtPlot1, 1, GREEN, 3, DashDotLine);
  qpwSetCurveYAxis(p, qwtPlot1, 1, yLeft);
  // qwt plot end ------------------------------------------------
  return 0;
}

static int slotNullEvent(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  if(d->pause) return 0;
  switch(d->display)
  {
    case 0:
      memset(d->cos_x, 0, sizeof(d->cos_x));
      break;
    case 1:
      memset(d->sin_x, 0, sizeof(d->sin_x));
      break;
    case 2:
      break;
    default:
      break;
  }

  // draw qwt_plot graphic
  if(d->tab == 0)
  {
```

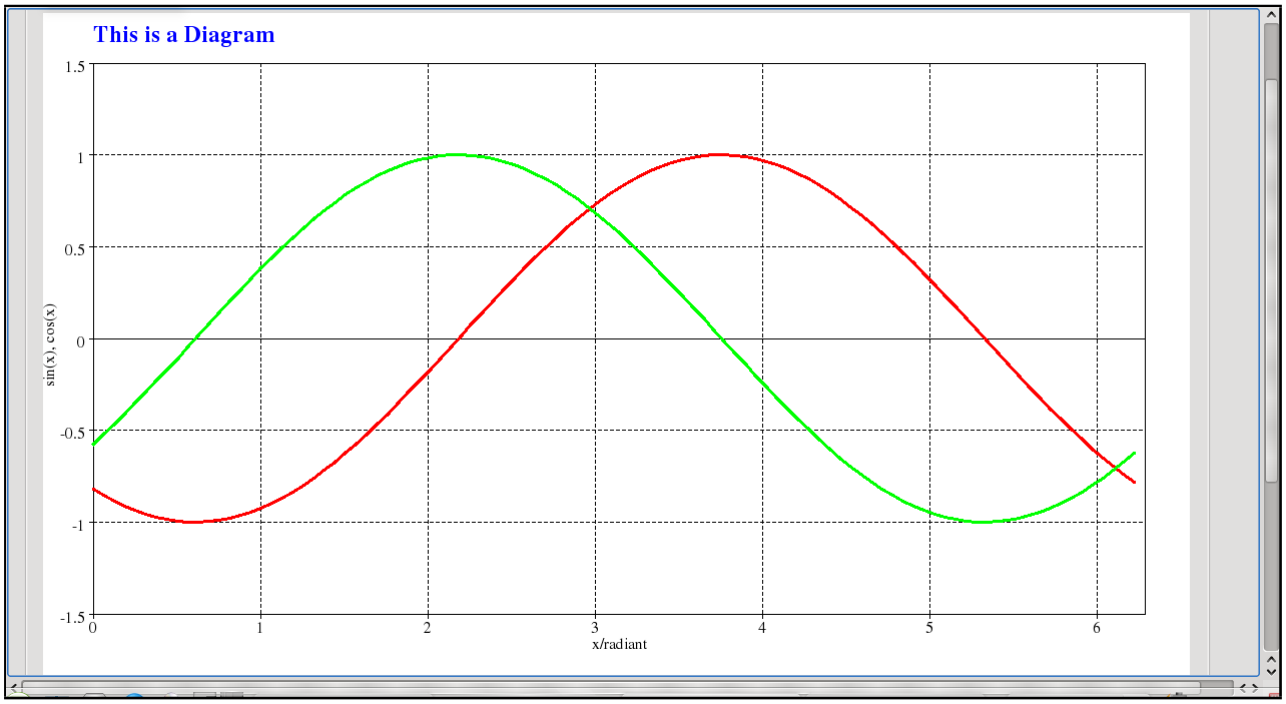

Abbildung 6.36: xy Grafik mit dem Draw Widget

```
    qpwSetCurveData(p, qwtPlot1, 0, 100, d->x1, d->sin_x);
    qpwSetCurveData(p, qwtPlot1, 1, 100, d->x2, d->cos_x);
    qpwReplot(p,qwtPlot1);
  }

  // draw graphic with gDraw routines
  if(d->tab == 1) drawGraphic1(p, qDraw1, d);

  // animate some data
  d->phi1 += d->step1/10.0f;
  d->phi2 += d->step2/10.0f;
  if(d->phi1 > (1000.0f*2.0f*PI)) d->phi1 = 0.0f;
  if(d->phi2 > (1000.0f*2.0f*PI)) d->phi2 = 0.0f;
  for(int i=0; i<100; i++)
  {
    d->x1[i]=(((float) i) * 2.0f * PI) / 100.0f;
    d->sin_x[i]=sinf((float) d->x1[i]+d->phi1);
    d->x2[i]=(((float) i) * 2.0f * PI) / 100.0f;
    d->cos_x[i]=cosf((float) d->x2[i]+d->phi2);
  }
  return 0;
}
```

In der Funktion 'drawGraphic1' wird ein Diagramm mit dem Draw Widget ausgegeben. Die Funktionen mit 'qpw' am Anfang, sind Aufrufe des QwtPlot Widget.

Das *Draw Widget* bietet einige 'g-Funktionen' (beginnen mit dem Buchstaben 'g'), mit denen man beliebige Grafiken zeichnen kann. Darunter gibt es auch Funktionen, die Achsen-kreuze und Kurven darin zeichnen.

Das *QwtPlot Widget* kann ausgefeiltere Diagramme zeichnen. Sogar logarithmisch skalierte Diagramme sind damit möglich.

Sehen Sie sich dazu bitte die Unterpunkte *Graphics* und *QwtPlotWidget* in der Referenz der pvslib an.

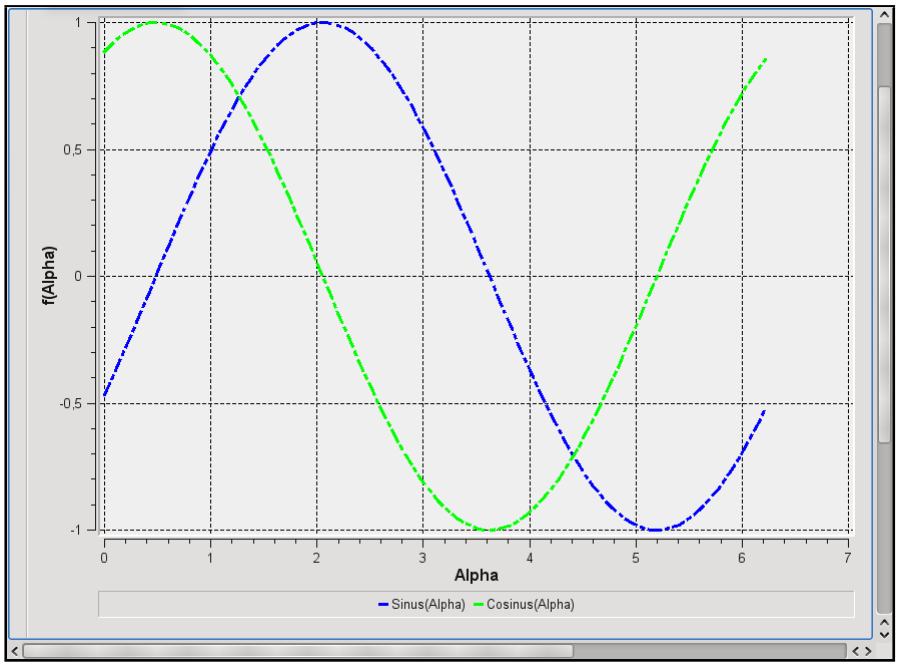

Abbildung 6.37: xy Grafik mit dem QwtPlot Widget

6.8.3 externe Plot Werkzeuge

Externe Plot Werkzeuge können aus dem pvserver aufgerufen werden. Wenn das Plot Werkzeug in der Lage ist, Bitmap Grafiken als Ausgabedatei zu erstellen, kann man die Grafik einfach in pvbrowser einbauen. Wir zeigen das hier am Beispiel von Gnuplot.

Gnuplot als Beispiel eines externen Plot Werkzeuges

```
static int showGnuplot1(PARAM *p, int id, DATA *d)
{
  FILE *fout;
  char buf[80];
  if(d == NULL) return 1; // you may use d for writing gnuplot.dem

#ifdef _WIN32
  if(1)
  {
    pvPrintf(p,label1,"Gnuplot1 if installed");
    return 0;
  }
#endif

  // write gnuplot file
  sprintf(buf,"%sgnuplot.dem",p->file_prefix); // p->file_prefix makes filenames unique for multiple
      clients
  fout = fopen(buf,"w");
  if(fout == NULL) return 1;
  fprintf(fout,"# gnuplot test.dem\n");
  fprintf(fout,"set output \"%sgnuplot.png\"\n",p->file_prefix);
  fprintf(fout,"set terminal png\n");
  fprintf(fout,"set xlabel \"x\"\n");
  fprintf(fout,"set ylabel \"y\"\n");
  fprintf(fout,"set key top\n");
  fprintf(fout,"set border 4095\n");
  fprintf(fout,"set xrange [-15:15]\n");
  fprintf(fout,"set yrange [-15:15]\n");
  fprintf(fout,"set zrange [-0.25:1]\n");
```

```
  fprintf(fout,"set␣samples␣25\n");
  fprintf(fout,"set␣isosamples␣20\n");
  fprintf(fout,"set␣title␣\"Radial␣sinc␣function.\"\n");
  fprintf(fout,"splot␣sin(sqrt(x**2+y**2))/sqrt(x**2+y**2)\n");
  fclose(fout);

  // run gnuplot
  sprintf(buf,"gnuplot␣%sgnuplot.dem",p->file_prefix);
  rlsystem(buf);

  // send result to pvbrowser
  sprintf(buf,"%sgnuplot.png",p->file_prefix);
  pvDownloadFile(p,buf);
  pvSetImage(p,id,buf);

  // temopary files will be cleaned up at browser exit
  return 0;
}

static int showGnuplot2(PARAM *p, int id, DATA *d)
{
  FILE *fout;
  char buf[80];
  if(d == NULL) return 1; // you may use d for writing gnuplot.dem

#ifdef _WIN32
  if(1)
  {
    pvPrintf(p,label1,"Gnuplot2␣if␣installed");
    return 0;
  }
#endif

  // write gnuplot file
  sprintf(buf,"%sgnuplot.dem",p->file_prefix); // p->file_prefix makes filenames unique for multiple
      clients
  fout = fopen(buf,"w");
  if(fout == NULL) return 1;
  fprintf(fout,"#␣gnuplot␣test.dem\n");
  fprintf(fout,"set␣output␣\"%sgnuplot.png\"\n",p->file_prefix);
  fprintf(fout,"set␣terminal␣png\n");
  fprintf(fout,"#\n");
  fprintf(fout,"#␣$Id:␣pm3dcolors.dem,v␣1.2␣2003/10/17␣15:02:21␣mikulik␣Exp␣$\n");
  fprintf(fout,"#\n");
  fprintf(fout,"#␣Test␣of␣new␣color␣modes␣for␣pm3d␣palettes.\n");
  fprintf(fout,"\n");
  fprintf(fout,"#\n");
  fprintf(fout,"#␣␣␣Gradient␣Palettes\n");
  fprintf(fout,"#\n");
  fprintf(fout,"set␣pm3d;␣set␣palette\n");
  fprintf(fout,"set␣palette␣color\n");
  fprintf(fout,"set␣pm3d␣map\n");
  fprintf(fout,"set␣cbrange␣[-10:10]\n");
  fprintf(fout,"set␣xrange␣[-10:10]\n");
  fprintf(fout,"set␣yrange␣[*:*]\n");
  fprintf(fout,"unset␣ztics\n");
  fprintf(fout,"unset␣ytics\n");
  fprintf(fout,"set␣samples␣101\n");
  fprintf(fout,"set␣isosamples␣2\n");
  fprintf(fout,"set␣xtics␣2\n");
  fprintf(fout,"\n");
  fprintf(fout,"set␣palette␣model␣RGB\n");
  fprintf(fout,"\n");
```

```
  fprintf(fout,"set␣palette␣defined\n");
  fprintf(fout,"set␣title␣\"set␣palette␣defined\"\n");
  fprintf(fout,"splot␣x\n");
  fclose(fout);

  // run gnuplot
  sprintf(buf,"gnuplot␣%sgnuplot.dem",p->file_prefix);
  rlsystem(buf);

  // send result to pvbrowser
  sprintf(buf,"%sgnuplot.png",p->file_prefix);
  pvDownloadFile(p,buf);
  pvSetImage(p,id,buf);

  // temopary files will be cleaned up at browser exit
  return 0;
}

static int slotInit(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  //memset(d,0,sizeof(DATA));
  pvPrintf(p,ID_TAB,"Gnuplot");
  pvResize(p,0,1280,1024);
  pvSetPixmap(p,back,"back.png");
  showGnuplot1(p,imageGnuplot1,d);
  return 0;
}

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

static int slotButtonEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  if(id == back) return WELLCOME;
  if(id == buttonPlot1) showGnuplot1(p,imageGnuplot1,d);
  if(id == buttonPlot2) showGnuplot2(p,imageGnuplot1,d);
  return 0;
}
```

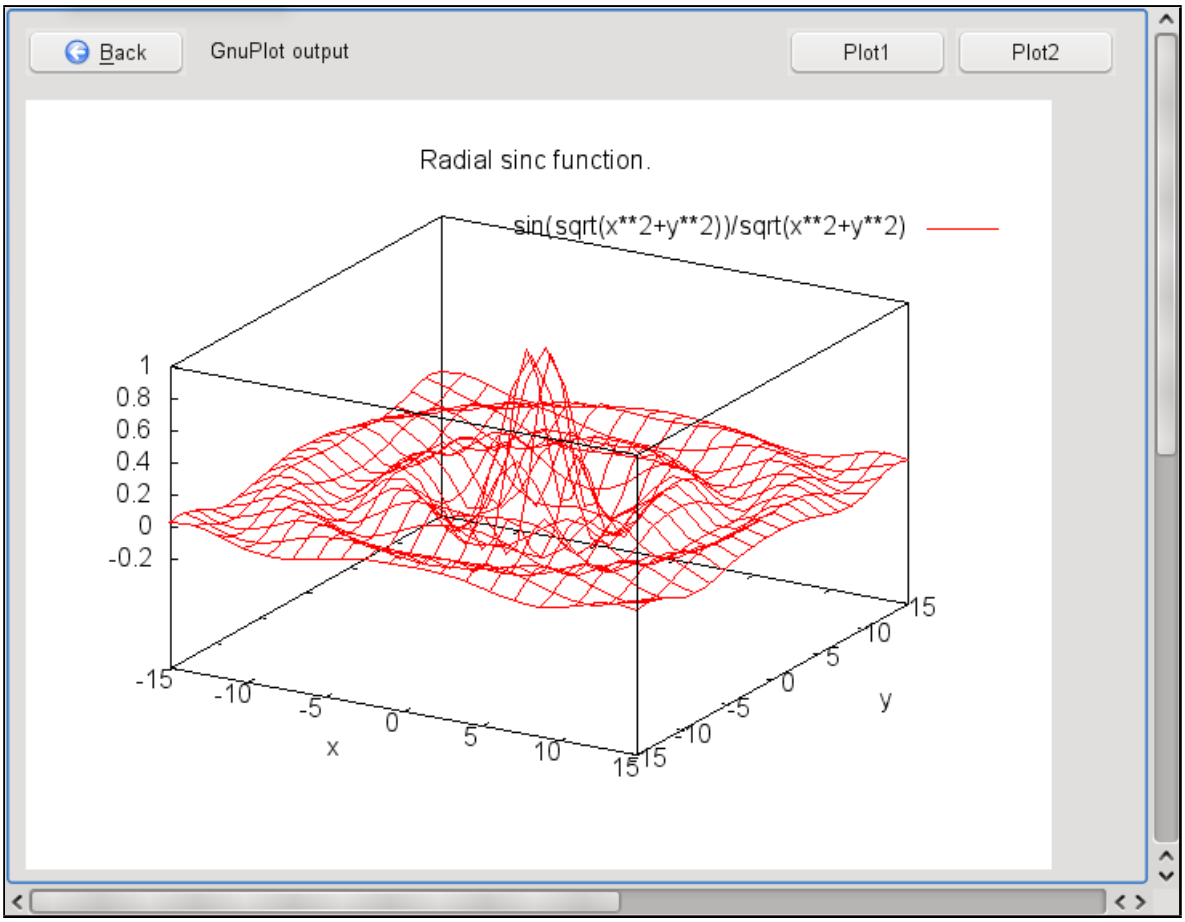

Abbildung 6.38: Gnuplot Ausgabe in pvbrowser

6.8.4 SVG Grafiken

Die Scalable Vektor Graphik ist eine XML Spezifikation und ein Dateiformat, um zwei-dimensionale Vektor Grafiken zu beschreiben, sowohl statisch als auch animiert. Es ist ein offener Standard, der vom der W3C SVGWorking Group erstellt wurde.
SVG Grafiken können mit darauf spezialisierten Zeichenprogrammen, wie Inkscape erstellt werden. Alternativ kann man auch Konverter verwenden, die CAD Formate wie DXF in SVG umwandeln.

SVG Beispiel Code

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:ev="http://www.w3.org/2001/xml-events"
    version="1.1" baseProfile="full"
    width="107" height="60" viewBox="-2␣-5␣105␣55">
  <line x1="0" y1="25" x2="100" y2="25" fill="none" stroke="black"
      stroke-width="3px"/>
  <rect x="10" y="15" width="80" height="20" fill="white" stroke="black"
      stroke-width="3px" />
  <polyline points="65␣5␣␣40␣40␣␣40␣50" fill="none" stroke="black"
      stroke-width="3px"/>
  <polygon points="60␣5␣␣70␣5␣␣65␣-5" stroke="black" stroke-width="3px"
      transform="rotate(33.7␣65␣5)" />
</svg>
```

Diese Grafiken können nun in pvbrowser verwendet werden. Fügen Sie einfach ein Draw/SVG Widget ein und benutzen rlSvgAnimator aus der rllib, um die Grafik zu zeichnen und zu animieren. Unter Verwendung

dieser Klasse wird die SVG Grafik vom Server zum pvbrowser Client gesendet. Wenn die Grafik erst einmal im pvbrowser Client ist, wird rlSvgAnimator verwendet, um die Grafik zu verändern (animieren). Der Server empfängt Ereignisse, wenn Sie Objekte der SVG Grafik anklicken. Darüber hinaus werden einige Maus Ereignisse gesendet.

Definition von rlSvgAnimator

public Teil von rlSvgAnimator

```
class rlSvgAnimator
{
  public:
    rlSvgAnimator();
    virtual ~rlSvgAnimator();

    /*! initialize the socket with pvbrowser p->s */
    int setSocket(int *socket);
    /*! initialize the id with the pvbrowser object id */
    int setId(int Id);
    /*! read SVG file infile and load it into the pvbrowser client.
        if inifile is given inifile will be set with properties within the SVG XML file.
        The id's of the SVG objects will result in section names of the inifile. */
    int read(const char *infile, rlIniFile *inifile=NULL);
    /*! update the SVG graphic with:
        gBeginDraw(p,id); d->svgAnimator.writeSocket(); gEndDraw(p); */
    //! The following methods are for modifying a object within a SVG graphic identified by
        objectname
    int writeSocket();
    /*! change a property of tag = "name=" */

    int svgPrintf(const char *objectname, const char *tag, const char *format, ...);
    /*! recursively change a property of tag = "name=" */
    int svgRecursivePrintf(const char *objectname, const char *tag, const char *format, ...);
    /*! search for "before" within "tag=" property and replace it with "after". You may use wildcards
         whin "before" */
    int svgSearchAndReplace(const char *objectname, const char *tag, const char *before, const char *
        after);
    /*! recursively search for "before" within "tag=" property and replace it with "after". You may
        use wildcards within "before" */
    int svgRecursiveSearchAndReplace(const char *objectname, const char *tag, const char *before,
        const char *after);
    /*! change the text of "objectname" */
    int svgTextPrintf(const char *objectname, const char *format, ...);
    /*! remove a style option of "objectname". option must end with ':'. Example: option="stroke:" */
    int svgRemoveStyleOption(const char *objectname, const char *option);
    /*! recursively remove a style option of "objectname". option must end with ':'. Example: option
        ="stroke:" */
    int svgRecursiveRemoveStyleOption(const char *objectname, const char *option);
    /*! change a style option of "objectname". option must end with ':'. Example: option="stroke:"
        value="#000000" */
    int svgChangeStyleOption(const char *objectname, const char *option, const char *value);
    /*! recursively change a style option of "objectname". option must end with ':'. Example: option
        ="stroke:" value="#000000" */
    int svgRecursiveChangeStyleOption(const char *objectname, const char *option, const char *value);
    /*! set a style option of "objectname". option must end with ':'. Example: value="fill:#9d9d9d;
        fill-opacity:1;fill-rule:evenodd;stroke:#000000;stroke-width:3.5;stroke-linecap:butt;stroke-
        linejoin:miter;stroke-dasharray:none;stroke-opacity:1" */
    int svgSetStyleOption(const char *objectname, const char *value);
    /*! recursively set a style option of "objectname". option must end with ':'. Example: value="
        fill:#9d9d9d;fill-opacity:1;fill-rule:evenodd;stroke:#000000;stroke-width:3.5;stroke-linecap:
        butt;stroke-linejoin:miter;stroke-dasharray:none;stroke-opacity:1" */
    int svgRecursiveSetStyleOption(const char *objectname, const char *value);
    /*! hide/show object state := 0=hide 1=show */
```

```
    int show(const char *objectname, int state); // state := 0|1
    /*! set transformation matrix of object */
    int setMatrix(const char *objectname, float sx, float alpha, float x0, float y0, float cx, float
        cy);
    /*! set transformation matrix of object */
    //! The following methods are for moveing and zooming the whole SVG identified by mainObject.
        default: main
    int setMatrix(const char *objectname, rlSvgPosition &pos);
    /*! set/get the name of the MainObject . The object name holding the whole SVG graphic. default:
        main */

    int setMainObject(const char *main_object);
    const char *mainObject();
    /*! set/get x0,y0 coordinates for the MainObject */
    int setXY0(float x0, float y0);
    float x0();
    float y0();
    /*! set/get mouse position 0 for the MainObject */
    int setMouseXY0(float x0, float y0);
    float mouseX0();
    float mouseY0();
    /*! set/get mouse position 1 for the MainObject */
    int setMouseXY1(float x1, float y1);
    float mouseX1();
    float mouseY1();
    /*! set/get the scaling factor for the MainObject */
    int setScale(float scale);
    float scale();
    /*! zooms the whole SVG graphic keeping it centered to the viewport */
    int zoomCenter(float newScale);
    /*! zooms the whole SVG graphic so that the visible section is from x0,x0 to x1,y1 */
    int zoomRect();
    /*! sets the MainObject matrix according to scale,x0,y0 */
    int setMainObjectMatrix();
    /*! call this method when the widget is resized */
    int setWindowSize(int width, int height);
    float windowWidth();
    float windowHeight();
    /*! move MainObject to position */
    int moveMainObject(float x, float y);

    int isModified;

  private:
```

Laden und Zeichnen einer SVG Grafik

Der folgende Code zeigt, wie eine SVG Grafik in den pvbrowser Client geladen und dann gezeichnet wird. Zunächst wird eine Instanz der Klasse rlSvgAnimator in DATA definiert. In 'slotInit' wird der Socket zur Kommunikation mit dem pvbrowser Client und die Objekt id in svgAnimator registriert. Mit der Methode 'read' wird dann die SVG Datei an den pvbrowser Client gesendet. Mit dem Aufrufen von 'pvSetZoomX' und 'pvSetZoomY' wird pvbrowser mitgeteilt, dass die Seitenverhältnisse in der Grafik bei Größenänderungen erhalten bleiben sollen. Mit der kleinen Hilfsfunktion 'drawSVG1' kann die SVG Grafik dann gezeichnet bzw. nach durchgeführten Animationsschritten aktualisiert werden.

SVG Zeichnung laden und zeichnen

```
typedef struct // (todo: define your data structure here)
{
  rlSvgAnimator svgAnimator;
}
DATA;
```

```
static int drawSVG1(PARAM *p, int id, DATA *d)
{
  if(d == NULL) return -1;
  if(d->svgAnimator.isModified == 0) return 0;
  printf("writeSocket\n");
  gBeginDraw(p,id);
  d->svgAnimator.writeSocket();
  gEndDraw(p);
  return 0;
}

static int slotInit(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  //memset(d,0,sizeof(DATA));

  // load HTML
  pvDownloadFile(p,"icon32x32.png");
  pvDownloadFile(p,"upperWidget.html");
  pvDownloadFile(p,"leftWidget.html");
  pvSetSource(p,upperWidget,"upperWidget.html");
  pvSetSource(p,leftWidget,"leftWidget.html");

  // load SVG
  d->svgAnimator.setSocket(&p->s);
  d->svgAnimator.setId(centerWidget);
  d->svgAnimator.read("test.svg");

  // keep aspect ratio of SVG
  pvSetZoomX(p, centerWidget, -1.0f);
  pvSetZoomY(p, centerWidget, -1.0f);

  // draw SVG
  drawSVG1(p,centerWidget,d);

  // download icons
  pvDownloadFile(p,"1center.png");
  pvDownloadFile(p,"1uparrow.png");
  pvDownloadFile(p,"1downarrow.png");
  pvDownloadFile(p,"1leftarrow.png");
  pvDownloadFile(p,"1rightarrow.png");
  pvDownloadFile(p,"1center2.png");
  pvDownloadFile(p,"1uparrow2.png");
  pvDownloadFile(p,"1downarrow2.png");
  pvDownloadFile(p,"1leftarrow2.png");
  pvDownloadFile(p,"1rightarrow2.png");

  // set sliderZoom to 100 percent
  pvSetValue(p,sliderZoom,100);

  //pvSetSource(p,upperWidget,"de_total.html");
  return 0;
}
```

Animieren einer SVG Grafik

Wenn Sie die SVG Grafik erst einmal geladen haben und in der Lage sind, sie neu zu zeichnen, können Sie die Grafik animieren (verändern), indem Sie rlSvgAnimator verwenden.
Der Objekt-name wird benutzt, um die graphischen Objekte in SVG zu adressieren (id='objectname'). Mit dem SVG Zeichenprogramm, kann man die id wie gewünscht setzen.

Den Text in einem SVG Objekt verändern

```
d->svgAnimator.svgTextPrintf("HelloObject", "Hello␣World");
```

Objekt verstecken und wieder anzeigen

```
d->svgAnimator.show("HelloObject", 0); // hide HelloObject
d->svgAnimator.show("HelloObject", 1); // show HelloObject
```

Objekt Eigenschaften setzen

```
d->svgAnimator.svgPrintf("HelloObject", "fill=", "#000");
```

Jedes Objekt kann über eine Transformationsmatrix skaliert, rotiert und verschoben werden. Dazu werden in SVG Transformationsmatrizen verwendet. Mit Hilfe von rlSvgAnimator kann man die Matrizen für jedes Objekt ändern.

Transformationsmatrix für ein Objekt ändern

```
// sx    := scale factor
// alpha := rotation angle in radiant
// x0,y0 := position
// cx,cy := position around which to rotate
d->svgAnimator.setMatrix(const char *objectname, float sx, float alpha, float x0, float y0, float cx,
     float cy);
// or using
d->svgAnimator.setMatrix(const char *objectname, rlSvgPosition &pos);
```

Achtung:
Bei dem ersten Aufruf von setMatrix() wird das Objekt 'objectname' mit einer Gruppierung versehen. Das führt dazu, dass alle versteckten Kind Objekte von 'objectname' wieder sichtbar werden.

Definition von rlSvgPosition

```
class rlSvgPosition
{
  public:
    rlSvgPosition();
    rlSvgPosition(float sx_init, float a_init, float x0_init, float y0_init, float cx_init, float
        cy_init);
    virtual ~rlSvgPosition();
    float sx, alpha, x0, y0, cx, cy;
    struct rlPositionInit {float sx, alpha, x0, y0, w, h;} init;
    void setInit(float x0_init, float y0_init, float w_init, float h_init);
    void move(float x, float y);
    void moveRelative(float dx, float dy);
    void scale(float s);
    void scaleRelative(float ds);
    void rotate(float alpha, float cx, float cy);
};
```

Nach dem Anlegen einer Instanz von rlSvgPosition ist die Matrix auf die Werte der Identitätsmatrix gesetzt (keine Verschiebung, keine Rotation, Skalierungsfaktor 1.0). Indem die rlSvgPosition verändert wird, können die Objekte skaliert, rotiert und verschoben werden. Nachdem Sie eine rlSvgPosition verändert haben, muss 'setMatrix' aus rlSvgAnimator aufgerufen werden, um die Änderung auf das Objekt anzuwenden.
Mit der Methode 'svgSearchAndReplace' aus rlSvgAnimator kann man Bestandteile von Eigenschaften ersetzen. Dies ist z.B. nützlich, wenn man einzelne Elemente einer Style Eigenschaft ersetzen möchte. Der 'before' Parameter kann auch Wildcards wie ∗(beliebige Zeichen) ? (ein beliebiges Zeichen) enthalten.

Eigenschaften modifizieren

```
int svgSearchAndReplace(const char *objectname, const char *tag, const char *before, const char *
    after);
```

Es stehen außerdem Methoden zur Verfügung, mit denen man rekursiv auf alle Elemente unterhalb eines graphischen Objektes arbeiten kann.

Objekteigenschaften rekursiv setzen bzw. modifizieren

```
int svgRecursivePrintf(const char *objectname, const char *tag, const char *format,...);
int svgRecursiveSearchAndReplace(const char *objectname, const char *tag, const char *before, const
    char *after);
```

Nachdem alle Objekte in der SVG Grafik wie gewünscht modifiziert worden sind, müssen sie mit 'drawSVG1' neu gezeichnet werden.
Die Methode 'read',die die SVG Grafik an den Browser sendet, kann man noch einen optionalen Parameter ini file anhängen. Dieser wird dann mit den Eigenschaften in der SVG Grafik gefüllt. Dabei entsprechen die id's der graphischen Objekte den Sektionen einer Ini Datei. Tipp: Sie können eigene Eigenschaften der Form your:tag='any value' in der SVG eingeben, die nicht vom SVG Renderer interpretiert werden. Diese Eigenschaften können von Ihnen ganz nach Belieben verwendet werden.

Optionale INI Datei bei read

```
int read(const char *infile, rlIniFile *inifile=NULL);
```

Ereignisverarbeitung von SVG Grafiken

SVG Grafiken werden mit Hilfe des Draw Widget implementiert. Das Draw Widget kann folgende Ereignisse liefern, wenn es eine SVG Grafik enthält.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotRightMouseEvent(PARAM *p, int id, DATA *d, const char *text)*
- *static int slotMouseMovedEvent(PARAM *p, int id, DATA *d, float x, float y)*
- *static int slotMousePressedEvent(PARAM *p, int id, DATA *d, float x, float y)*
- *static int slotMouseReleasedEvent(PARAM *p, int id, DATA *d, float x, float y)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*

Besonders interessant sollte das TextEvent sein.
Falls Sie ein Ereignis haben wollen, wenn ein graphisches Objekt in der SVG Grafik angeklickt wird, muss die id des Objektes mit 'pv.' oder 'PV.' beginnen.
Beispiel: id='pv.testclick'
Dann wird 'slotTextEvent' aufgerufen. Wenn der Name mit 'PV.' beginnt id='PV.textclick' wird pvbrowser zusätzlich eine visuelle Rückkopplung geben, indem sich der Maus Cursor in ein Qt::PointingHandCursor verwandelt, wenn der Benutzer die Maus über das Objekt führt. Damit lassen sich beispielsweise 'Buttons' mit visueller Rückkopplung erzeugen.
slotTextEvent wird nicht nur für dieses Klick Ereignis benutzt, sondern es gibt mehrere Funktionen, die slotTextEvent auslösen. Dafür gibt es ein paar Hilfsfunktionen, die den Text entsprechend interpretieren.

slotTextEvent bei einer SVG Grafik

```
static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)
{
  if(p == NULL || id == 0 || d == NULL || text == NULL) return -1;
  float x,y,w,h;
  float m11,m12,m21,m22,det,dx,dy;
  switch(textEventType(text))
  {
    case PLAIN_TEXT_EVENT:
      printf("plain\n");
      break;
    case WIDGET_GEOMETRY:
      int X,Y,W,H;
      getGeometry(text,&X,&Y,&W,&H);
```

```
      printf("geometry(%d)=%d,%d,%d,%d\n",id,X,Y,W,H);
      break;
    case PARENT_WIDGET_ID:
      int PID;
      getParentWidgetId(text,&PID);
      printf("parent(%d)=%d\n",id,PID);
      break;
    case SVG_LEFT_BUTTON_PRESSED:
      printf("left␣pressed\n");
      printf("objectname=%s\n",svgObjectName(text));
      break;
    case SVG_MIDDLE_BUTTON_PRESSED:
      printf("middle␣pressed\n");
      printf("objectname=%s\n",svgObjectName(text));
      break;
    case SVG_RIGHT_BUTTON_PRESSED:
      printf("right␣pressed\n");
      printf("objectname=%s\n",svgObjectName(text));
      break;
    case SVG_LEFT_BUTTON_RELEASED:
      printf("left␣released\n");
      printf("objectname=%s\n",svgObjectName(text));
      break;
    case SVG_MIDDLE_BUTTON_RELEASED:
      printf("middle␣released\n");
      printf("objectname=%s\n",svgObjectName(text));
      break;
    case SVG_RIGHT_BUTTON_RELEASED:
      printf("right␣released\n");
      break;
    case SVG_BOUNDS_ON_ELEMENT:
      getSvgBoundsOnElement(text, &x, &y, &w, &h);
      printf("bounds␣object=%s␣xywh=%f,%f,%f,%f\n",svgObjectName(text),x,y,w,h);
      break;
    case SVG_MATRIX_FOR_ELEMENT:
      getSvgMatrixForElement(text, &m11, &m12, &m21, &m22, &det, &dx, &dy);
      printf("matrix␣object=%s␣m=%f,%f,%f,%f␣det=%f␣dx=%f␣dy=%f\n",svgObjectName(text),
                              m11,m12,m21,m22,det,dx,dy);
      break;
    default:
      printf("default\n");
      break;
  }
  return 0;
}
```

Über die Hilfsfunktion 'svgObjectName' erhalten Sie den Namen des SVG Objektes. Mit der Hilfsfunktion 'textEventType' wird entschieden, ob es sich um ein normales TextEvent oder um eines der speziell für SVG vorgesehenen Ereignisse handelt. Sie erfahren dann, ob Ihr Objekt mit der Maus angeklickt wurde und mit welcher Maustaste das passiert ist.

Der 'case SVG_BOUNDS_ON_ELEMENT:' ist dabei eine Antwort auf folgende Anfrage.

Einschließendes Rechteck für ein SVG Objekt anfragen

```
pvRequestSvgBoundsOnElement(p, svgExample, "testobject");
```

Der 'case SVG_MATRIX_FOR_ELEMENT:' ist dabei eine Antwort auf folgende Anfrage.

Matrix für ein SVG Objekt anfragen

```
pvRequestSvgMatrixForElement(p, svgExample, "testobject");
```

Zoomen und Verschieben der gesamten SVG Grafik

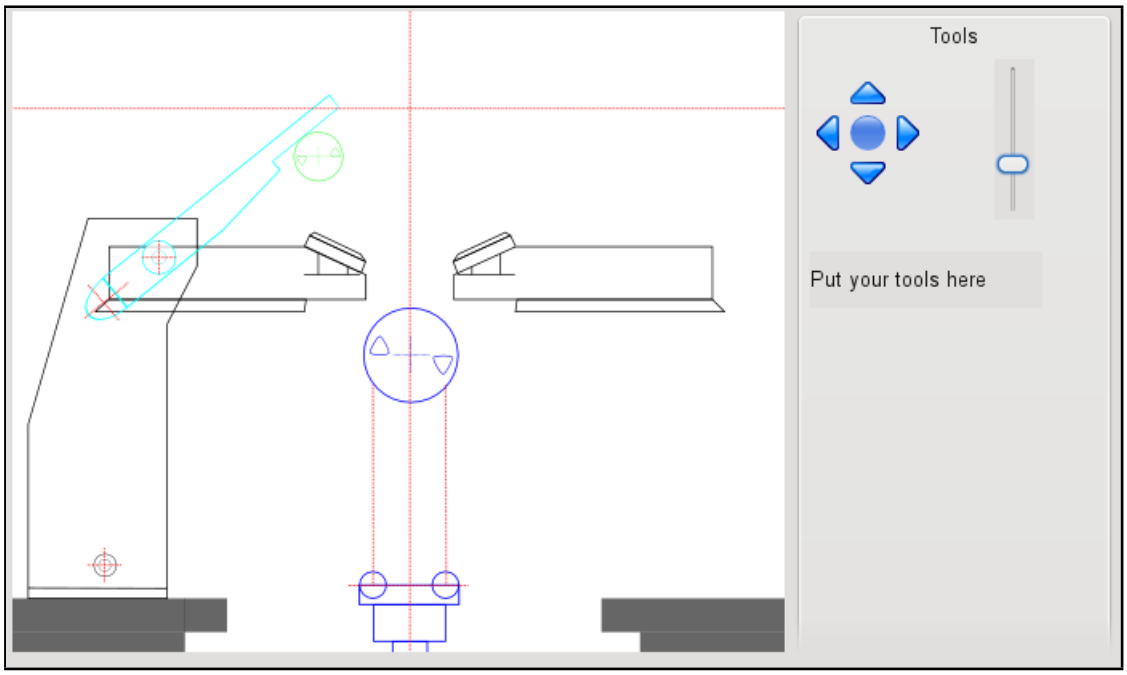

Abbildung 6.39: Zoomen und Verschieben der gesamten SVG Grafik

In pvbaddon Verzeichnis pvbaddon/template/weblayout/ finden Sie eine Vorlage, wo das Zoomen und Verschieben der gesamten SVG Grafik implementiert ist.
Sehen Sie zunächst nach, wie der Name (id) der gesamten SVG Grafik im SVG XML Code heißt. Wir schlagen vor, dieses Objekt 'main' zu nennen, da das der default Name ist, den rlSvgAnimator dafür annimmt.
Darin sind folgende slot Funktionen für Zoomen und Verschieben verantwortlich.

Zoomen der gesamten SVG Grafik in Prozent

```
static int slotSliderEvent(PARAM *p, int id, DATA *d, int val)
{
  if(p == NULL || id == 0 || d == NULL || val < -1000) return -1;
  if(id == sliderZoom)
  {
    float zoom = ((float) val) / 100.0f;
    d->svgAnimator.setScale(zoom);
    d->svgAnimator.setMainObjectMatrix();
    drawSVG1(p,centerWidget,d);
  }
  return 0;
}
```

Anpassung an aktuelle Fenstergröße

```
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;
  d->svgAnimator.setWindowSize(width,height);
  drawSVG1(p,centerWidget,d);
  return 0;
}
```

Grafik mit Hilfe der Maus verschieben

```
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;
  if(id == centerWidget) // the SVG
  {
    // drag the SVG with your mouse
    //float x0 = d->svgAnimator.x0() + (((x*d->svgAnimator.windowWidth()) / 100.0f) - d->svgAnimator.
        mouseX0());
    //float y0 = d->svgAnimator.y0() + (((y*d->svgAnimator.windowHeight()) / 100.0f) - d->svgAnimator
        .mouseY0());
    //d->svgAnimator.setXY0(x0,y0);
    //d->svgAnimator.setMouseXY0(x,y);
    //d->svgAnimator.setMainObjectMatrix();
    //drawSVG1(p,centerWidget,d);
    d->svgAnimator.moveMainObject(x,y);
    drawSVG1(p,centerWidget,d);
    d->svgAnimator.setMouseXY0(x,y);
  }
  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;
  if(id == centerWidget) // the SVG
  {
    // remember initial position for dragging
    d->svgAnimator.setMouseXY0(x,y);
  }
  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;
  if(id == centerWidget) // the SVG
  {
    // drag the SVG with your mouse
    d->svgAnimator.moveMainObject(x,y);
    drawSVG1(p,centerWidget,d);
  }
  return 0;
}
```

Verschiebung der SVG Grafik mit Buttons

```
static int slotButtonEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  if    (id == iCenter)
  {
    pvSetImage(p,iCenter,"1center2.png");
    d->svgAnimator.zoomCenter(1.0f);
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.setXY0(0.0f,0.0f);
    d->svgAnimator.moveMainObject(0,0);
    drawSVG1(p,centerWidget,d);
    pvSetValue(p,sliderZoom,100);
  }
  else if(id == iUp)
  {
    pvSetImage(p,iUp,"1uparrow2.png");
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.moveMainObject(0,-DELTA);
    drawSVG1(p,centerWidget,d);
```

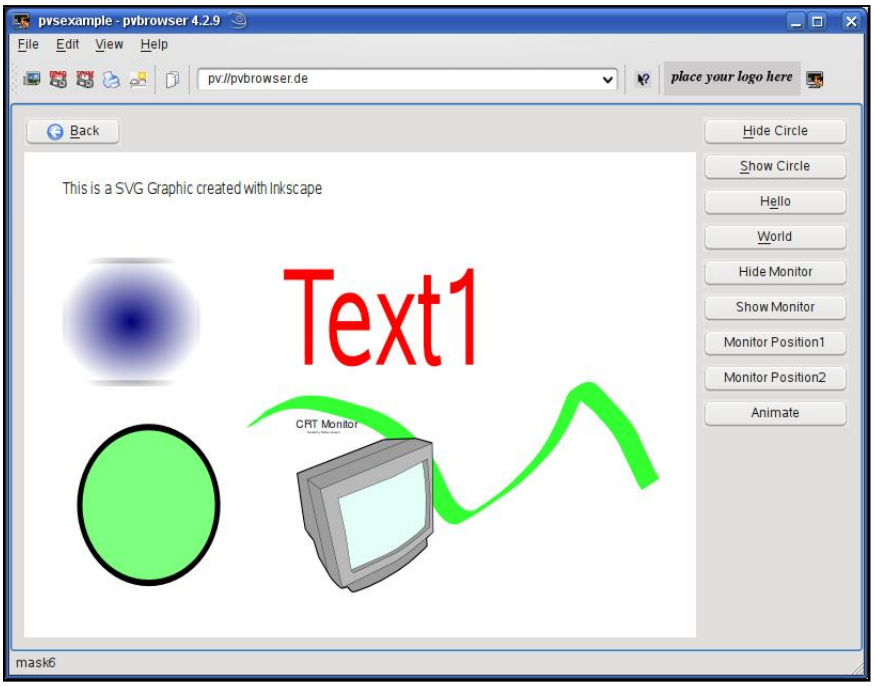

Abbildung 6.40: Eine Einfache SVG

```
  }
  else if(id == iDown)
  {
    pvSetImage(p,iDown,"1downarrow2.png");
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.moveMainObject(0,DELTA);
    drawSVG1(p,centerWidget,d);
  }
  else if(id == iLeft)
  {
    pvSetImage(p,iLeft,"1leftarrow2.png");
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.moveMainObject(-DELTA,0);
    drawSVG1(p,centerWidget,d);
  }
  else if(id == iRight)
  {
    pvSetImage(p,iRight,"1rightarrow2.png");
    d->svgAnimator.setMouseXY0(0,0);
    d->svgAnimator.moveMainObject(DELTA,0);
    drawSVG1(p,centerWidget,d);
  }
  else if(id == pbPrintHtml)
  {
    pvPrintHtmlOnPrinter(p,upperWidget);
  }
  return 0;
}
```

Beispiele für SVG Grafiken

Die ersten beiden Beispiele stammen aus pvsexample, dass mit pvbrowser mitgeliefert wird.

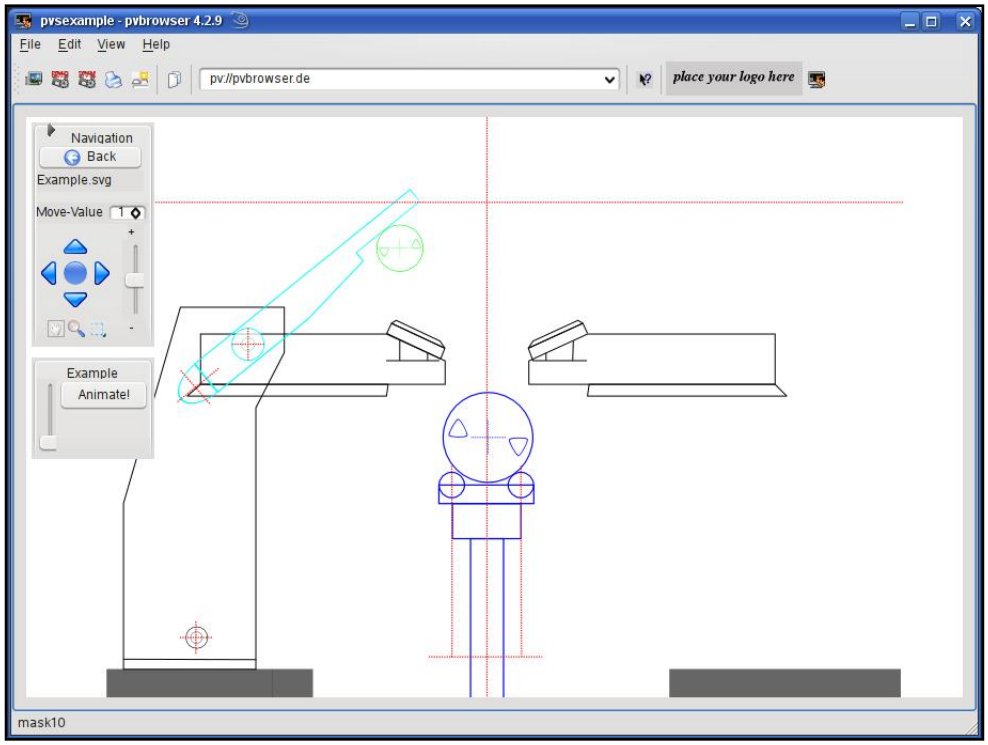

Abbildung 6.41: Eine mechanische Zeichnung

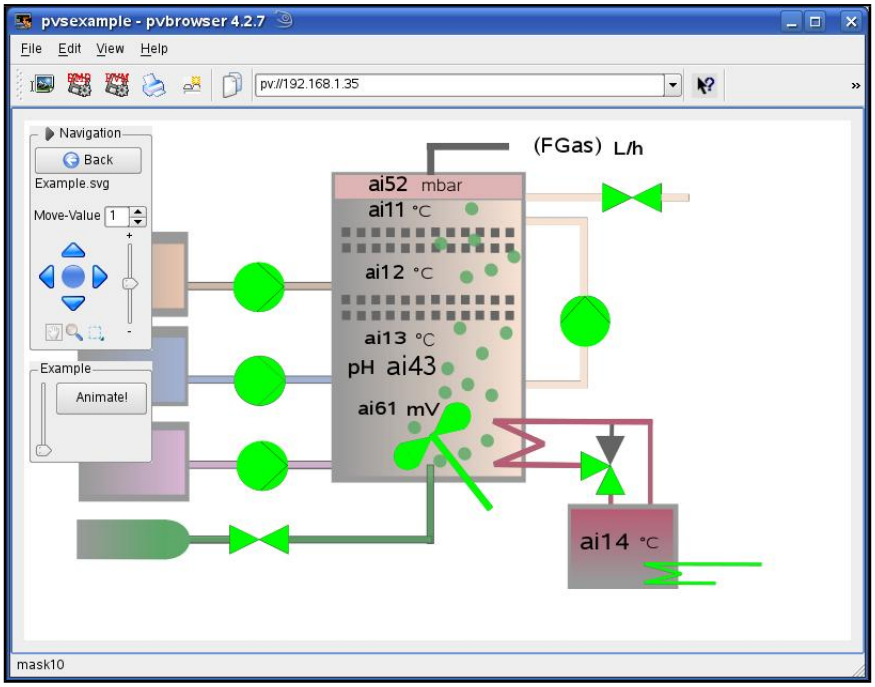

Abbildung 6.42: Eine SVG eines Prozesses

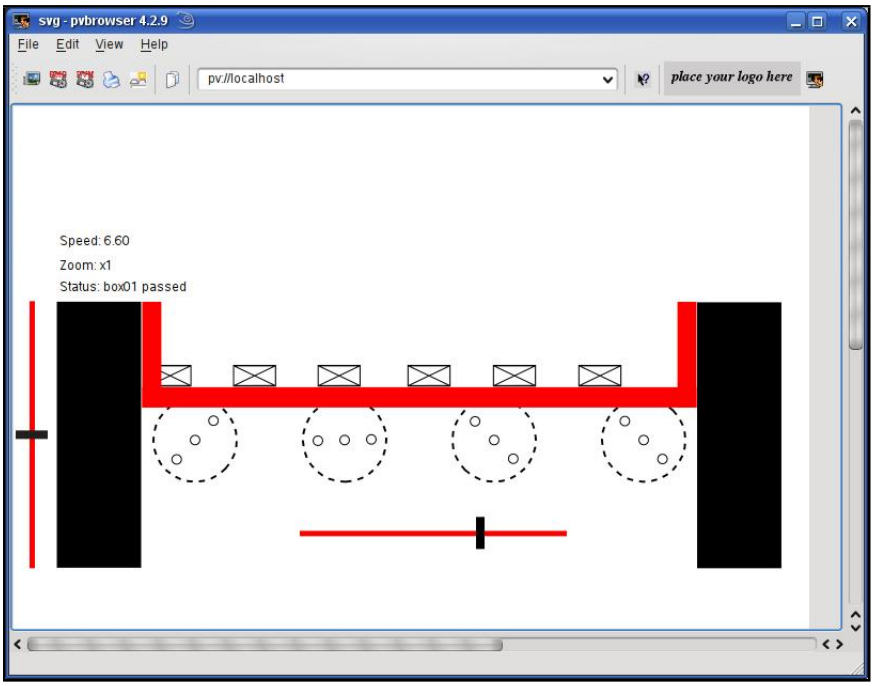

Abbildung 6.43: Eine SVG mit sich bewegenden Pakten auf einem Förderband

6.8.5 OpenGL

OpenGL Widgets können in pvbrowser verwendet werden. Der pvserver sendet die OpenGL Befehle zum pvbrowser Client, wo sie umgesetzt werden. Eine Anwendung von OpenGL kann die Einbindung von CAD Zeichnungen in pvbrowser sein. Autocad verwendet z.B. das DWF Dateiformat, um CAD Zeichnungen an andere Anwendungen zu übergeben. Für dieses Dateiformat gibt es bei uns einen Konverter, der DWF nach OpenGL umsetzt und dann für pvbrowser nutzbar macht. Die generierte Zeichnung wird mit 'pvSendOpenGL' an den pvbrowser Client gesendet und dort dargestellt. Die in der Zeichnung enthaltenen Objekte liegen als Display Liste vor (listarray). Mit Hilfe dieser Display Listen können die Grafiken animiert werden.
OpenGL Widgets liefern Ereignisse in folgenden Slots.

- *static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)*
- *static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)*
- *static int slotMouseMovedEvent(PARAM *p, int id, DATA *d, float x, float y)*
- *static int slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)*
- *static int slotGlInitializeEvent(PARAM *p, int id, DATA *d)*
- *static int slotGlPaintEvent(PARAM *p, int id, DATA *d)*
- *static int slotGlResizeEvent(PARAM *p, int id, DATA *d, int width, int height)*
- *static int slotGlIdleEvent(PARAM *p, int id, DATA *d)*

Eine aus einer Autocad DWF Datei exportierte OpenGL Datei wird in pvbrowser angezeigt und animiert

```
//#################################################################
//# mask1_slots.h for ProcessViewServer created: Do Nov 20 07:46:31 2008
//# please fill out these slots
//# here you find all possible events
//# Yours: Lehrig Software Engineering
//#################################################################
// Autocad DWF2OpenGL
```

```
// 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;

// constants for OpenGL scene
static const float  scale0 = 3.0f;
static const GLfloat mat_specular[] = {1.0,1.0,1.0,1.0};
static const GLfloat mat_shininess[] = {50.0};
static const GLfloat light_position[] = {1.0,1.0,1.0,1.0};
static const GLfloat white_light[] = {1.0,1.0,1.0,1.0};

// OpenGL variables
typedef struct
{
  GLdouble frustSize;
  GLdouble frustNear;
  GLdouble frustFar;
  GLfloat scale;
  GLfloat xRot;
  GLfloat yRot;
  GLfloat zRot;
  GLuint  listarray[100];
  int     num_listarray;
  GLfloat pos;
  GLfloat posAll;
  GLfloat posVertAll;
  GLfloat posAllOld;
  GLfloat posVertAllOld;
  GLfloat X0, Y0;
  int     height, width;
  int     mouseFirstPressed;
  glFont  proportional, fixed;
}GL;

typedef struct // (todo: define your data structure here)
{
  GL gl;
}
DATA;

int initializeGL(PARAM *p)
{
  if(p == NULL) return -1;
  glClearColor(0.9,0.9,0.9,0.0);   // Let OpenGL clear color
  glEnable(GL_DEPTH_TEST);
  glClear(GL_COLOR_BUFFER_BIT);
  glShadeModel(GL_SMOOTH);
  glEnable(GL_DEPTH_TEST);
  glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
  glEnable(GL_TEXTURE_2D);
  return 0;
}

int resizeGL(PARAM *p, int width, int height)
{
  if(p == NULL) return -1;
  glViewport(0, 0, (GLint) width, (GLint) height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  return 0;
```

```
}

static int paintGL(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  pvGlBegin(p,OpenGL1);

  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glFrustum(-d->gl.frustSize, d->gl.frustSize, -d->gl.frustSize, d->gl.frustSize, d->gl.frustNear, d
      ->gl.frustFar);
  glTranslatef( 0.0, 0.0, -3.5f );
  double aspect = (double) d->gl.width / (double) d->gl.height;
  glScalef( d->gl.scale, d->gl.scale*aspect, d->gl.scale );
  glTranslatef( d->gl.posAll, d->gl.posVertAll, 0.0 );
  for(int i=1; i< d->gl.num_listarray; i++) glCallList(d->gl.listarray[i]);
  glTranslatef( d->gl.pos, 0.0, 0.0 );
  glCallList(d->gl.listarray[0]);
  pvGlEnd(p);
  pvGlUpdate(p,OpenGL1);
  return 0;
}

static int slotInit(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  printf("slotInit␣1\n");
  // init DATA d
  memset(d,0,sizeof(DATA));
  d->gl.frustSize = 0.5;
  d->gl.frustNear = scale0;
  d->gl.frustFar = 200.0;
  d->gl.scale = 1.0f;
  d->gl.xRot = 0.0f;
  d->gl.yRot = 0.0f;
  d->gl.zRot = 0.0f;
  d->gl.num_listarray = 0;
  d->gl.pos         = 0.0f;
  d->gl.posAll      = 0.0f;
  d->gl.posVertAll  = 0.0f;
  d->gl.posAllOld   = 0.0f;
  d->gl.posVertAllOld = 0.0f;
  d->gl.X0 = d->gl.Y0 = 0.0f;
  d->gl.height = d->gl.width = 1;
  d->gl.mouseFirstPressed = 0;

  // set sliders
  printf("slotInit␣2\n");
  pvSetValue(p,sliderPos,50);
  printf("slotInit␣2.1\n");

  // load OpenGL graphic
  printf("slotInit␣3\n");
  pvGlBegin(p,OpenGL1);
  printf("slotInit␣3.1\n");
  d->gl.proportional.read("gl/proportional.glfont"); // load proportional font
  printf("slotInit␣3.2\n");
  d->gl.fixed.read("gl/fixed.glfont");          // load fixed font
  printf("slotInit␣3.3\n");
  d->gl.proportional.setZoom(0.9f);
  printf("slotInit␣3.4\n");
  d->gl.fixed.setZoom(0.9f);
```

```
  printf("slotInit␣3.5\n");
  d->gl.num_listarray = pvSendOpenGL(p,"gl/scene.gl.cpp",d->gl.listarray,100,&d->gl.proportional,&d->
      gl.fixed);
  printf("slotInit␣3.6\n");
  pvGlEnd(p);

  printf("slotInit␣4\n");
  paintGL(p,d);

  // Download Graphics
  printf("slotInit␣5\n");
  pvDownloadFile(p,"1uparrow2.png");
  pvDownloadFile(p,"1uparrow3.png");
  pvDownloadFile(p,"1leftarrow2.png");
  pvDownloadFile(p,"1leftarrow3.png");
  pvDownloadFile(p,"1rightarrow2.png");
  pvDownloadFile(p,"1rightarrow3.png");
  pvDownloadFile(p,"1downarrow2.png");
  pvDownloadFile(p,"1downarrow3.png");
  pvDownloadFile(p,"1center2.png");
  pvDownloadFile(p,"1center3.png");

  // set images 4 buttons
  pvSetPixmap(p,btBack,"back.png");

  printf("slotInit␣end\n");
  return 0;
}

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

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

  if(id == btBack)
  {
    return WELLCOME; // call mask 1
  }
  if(id == btCenter)
  {
    d->gl.posAll = 0.0f;
    d->gl.posVertAll = 0.0f;
    paintGL(p,d);
    pvSetImage(p,id,"1center2.png");
  }
  if(id == btLeft)
  {
    d->gl.posAll -= 0.1f;
    paintGL(p,d);
    pvSetImage(p,id,"1leftarrow2.png");
  }
  if(id == btRight)
  {
    d->gl.posAll += 0.1f;
    paintGL(p,d);
    pvSetImage(p,id,"1rightarrow2.png");
  }
```

```
  if(id == btUp)
  {
    d->gl.posVertAll -= 0.1f;
    paintGL(p,d);
    pvSetImage(p,id,"1uparrow2.png");
  }
  if(id == btDown)
  {
    d->gl.posVertAll += 0.1f;
    paintGL(p,d);
    pvSetImage(p,id,"1downarrow2.png");
  }

  return 0;
}

static int slotButtonPressedEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  if(id == OpenGL1) d->gl.mouseFirstPressed = 1;

  if(id == btCenter)
  {
    pvSetImage(p,id,"1center3.png");
  }
  if(id == btLeft)
  {
    pvSetImage(p,id,"1leftarrow3.png");
  }
  if(id == btRight)
  {
    pvSetImage(p,id,"1rightarrow3.png");
  }
  if(id == btUp)
  {
    pvSetImage(p,id,"1uparrow3.png");
  }
  if(id == btDown)
  {
    pvSetImage(p,id,"1downarrow3.png");
  }
  return 0;
}

static int slotButtonReleasedEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  if(id == OpenGL1)
  {
    d->gl.mouseFirstPressed = 0;
    d->gl.posAllOld   = d->gl.posAll;
    d->gl.posVertAllOld = d->gl.posVertAll;
  }

  if(id == btCenter)
  {
    pvSetImage(p,id,"1center.png");
  }
  if(id == btLeft)
  {
    pvSetImage(p,id,"1leftarrow.png");
  }
  if(id == btRight)
```

```
  {
    pvSetImage(p,id,"1rightarrow.png");
  }
  if(id == btUp)
  {
    pvSetImage(p,id,"1uparrow.png");
  }
  if(id == btDown)
  {
    pvSetImage(p,id,"1downarrow.png");
  }

  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 == sliderPos)
  {
    d->gl.pos = val/100.0f - 0.5f;
    paintGL(p,d);
  }
  if(id == sliderScale)
  {
    d->gl.scale = 0.5f + 8.0f*(val/100.0f);
    paintGL(p,d);
  }

  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;
  //initializeGL(p);
  return 0;
}

static int slotGlPaintEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  if(id == OpenGL1) paintGL(p,d);
  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;
  if(id == OpenGL1)
  {
    d->gl.width = width;
    d->gl.height = height;
    pvGlBegin(p,id);
    resizeGL(p,width,height);
    pvGlEnd(p);
    //pvGlUpdate(p,id);
    paintGL(p,d);
  }
  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;
  if(id == OpenGL1)
  {
    if(d->gl.mouseFirstPressed == 1)
    {
      d->gl.mouseFirstPressed = 0;
      d->gl.X0 = x;
      d->gl.Y0 = y;
    }
    d->gl.posAll    = d->gl.posAllOld   + ((x - d->gl.X0)/1000.0f);
    d->gl.posVertAll = d->gl.posVertAllOld - ((y - d->gl.Y0)/1000.0f);
    paintGL(p,d);
  }
  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;
  if(id == OpenGL1)
  {
    d->gl.X0 = x;
    d->gl.Y0 = y;
  }
  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 slotMouseOverEvent(PARAM *p, int id, DATA *d, int enter)
{
  if(p == NULL || id == 0 || d == NULL || enter < -1000) return -1;

  if(id == OpenGL1)
  {
    if(enter) pvSetMouseShape(p, OpenHandCursor);
    else      pvSetMouseShape(p, ArrowCursor);
  }
  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;
}
```

6.8.6 VTK

VTK ist ein leistungsfähiges 3D Visualisierungstool, das in C++ geschrieben ist und auf OpenGL basiert. VTK kann komfortabel über die Skriptsprache Tcl programmiert werden.
In pvbrowser ist ein VTK Widget im pvbrowser Client eingebaut, wenn man pvbrowser mit VTK Unterstützung

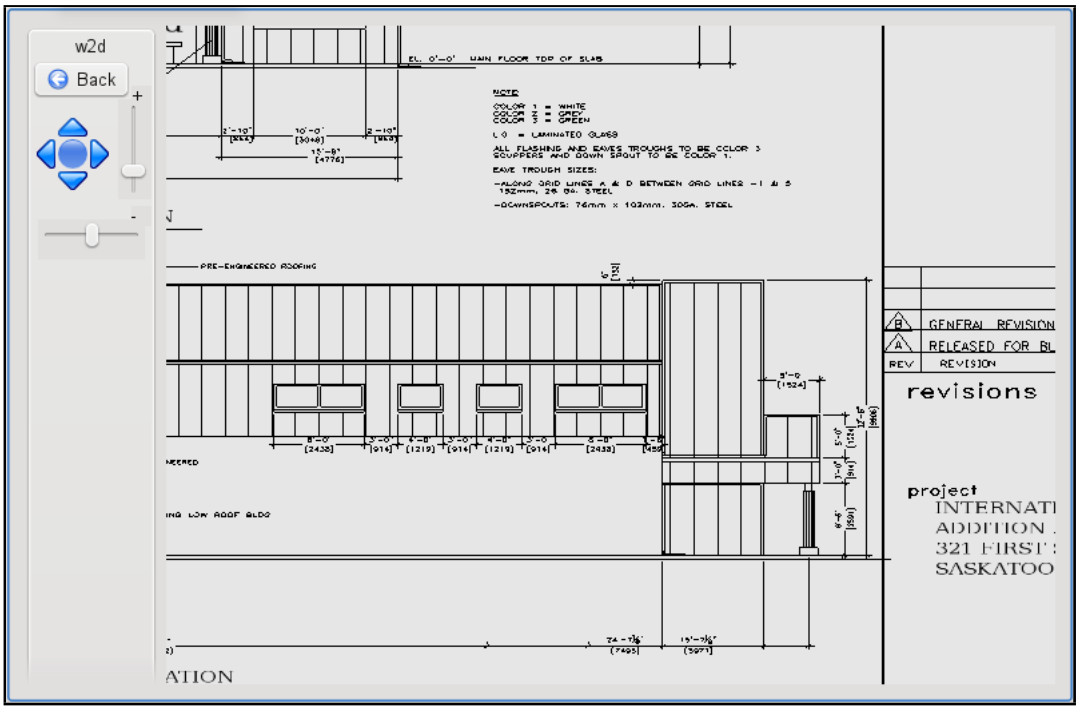

Abbildung 6.44: Autocad Zeichnung in pvbrowser

compiliert hat. Ein pvserver kann nun Tcl Scripte an ein solches Widget senden, um es dort interpretieren zu lassen. Damit kann VTK in Visualisierungen eingebunden werden ohne den pvserver mit einer großen Rechnenlast zu belegen. Das Rendering der 3D Szenen erfolgt vollständig auf dem Client. Der pvserver sendet lediglich einige wenige Tcl Befehle.
Achtung: Sie müssen VTK selbst übersetzen und installieren. Bitte testen Sie dann, ob die mit VTK geliefeten Beispiele funktionieren. Dazu müssen die Umgebungsvariablen korrekt eingestellt sein. Zum Testen verwenden die "wish"(siehe Tcl).

Beispiel für die Visualisierung eines 2D Datensatzes data1.vtk (surface.tcl)

```
# create pipeline

#   create a hue lookup table
vtkLookupTable lut
# blue to red
    lut SetHueRange 0.66667 0.0
    lut Build

# create black to white colormap
vtkLookupTable lbw
#  black to white
    lbw SetHueRange 0 0.0
    lbw SetSaturationRange 0 0
    lbw SetValueRange 0 1

vtkStructuredPointsReader reader
    reader SetFileName "data1.vtk"
    reader Update

#reader needed otherwise range 0..1
    set valuerange [[reader GetOutput] GetScalarRange]
    set minv [lindex $valuerange 0]
    set maxv [lindex $valuerange 1]
#    puts "data range $minv .. $maxv"
```

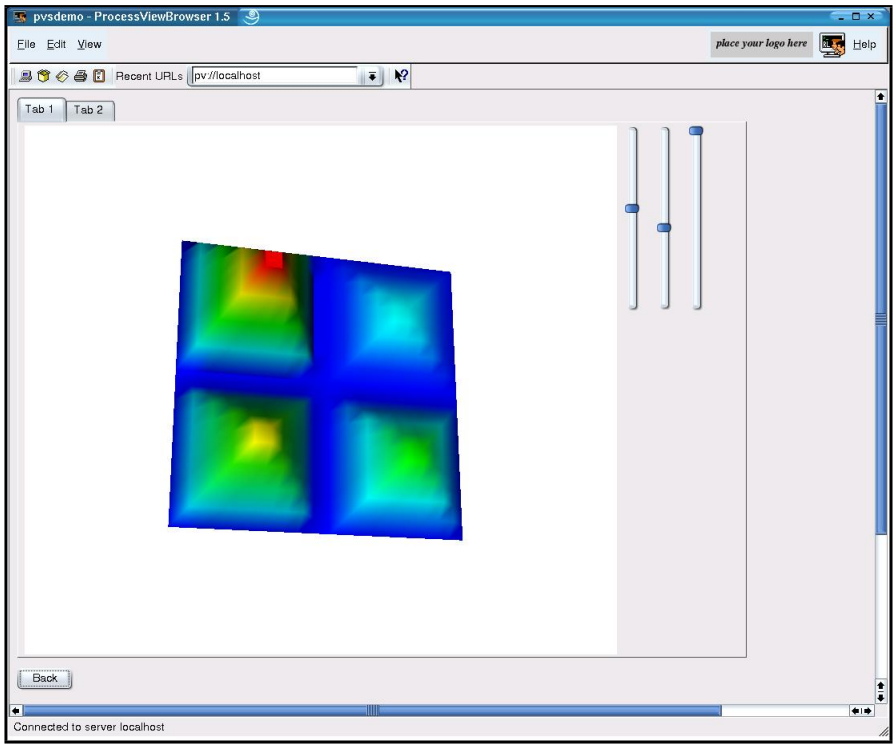

Abbildung 6.45: Darstellung eines 2D Datensatzes data1.vtk

```
    set dims [[reader GetOutput] GetDimensions]
    set dim1 [lindex $dims 0]
    set dim2 [lindex $dims 1]
    set dim3 [lindex $dims 2]
#    puts "dim1 = $dim1 dim2 = $dim2"

# volgende echt nodig ...
# vtkStructuredPointsGeometryFilter plane
vtkImageDataGeometryFilter plane
    plane SetInput [reader GetOutput]
# SetExtent not needed ..

vtkWarpScalar warp
    warp SetInput [plane GetOutput]
    warp UseNormalOn
    warp SetNormal 0.0 0.0 1
    warp SetScaleFactor 1
vtkCastToConcrete caster
    caster SetInput [warp GetOutput]
vtkPolyDataNormals normals
    normals SetInput [caster GetPolyDataOutput]
    normals SetFeatureAngle 60
vtkPolyDataMapper planeMapper
    planeMapper SetInput [normals GetOutput]
    planeMapper SetLookupTable lut
    eval planeMapper SetScalarRange [[reader GetOutput] GetScalarRange]

vtkTransform transform
    transform Scale 0.02 0.02 0.02

vtkActor dataActor
    dataActor SetMapper planeMapper
    dataActor SetUserMatrix [transform GetMatrix]
```

```
    renderer AddActor dataActor
    renderer SetBackground 1 1 1
    set cam1 [renderer GetActiveCamera]

$cam1 ParallelProjectionOff
```

data1.vtk könnte eine Profilmessung sein. Ihr pvserver könnte data1.vtk generieren lassen und den Bildschirm mit den neuen Messungen updaten, wenn Sie verfügbar sind.

data1.vtk

```
# vtk DataFile Version 2.0
2D scalar data
ASCII

DATASET STRUCTURED_POINTS
DIMENSIONS   20   20    1
ORIGIN  -10.000 -10.000 0.000
SPACING   1.000   1.000   1.000

POINT_DATA    400
SCALARS scalars float
LOOKUP_TABLE default
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0
 0.0 1.5 3.0 3.0 3.0 3.0 3.0 3.0 1.5 0.0 0.0 1.0 2.0 2.0 2.0 2.0 2.0 2.0 1.0 0.0
 0.0 1.5 3.0 4.5 4.5 4.5 4.5 3.0 1.5 0.0 0.0 1.0 2.0 3.0 3.0 3.0 3.0 2.0 1.0 0.0
 0.0 1.5 3.0 4.5 6.0 6.0 4.5 3.0 1.5 0.0 0.0 1.0 2.0 3.0 4.0 4.0 3.0 2.0 1.0 0.0
 0.0 1.5 3.0 4.5 6.0 6.0 4.5 3.0 1.5 0.0 0.0 1.0 2.0 3.0 4.0 4.0 3.0 2.0 1.0 0.0
 0.0 1.5 3.0 4.5 4.5 4.5 4.5 3.0 1.5 0.0 0.0 1.0 2.0 3.0 3.0 3.0 3.0 2.0 1.0 0.0
 0.0 1.5 3.0 3.0 3.0 3.0 3.0 3.0 1.5 0.0 0.0 1.0 2.0 2.0 2.0 2.0 2.0 2.0 1.0 0.0
 0.0 1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 0.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 0.0 0.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.0
 0.0 2.0 4.0 4.0 4.0 4.0 4.0 4.0 2.0 0.0 0.0 0.5 1.0 1.0 1.0 1.0 1.0 1.0 0.5 0.0
 0.0 2.0 4.0 6.0 6.0 6.0 6.0 4.0 2.0 0.0 0.0 0.5 1.0 1.5 1.5 1.5 1.5 1.0 0.5 0.0
 0.0 2.0 4.0 6.0 8.0 8.0 6.0 4.0 2.0 0.0 0.0 0.5 1.0 1.5 2.0 2.0 1.5 1.0 0.5 0.0
 0.0 2.0 4.0 6.0 8.0 8.0 6.0 4.0 2.0 0.0 0.0 0.5 1.0 1.5 2.0 2.0 1.5 1.0 0.5 0.0
 0.0 2.0 4.0 6.0 6.0 6.0 6.0 4.0 2.0 0.0 0.0 0.5 1.0 1.5 1.5 1.5 1.5 1.0 0.5 0.0
 0.0 2.0 4.0 4.0 4.0 4.0 4.0 4.0 2.0 0.0 0.0 0.5 1.0 1.0 1.0 1.0 1.0 1.0 0.5 0.0
 0.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 0.0 0.0 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.0
 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Die entsprechenden Slot Funktionen für VTK

```
typedef struct // (todo: define your data structure here)
{
  int xangle;
}
DATA;

static int slotInit(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  d->xangle = 0;
  pvDownloadFile(p,"data1.vtk");
  //...
  pvVtkTclScript(p,VtkTclWidget1,"surface.tcl");
  pvVtkTclPrintf(p,VtkTclWidget1,"dataActor␣RotateX␣%d",0);
  pvVtkTclPrintf(p,VtkTclWidget1,"dataActor␣RotateY␣%d",0);
  pvVtkTclPrintf(p,VtkTclWidget1,"renderer␣Render");
  pvVtkTclPrintf(p,VtkTclWidget1,"reader␣Modified");
  pvVtkTclPrintf(p,VtkTclWidget1,"reader␣Update");
```

```
  pvVtkTclPrintf(p,VtkTclWidget1,"renderer␣Render");
  //...
  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 == Slider1)
  {
    int delta;
    delta = (val-50)*3 - d->xangle;
    d->xangle += delta;
    pvVtkTclPrintf(p,VtkTclWidget1,"reader␣SetFileName␣\"data1.vtk\"");
    pvVtkTclPrintf(p,VtkTclWidget1,"reader␣Modified");
    pvVtkTclPrintf(p,VtkTclWidget1,"reader␣Update");
    pvVtkTclPrintf(p,VtkTclWidget1,"dataActor␣RotateX␣%d",delta);
    pvVtkTclPrintf(p,VtkTclWidget1,"renderer␣Render");
  }
  return 0;
}
```

6.8.7 Grafik Schnittstelle für den Server

Die Klasse rlSvgVdi aus rllib bietet ein virtuelles device interface für den Server, welches SVG verwendet. Sie enthält einige Methoden, die ähnlich zu den in Kapitel "xy Grafiken"beschriebenen Methoden funktionieren.

Die Methoden produzieren gültigen SVG Text, der gesammelt und/oder an eine Zieladresse geschickt werden kann.

Setzen der Zieladresse von rlSvgVdi

```
int setOutput(int *socket_out, int idForPvbrowser=0); // Ausgabe der SVG ber Netzwerk direkt an einen
     pvbrowser client
int setOutput(FILE *fout);                          // Ausgabe der SVG in einen FILE Datenstrom
int setOutput(const char *outputfilename);          // Ausgabe der SVG in eine file.svg Datei
int setOutput(rlSpawn *pipe);                       // Senden der SVG ber eine Pipe zu einem SVG
    Konverter
int endOutput();                                    // Beenden der Ausgabe
```

Benutzen Sie SVG Konverter mit Kommandozeilen Interface, um SVG in png, jpg, pdf ... zu konvertieren.

Einige SVG Konverter:

`https://wiki.gnome.org/action/show/Projects/LibRsvg`

`http://imagemagick.org/script/index.php`

6.9 Dialoge

In pvbrowser sind einige einfache Dialog-boxen verfügbar. Darüber hinaus kann man auch komplexe Dialoge erstellen.

6.9.1 MessageBox

Abbildung 6.46: Eine MessageBox

Message Boxen lassen sich mit der Funktion 'pvMessageBox' programmieren. id_return gibt an, unter welcher id das Ergebnis der MessageBox geliefert werden soll. Sie sollten dafür negative Werte nehmen, um nicht in Konflikt mit den von Ihnen entworfenen Widgets zu kommen. Der Wert des gedrückten Buttons wird in einem 'slotSliderEvent' zurückgeliefert. Wenn Sie weniger als 3 Buttons nutzen möchten, geben Sie für die restlichen Buttons 0 oder MessageBoxNoButton ein.

pvMessageBox

```
int pvMessageBox(PARAM *p, int id_return, int type, const char *text, int button0, int button1, int
    button2);
// mit:
// type  := BoxInformation | BoxWarning | BoxCritical
// button := MessageBoxOk |
//           MessageBoxOpen
//           MessageBoxSave
//           MessageBoxCancel
//           MessageBoxClose
//           MessageBoxDiscard
//           MessageBoxApply
//           MessageBoxReset
//           MessageBoxRestoreDefaults
//           MessageBoxHelp
//           MessageBoxSaveAll
//           MessageBoxYes
//           MessageBoxYesToAll
//           MessageBoxNo
//           MessageBoxNoToAll
//           MessageBoxAbort
//           MessageBoxRetry
//           MessageBoxIgnore
//           MessageBoxNoButton
```

6.9.2 InputDialog

Abbildung 6.47: Ein Input Dialog

Input Dialoge lassen sich mit der Funktion 'pvInputDialog' programmieren. id_return gibt an, unter welcher id das Ergebnis der InputDialog geliefert werden soll. Sie sollten dafür negative Werte nehmen, um nicht in Konflikt mit den von Ihnen entworfenen Widgets zu kommen. Der vom Benutzer eingegebene Text wird in einem 'slotTextEvent' zurückgeliefert.

pvInputDialog

```
int pvInputDialog(PARAM *p, int id_return, const char *text, const char *default_text);
```

6.9.3 FileDialog

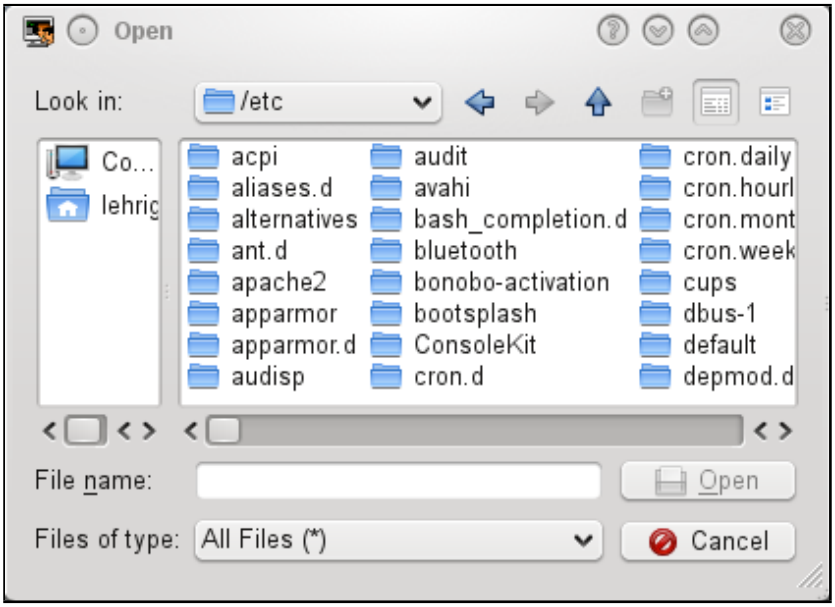

Abbildung 6.48: Ein File Dialog

File Dialoge lassen sich mit der Funktion 'pvFileDialog' programmieren. id_return gibt an, unter welcher id das Ergebnis des FileDialog geliefert werden soll. Sie sollten dafür negative Werte nehmen, um nicht in Konflikt mit den von Ihnen entworfenen Widgets zu kommen. Das Ergebnis wird in einem 'slotTextEvent' zurückgeliefert.

File Dialog

```
int   pvFileDialog(PARAM *p, int id_return, int type);
// mit:
// type := FileOpenDialog | FileSaveDialog | FindDirectoryDialog
```

6.9.4 ModalDialog

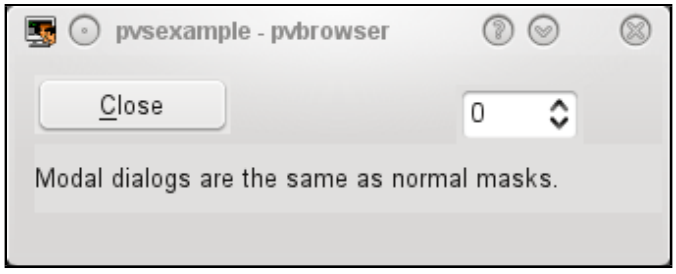

Abbildung 6.49: Ein modaler Dialog

modale Dialoge lassen sich wie normale Masken erstellen. In der aufrufenden Maske wird die Funktion 'pvRunModalDialog' verwendet. Dabei wird die Größe des Dialogfensters mit width, height angegeben. 'showMask' wird mit dem Namen der show Funktion des modalen Dialogs besetzt. 'userData' ist die Adresse einer Datenstruktur, in die der modale Dialog Rückgabe-werte ablegen kann. Falls gewünscht wird, dass die Maske aus der aufgerufen wird, aktualisiert wird, während der modale Dialog läuft, gibt man 'slotNullEvent' für den 'showData' Parameter an. Sonst können 'readData' und 'showData' auf NULL gesetzt werden. Der Parameter 'd' am Ende wird für 'showData' benötigt.

modaler Dialog

```
int  pvRunModalDialog (PARAM *p, int width, int height, int(*showMask)(PARAM *p), void *userData, int
    (*readData)(void *d), int(*showData)(PARAM *p, void *d), void *d);
```

Modalen Dialog aus einer Maske heraus aufrufen

```
pvRunModalDialog(p,330,100,show_mask4,&d->modalInput,NULL,(showDataCast)slotNullEvent,d);
```

Innerhalb von 'slotNullEvent' kann der modale Dialog dann das Basisfenster aktualisieren.

Maske des Basisfensters aus dem modalen Dialog heraus aktualisieren

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

modalen Dialog beenden und Werte an die aufrufende Maske zurückgeben

```
static int slotButtonEvent(PARAM *p, int id, DATA *d)
{
  if(p == NULL || id == 0 || d == NULL) return -1;
  if(id == back)
  {
    int *ptr = (int *) p->modalUserData; // entspricht 'userData' beim Erzeugen des Dialoges
    *ptr = d->input;                 // Rueckgabewerte uebernehmen
                                     // Anstatt eines einfachen int koennte es sich auch um eine
                                     // Datenstruktur mit vielen Parametern handeln
    pvTerminateModalDialog(p);       // Modalen Dialog beenden
  }
  return 0;
}
```

6.9.5 DockWidget

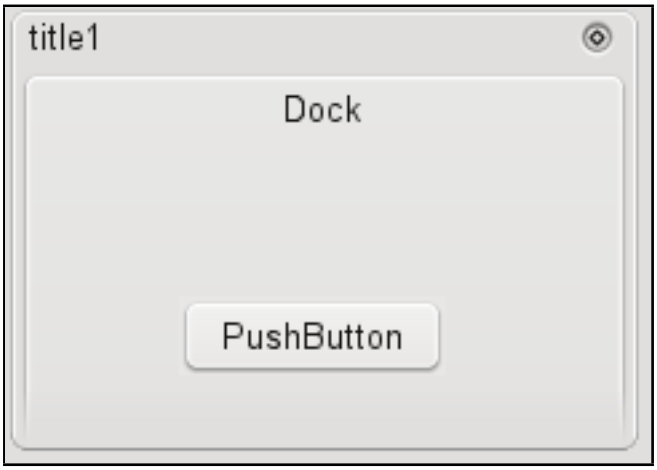

Abbildung 6.50: Dock Widget

Dock Widgets verhalten sich prinzipiell wie nichtmodale Dialoge. Sie können aber innerhalb des Fensters von pvbrowser an den Seiten angedockt werden. Der Inhalt eines Dock Widgets kann an einer Stelle innerhalb der Maske entworfen werden. Sie könnten diesen Bereich z.B. nach rechts außen legen. Dann setzen Sie das Wurzelobjekt dieses Entwurfs (z.B. eine GroupBox) als root_id in das DockWidget ein. Daraufhin verschwindet dieses Objekt in der Maske und wird in das Dock Widget eingesetzt.

Dock Widget hinzufügen

```
int  pvAddDockWidget (PARAM *p, const char *title, int dock_id, int root_id, int allow_close=0, int
    floating=1, int allow_left=1, int allow_right=0, int allow_top=0, int allow_bottom=0)

// mit:
// title      := Titel des Dialogs
// dock_id    := ID_DOCK_WIDGETS + n . Mit n = 0...31 MAX_DOCK_WIDGETS
// root_id    := id des Wurzelobjektes. root_id ist die id eines der entworfenen Widgets.
//               Das root Objekt wird in das Dock Widget eingesetzt und
//               verschwindet daher in der Maske.
//               allow_close := 0|1 dem Benutzer das Verstecken des Dialogs erlauben
//               floating   := 0|1 bewegbar durch den Benutzer
//               allow_X    := 0|1 Docking Positionen
//               Funktionen, die sich auf Dock Widgets anwenden lassen:
//               pvSetGeometry();
//               pvMove();
//               pvResize();
//               pvHide();
//               pvShow();
```

6.9.6 PopupMenu

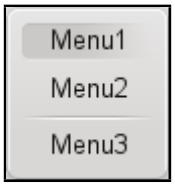

Abbildung 6.51: Popup Menu

Ein Popup Menu kann mit folgendem Code erzeugt werden. Popup Menus lassen sich mit der Funktion 'pvPopupMenu' programmieren. id_return gibt an, unter welcher id das Ergebnis des PopupMenu geliefert werden soll. Sie sollten dafür negative Werte nehmen, um nicht in Konflikt mit den von Ihnen entworfenen Widgets zu kommen. Der vom Benutzer ausgewählte Text wird in einem 'slotTextEvent' zurückgeliefert. Zwei Kommata in dem Text erzeugen einen Separator innerhalb des Menus.

Dock Widget hinzufügen

```
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;
}
```

6.10 Sprachübersetzung

pvbrowser arbeitet vollständig mit UTF-8. Daher können auch Zeichen aus nicht lateinischen Sprachen verwendet werden. So ist es auch möglich kyrillische oder chinesische Zeichen in Ihrer Visualisierung zu verwenden. D.h. Sie können alle in UTF-8 darstellbaren Zeichen verwenden. Der Menu Text in pvbrowser kann über die INI Datei '.pvbrowser.ini' an verschiedene Sprachen angepasst werden.
Während pvbrowser läuft, kann man die Masken von einer Sprache auf die andere umschalten, wenn Sie das in Ihrem pvserver vorsehen. In processviewserver.h wird dazu das Makro '#define pvtr(txt) txt' definiert. Das Makro gibt also einfach den original "txt" zurück. Man kann nun den pvserver erstellen und an jeder Stelle, wo ein Text vorkommt, das Konstrukt 'pvtr("Irgend ein Text")' verwenden.
Wenn man den pvserver dann mehrsprachig machen möchte, kann man dies über eine INI Datei (rlIniFile Klasse) realisieren. Dazu wird rlinifile.h in pvapp.h inkludiert. Das Makro 'pvtr(txt)' wird dabei neu definiert und es wird versucht den Text mit Hilfe einer INI Datei zu übersetzen. In main.cpp fügt man dazu 'rlSetTranslator' am Anfang von main() ein.

Setzen der default Sprache für den pvserver

```
int main(int ac, char **av)
{
PARAM p;
int  s;

  pvInit(ac,av,&p);
  rlSetTranslator("GERMAN","translation.ini");
  /* here you may interpret ac,av and set p->user to your data */
  while(1)
  {
    s = pvAccept(&p);
    if(s != -1) pvCreateThread(&p,s);
    else       break;
  }
  return 0;
}
```

Damit wird die INI Datei 'translation.ini' geladen und darin die default section auf "GERMAN" gesetzt. Jeder Aufruft von 'pvtr("Any text")' würde nun versucht nach Deutsch zu übersetzen.
Während ein Client verbunden ist, kann die Sprache umgestellt werden. Z.B. könnte der Benutzer einen Button drücken und im pvserver würde 'pvSelectLanguage(p,"ENGLISCH");' aufgerufen, um die Sprache für den aktuellen Benutzer auf Englisch umzuschalten.

Setzen der Sprache für einen Client

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

```
    pvSelectLanguage(p,"ENGLISH");
    return 1;
  }
  else if(id == german)
  {
    pvSelectLanguage(p,"GERMAN");
    return 1;
  }
  return 0;
}
```

Nachdem die Sprache mit 'pvSelectLanguage()' umgestellt worden ist, wird im Beispiel ein 'return 1' gemacht und damit ein erneuter Aufbau der Maske in der geänderten Sprache erreicht. Die Sprache entspricht dem Namen einer Sektion in der INI Datei. Sie können in einer INI Datei also beliebig viele Sprachen eingeben. Es dürfte am einfachsten sein, bei Texten konsequent 'pvtr("Text in der Ursprungssprache")' zu verwenden und den pvserver zunächst in der default Sprache zuende zu programmieren. Erst dann könnte man die Übersetzung machen.
Damit man die zu übersetzenden Texte erhält, verfügt pvdevelop über einen spezielles Kommandozeilen Parameter.

Zu übersetzende Text extrahieren

```
pvdevelop -action=dumpTranslations > dump.ini
```

Damit würden die Quellen des pvserver nach 'pvtr(' durchsucht und die Texte extrahiert.

Beispiel dump.ini

```
[aLANGUAGE]
Hallo Welt=
Klick %d\==
```

Daraus kann nun die INI Datei erstellt werden und die Übersetzungen für die anderen Sprachen eingefügt werden.

Übersetzung nach Englisch

```
[ENGLISCH]
Hallo Welt=hello world
Klick %d\==Click %d\=
```

Beachten Sie, dass man auch die Format Anweisungen eines printf verwenden kann. Wenn der zu übersetzende Text ein '=' Zeichen enthält, muss dieses Zeichen mit '\' quotiert werden, damit dies vom '=' zur Trennung von Name und Wert in der INI Datei unterschieden werden kann.
pvdevelop fügt das Makro 'pvtr(txt)' beim Generieren der Masken mit ein. Falls zu einem Text keine Übersetzung vorliegt, wird der Ursprungstext verwendet. In pvbaddon 'pvbaddon/templates/myeventhandler' ist ein Beispiel mit Sprachübersetzung zu finden.
Die PARAM Struktur enthält eine Variable 'int language', in der man sich die gewählte Sprache merken kann. Diese Variable kann man z.B. abfragen, wenn man Werte in landesspezifischen Maßeinheiten darstellen möchte.

Setzen der Sprache in pvserver

```
p->language = GERMAN_LANGUAGE; // standard ist: p->language = DEFAULT_LANGUAGE;
```

6.11 Umrechnung von Einheiten

In der PARAM Struktur gibt es eine Variable 'convert_units'. Sie können p->convert_units auf 0 oder 1 setzen. (standard: 0).
Nun benutzen Sie die Funktion 'float unit(PARAM *p, float val, int conversion);' zur Einheitenumrechnung. Wenn (p->convert_units == 0) ist, wird der original Eingabewert zurückgeliefert.

Einheitenumrechnung

```
val = unit(p, val, MM2INCH);
```

Dies sind die verfügbaren Umrechnungen. Sie könnten p->convert_units in Abhängigkeit von der Sprachauswahl setzen.

Einheiten Umrechnungen

```
enum UNIT_CONVERSION
{
  MM2INCH = 1,
  INCH2MM ,
  CM2FOOT ,
  FOOT2CM ,
  CM2YARD ,
  YARD2CM ,
  KM2MILE ,
  MILE2KM ,
  KM2NAUTICAL_MILE ,
  NAUTICAL_MILE2KM ,
  QMM2SQINCH ,
  SQINCH2QMM ,
  QCM2SQFOOT ,
  SQFOOT2QCM ,
  QM2SQYARD ,
  SQYARD2QM ,
  QM2ACRE ,
  ACRE2QM ,
  QKM2SQMILE ,
  SQMILE2QKM ,
  ML2TEASPOON ,
  TEASPOON2ML ,
  ML2TABLESPOON ,
  TABLESPOON2ML ,
  ML2OUNCE ,
  OUNCE2ML ,
  L2CUP ,
  CUP2L ,
  L2PINT ,
  PINT2L ,
  L2QUART ,
  QUART2L ,
  L2GALLON ,
  GALLON2L ,
  GR2OUNCE ,
  OUNCE2GR ,
  KG2POUND ,
  POUND2KG ,
  T2TON ,
  TON2T ,
  C2FAHRENHEIT ,
  FAHRENHEIT2C
};
```

6.12 Layout Management

Das Layout Management kann in pvdevelop festgelegt werden. Wählen Sie dazu das entsprechende Menu (rechte Maustaste) im graphischen Designer von pvdevelop aus. Beim Entwurf der Masken können Sie die Min- und Max- Werte für die Dimension der Widgets setzen, um zu erreichen, dass ein Widget nur innerhalb gewisser Grenzen verändert wird. Wenn sie Min- und Max- Werte identisch setzen, wird die Größe des Widgets nicht verändert.
Hier ist ein Beispiel-code für das Layout Management.

Layout Management

```
pvQLayoutHbox(p,ID_MAIN_WIDGET,-1);             // horizontally layout all widgets
```

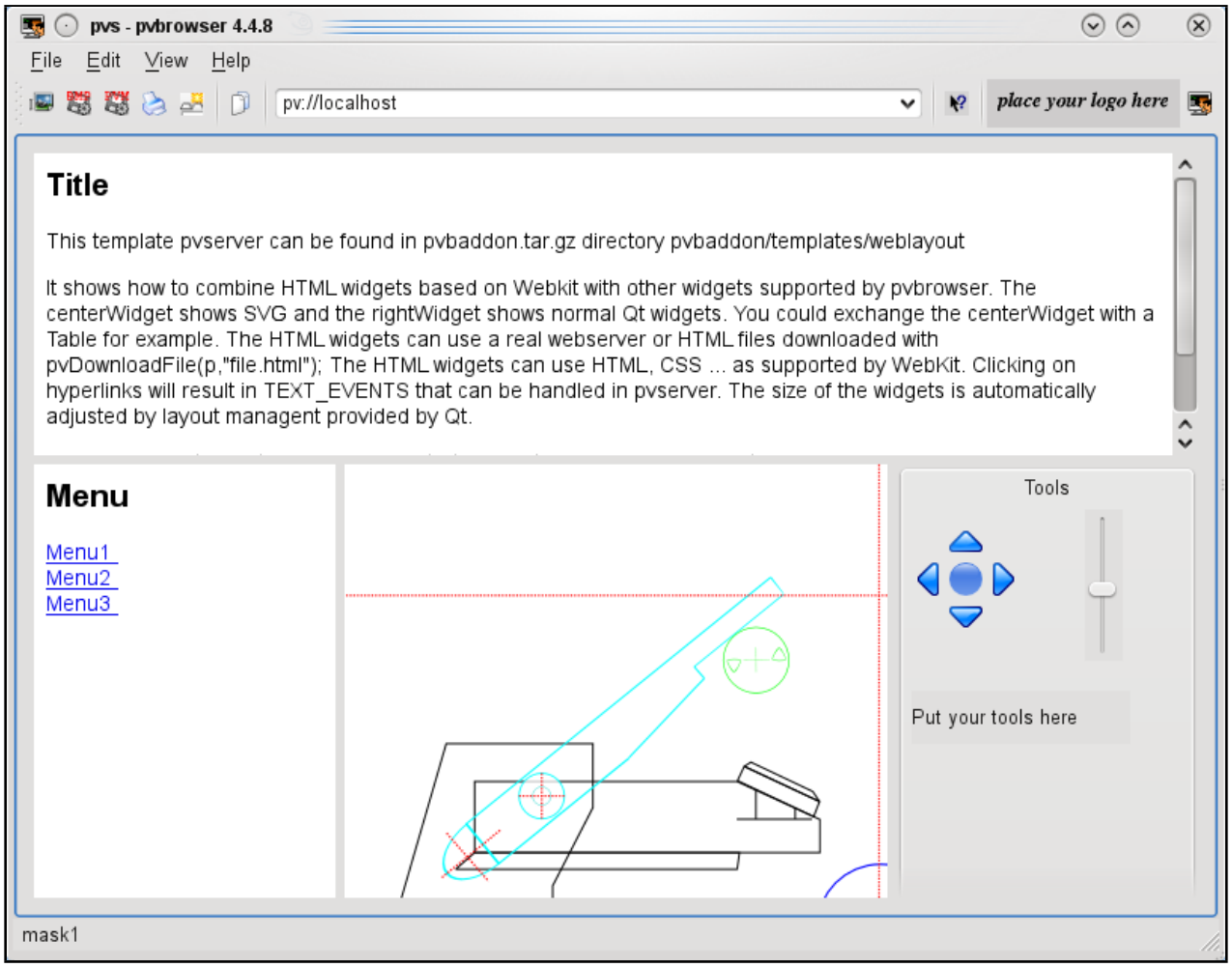

Abbildung 6.52: Layout für Beispiel-code

```
pvQLayoutVbox(p,layout1,ID_MAIN_WIDGET);          // create a vertical box layout
                                                  // parent is main widget

pvQLayoutHbox(p,layout2,layout1);                 // create a horizontal box layout
                                                  // parent is layout1

pvAddWidgetOrLayout(p,ID_MAIN_WIDGET,layout1,-1,-1); // put layout1 into the main layout
pvAddWidgetOrLayout(p,layout1,upperWidget,-1,-1); // add the upperWidget
pvAddWidgetOrLayout(p,layout1,layout2,-1,-1);     // add layout2 below the upperWidget
pvAddWidgetOrLayout(p,layout2,leftWidget,-1,-1); // add the reamining widgets from left to right
pvAddWidgetOrLayout(p,layout2,centerWidget,-1,-1);
pvAddWidgetOrLayout(p,layout2,rightWidget,-1,-1);
```

6.13 Setzen der Tab Order

Die Tab Order kann in pvdevelop festgelegt werden. Wählen Sie dazu das entsprechende Menu (rechte Maustaste) in graphischen Designer von pvdevelop aus. Dazu klicken Sie die Objekte in der gewünschten Reihenfolge. Die schon angeklickten Objekte werden dabei versteckt.

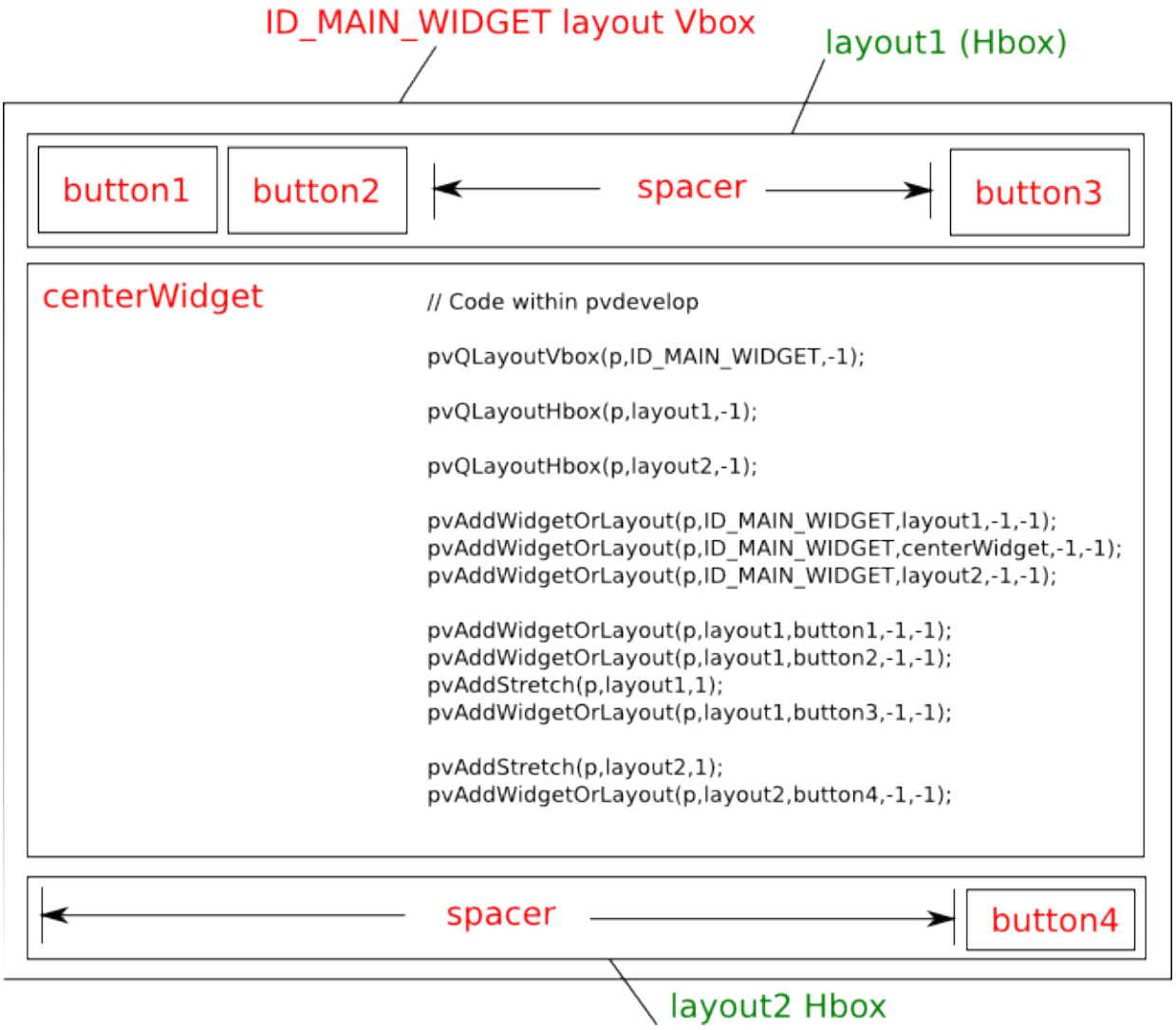

Abbildung 6.53: Layout Beispiel

6.14 Benutzung von Stylesheets in pvbrowser

Sowohl Qt Widgets als auch HTML Elemente können mit Stylesheets beeinflusst werden. Dies könnte eine Möglichkeit sein, um eine Visualisierung an unterschiedliche Anforderungen auf normalen PC und Mobilgeräten anzupassen.
Um Qt Widgets zu designen verwenden Sie bitte die Funktion

pvSetStyleSheet

```
int pvSetStyleSheet(PARAM *p, int id, const char *text);
```

Beispiel: Den Hintergrund eines Button auf rot setzen

```
pvSetStyleSheet(p,pb,"background:␣red");
```

Für den text könnte man ein rlString Objekt verwenden, wenn es sich um einen längeren Text handelt. Mit der

read Methode des rlString

```
rlString::read(const char *filename);
```

kann man den Stylesheet text aus einer Datei lesen.

Setzen des Style aus einer Datei

```
rlString pbstyle;
pbstyle.read("pbstyle.css");
pvSetStyleSheet(p,pb, pbstyle.text());
```

Beispiele und eine detaillierte Referenz der Qt Stylesheets findet man auf:
`http://doc.qt.io/qt-5.5/stylesheet-examples.html`
`http://doc.qt.io/qt-5.5/stylesheet-reference.html`
pvbrowser kann auch Standard HTML Elemente innerhalb des TextBrowser Objektes verwenden. Siehe den Konstruktor des TextBrowser.

CSS Stylesheets können für HTML Inhalte verwendet werden

```
int pvQTextBrowser(PARAM *p, int id, int parent);
```

Im Internet findet man viele Dokumentationen mit Informationen über CSS Stylesheets.
Beispielsweise:
`https://wiki.selfhtml.org/wiki/CSS`

6.15 Webcam

Webcams, die auf Motion JPEG Streams über http basieren, können mit Hilfe der Klasse rlWebcam genutzt werden.
Falls Sie nicht wissen, unter welcher URL die Webcam den M-JPEG Video Stream anbietet, können Sie das mit Hilfe von tcpdump herausfinden.

Einsatz von tcpdump

```
tcpdump -X -i eth0 -t -q -s 0 "host␣192.168.1.200␣&&␣port␣80" | grep -A 10 GET
IP myhost.46727 > 192.168.1.200.http: tcp 97
 0x0000: 4500 0089 edb6 4000 4006 c891 c0a8 010e E.....@.@.......
 0x0010: c0a8 01c8 b687 0050 d99f 8b7d 0003 d5b2 .......P...}....
 0x0020: 5018 16d0 2460 0000 4745 5420 2f63 6769 P...$‘..GET./cgi
 0x0030: 2d62 696e 2f53 7472 6561 6d3f 5669 6465 -bin/Stream?Vide
 0x0040: 6f20 4175 7468 6f72 697a 6174 696f 6e3a o.Authorization:
 0x0050: 2042 6173 6963 2059 5752 7461 5734 3663 .Basic.YWRtaW46c
 0x0060: 4746 7a63 3364 7663 6d51 3d3f 7765 6263 GFzc3dvcmQ=?webc
 0x0070: 616d 5057 443d 526f 6f74 436f 6f6b 6965 amPWD=RootCookie
 0x0080: 3030 3030 300d 0a0d 0a               00000....
```

Webcam Einbindung in pvserver

```
include "rlwebcam.h"
typedef struct // (todo: define your data structure here)
{
  rlWebcam webcamBig;
}
DATA;
static int slotInit(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  p->sleep = 20;
  p->force_null_event = 0;
  d->webcamBig.debug = 0;
  d->webcamBig.filename.printf("%swebcam.jpg", p->file_prefix);
  d->webcamBig.setUrl("http://192.168.1.200/cgi-bin/Stream?Video␣Authorization:␣Basic␣
      YWRtaW46cGFzc3dvcmQ=?webcamPWD=RootCookie00000");
  return 0;
}
static int slotNullEvent(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  if(const char *fname = d->webcamBig.getFrame()) // OR if(const char *fname = d->webcamBig.
      getSnapshot())
  {
    pvDownloadFileAs(p,fname,"webcam.jpg");
    pvSetImage(p,WebcamBig,"webcam.jpg"); // WebcamBig is a pvQImage object that accepts jpeg images
  }
  return 0;
}
```

6.16 Cookies

Cookies sind kurze Informationen, die auf dem Client Rechner gespeichert werden können, wenn der pvbrowser Client so eingestellt ist, diese Cookies zu akzeptieren.

Beispiel zu Cookies

```
pvPrintf(p,ID_COOKIE,"%s=%s","cookie_name","cookie_values"); // Cookie "cookie_name" setzen
// snip
pvPrintf(p,ID_COOKIE,"cookie_name"); // Cookie "cookie_name" erfragen
                                     // Als Ergebnis wird mit einem Text Event unter
                                     // ID_COOKIE geantwortet
```

6.17 Duale httpd und pvserver Funktionalität

Neben dem pvserver Protokoll pv für den pvbrowser client kann ein pvserver auch als httpd für einen standard Webbrowser mit http Protokoll genutzt werden. Dazu muss der pvserver mit der '-http' Option gestartet werden. Das unterdrückt das automatische Senden der pvserver Version an den Client. Im Falle, das es sich um einen pvbrowser Client handelt, kann pvSendVersion() aufgerufen werden.

pvMain() für einen dualen http und pvserver

```
int pvMain(PARAM *p)
{
int ret;

  ret = 1;
  pvGetInitialMask(p);
  if(trace) printf("p->url=%s\n", p->url);
  if(strncmp(p->url,"GET␣",4) != 0) // test if we got a http GET request
  {
    pvSendVersion(p); // if a pvbrowser client is connected
    pvSetCaption(p,"pvs");
  }

  while(1)
  {
    if(trace) printf("show_mask%d\n", ret);
    switch(ret)
    {
      case 1:
        ret = show_mask1(p);
        break;
      default:
        return 0;
    }
  }
}
```

Im Falle eines http request wird generated_defineMask() in maskX.cpp nicht aufgerufen

```
static int defineMask(PARAM *p)
{
  if(p == NULL) return 1;
  if(strncmp(p->url,"GET␣",4) == 0) return 0; // if it is a http GET request then return
  generated_defineMask(p);
  // (todo: add your code here)
  return 0;
}
```

slotFunctions für einen dualen http und pvserver

```
rlString header("<html><head><meta␣http-equiv=\"refresh\"␣content=\"1;\"></head><body>");
```

```
rlString trailer("</body></html>");

typedef struct // (todo: define your data structure here)
{
  int i;
}
DATA;

static int slotInit(PARAM *p, DATA *d)
{
  if(p == NULL || d == NULL) return -1;
  //memset(d,0,sizeof(DATA));
  d->i = 0;
  if(strncmp(p->url,"GET␣",4) == 0) // test if we got a http GET request
  {
    char buf[MAX_EVENT_LENGTH];
    while(1) // read the http header that follows the http url
    { // here we simply ignore the http header
      pvtcpreceive(p,buf, sizeof(buf) -1);
      if(trace) printf("while␣http␣header␣buf=%s\n", buf);
      if(strlen(buf) < 3) break;
    }

    if(trace) printf("send␣response␣from␣slotInit\n");
    rlString body;
    body.printf("<p>Hello␣World␣from␣slotInit␣p->url=\"%s\"␣d->i=%d</p>", p->url, d->i);
    rlString html;
    html += header;
    html += body;
    html += trailer;
    pvSendHttpResponse(p, html.text());
  }
  else
  {
    // ... snip normal pvserver code
  }
  return 0;
}

// ... snip

static int slotTextEvent(PARAM *p, int id, DATA *d, const char *text)
{
  if(p == NULL || id == 0 || d == NULL || text == NULL) return -1;
  if(id == -1 && strncmp(text,"GET␣",4) == 0) // test if we got a http GET request
  {
    char buf[MAX_EVENT_LENGTH];
    while(1) // read the http header that follows the http url
    { // here we simply ignore the http header
      pvtcpreceive(p,buf, sizeof(buf) -1);
      if(trace) printf("while␣buf=%s\n", buf);
      if(strlen(buf) < 3) break;
    }

    if(trace) printf("send␣response␣from␣slotTextEvent\n");
    // this would send a file
    // sprintf(buf,"HTTP/1.1 200 OK\n");
    // pvtcpsendstring(p,buf);
    // sprintf(buf,"Server: pvserver-%s\n", pvserver_version);
    // pvtcpsendstring(p,buf);
    // sprintf(buf,"Keep-Alive: timeout=15, max=100\n");
    // pvtcpsendstring(p,buf);
    // sprintf(buf,"Connection: Keep-Alive\n");
```

```
    // pvtcpsendstring(p,buf);
    // sprintf(buf,"Content-Type: text/html\n");
    // pvtcpsendstring(p,buf);
    // pvSendHttpContentLength(p,"test.html");

    // this will send html text
    rlString body;
    body.printf("<p>Hello␣World␣from␣slotTextEvent␣text=\"%s\"␣d->i=%d</p>", text, d->i);
    rlString html;
    html += header;
    html += body;
    html += trailer;
    pvSendHttpResponse(p, html.text());
    d->i++;
  }
  else
  {
    // ... snip normal pvserver code
  }
  return 0;
}
```

Kapitel 7

Datenerfassung

Die Datenerfassung in pvbrowser wird über separate Daemonen (Hintergrundprogramme) realisiert. Diese Daemonen 'sprechen' das Protokoll des jeweiligen Feldbusses bzw. der SPS. Ein Daemon besteht aus zwei Threads, wobei ein Thread die Daten zyklisch einliest und das Ergebnis in ein Shared Memory schreibt und der andere Thread auf einer Mailbox wartet, um Befehle zur Ausgabe auf die Schnittstelle zu empfangen. Die Visualisierung kann nun das Shared Memory lesen und dessen Inhalt visualisieren. Wenn Signale ausgegeben werden sollen, sendet die Visualisierung eine entsprechende Botschaft an die Mailbox.

Ein Vorteil dieser Architektur ist, dass Datenerfassung und Visualisierung voneinander getrennt sind. Die Visualisierung kann z.B. neu gestartet werden, ohne alle Eingänge neu einlesen zu müssen, denn die Eingangsgrößen bleiben im Shared Memory erhalten und deren Einlesen läuft weiter, obwohl man die Visualisierung stoppt, um darin Änderungen zu machen. Ausserdem ist es problemlos möglich mehrere Daemonen gleichzeitig laufen zu lassen, wobei jedem Daemonen ein eigenes Shared Memory und eine Mailbox zugeordnet sind. Bei den Protokollen ist man damit auch variabel. Es ist denkbar mehrere Daemonen mit jeweils unterschiedlichen Protokollen zu verwenden.

ACHTUNG: Unter unixartigen Betriebssystemen muss man das alte Shared Memory bzw. die Mailbox löschen, wenn man deren Größe geändert hat.

```
# Hilfe zu den Befehlen
man ipcs
man ipcrm
# ipc objekte auflisten
ipcs
# shared memory mit id=425986 loeschen
ipcrm -m 425986
# mailbox mit id=32769 loeschen
ipcrm -q 32769
```

In pvbaddon sind einige Daemonen enthalten, die lediglich über INI Dateien parametriert werden müssen. Diese Daemonen arbeiten mit der Klasse rlDataAcquisitionProvider aus der rllib. Auf der Seite des pvserver kann dann mit der Klasse rlDataAcquisition auf das Shared Memory und die Mailbox zugegriffen werden. Die Variablen werden dabei als lesbarer ASCII Text dargestellt.

In pvdevelop gibt es darüber hinaus die Möglichkeit Daemonen zu generieren, die die Variablen binär darstellen.

Es sollte im Einzelfall entschieden werden, ob der Overhead der ASCII Darstellung akzeptiert werden kann oder ob man lieber binär kodierte Variablen verwendet. Die Darstellung als ASCII Text macht die Programmierung einfacher, während man auf die binäre Darstellung schneller zugreifen kann.

Falls neue Protokolle implementiert werden sollen, kann man einen der vorhandenen Daemonen als Vorlage nehmen und entsprechend dem neuen Protokoll umschreiben. Am besten geeignet hierfür sollte der Modbus Daemon sein.

Ihnen steht es also offen auch Protokolle zu verwenden, die noch nicht von pvbrowser unterstützt werden. Sie benötigen dazu lediglich eine Bibliothek, die das Protokoll implementiert. Das können sowohl fremde Bibliotheken, die in C oder C++ geschrieben sind sein oder Bibliotheken, die Sie auf Basis der anderen Klassen der rllib selber geschrieben haben. Falls Sie diesen Weg gehen sollten, bitten wir Sie uns Ihre Implementierung zuzusenden, damit wir das in unser Projekt aufnehmen können.

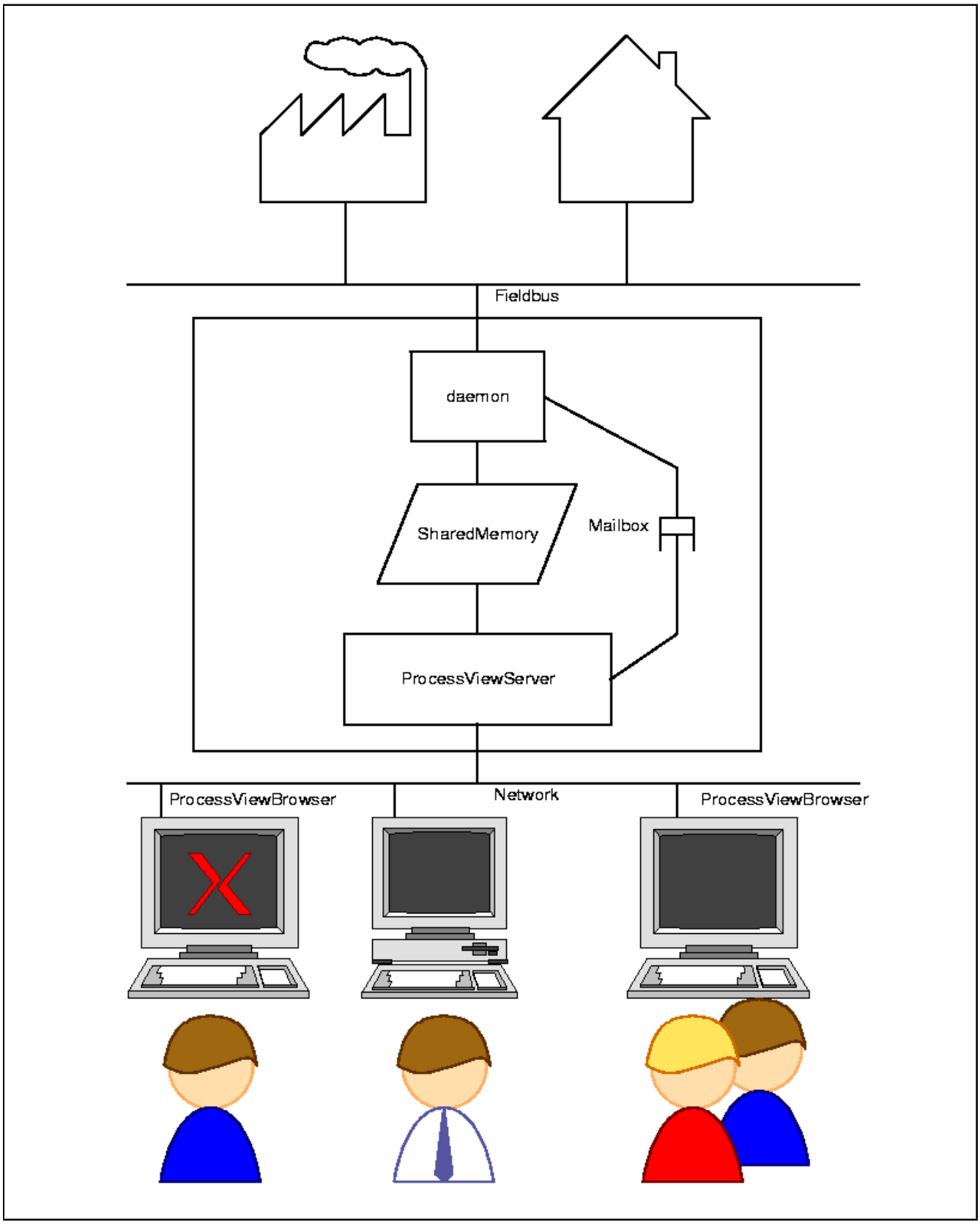

Abbildung 7.1: Prinzip der Datenerfassung mit pvbrowser

7.0.1 Kopieren der Daemonen in ein standard Verzeichnis

Die Daemonen aus pvbaddon sollten in ein Verzeichnis kopiert werden, das von der $PATH Environment Variable Ihres Betriebssystems erfasst wird, weil Sie den Daemon dann einfach durch Eingabe seines Namens in der Kommadozeile starten können.

Linux

```
cp ihren_daemon /usr/bin/
```

Windows

```
copy ihren_daemon.exe %PVBDIR%\win-mingw\bin\
```

7.0.2 Die INI Datei für den Daemon

Die INI Datei für den Daemon kann an einem beliebigen Ort gespeichert werden. Starten Sie den Daemon mit folgendem Kommado.

Start eines Daemon aus einem Terminal / Dos Box

```
ihr_daemon /pfad/zu/ihre_config.ini
```

Benutzen Sie zum Testen bitte ein Terminal / Dos Box. Falls alles beim Booten des Rechners gestartet werden soll lesen Sie bitte 'Einrichtung eines pvserver zum Start im Hintergrund' in diesem Dokument.

7.0.3 Konfiguration des shared memory und der mailbox

Die Beispiele in pvbaddon schlagen folgende Orte für das shared memory und die mailbox vor.

Linux

```
/srv/automation/shm (shared memory)
/srv/automation/mbx (mailbox)
```

Windows

```
c:\automation\shm (shared memory)
c:\automation\mbx (mailbox)
```

Beachten Sie, dass Sie diese Verzeichnisse zunächst selbst anlegen müssen. Beachten Sie, dass die Größe des shared memory im Daemon und im pvserver identisch sein muss.

7.0.4 Daemon und pvserver zum Testen starten

Öffnen Sie bitte 2 Terminals / Dos Boxen. Starten Sie den Daemon im ersten Fenster und den pvserver im zweiten Fenster. Jeder pvserver kennt einige Kommandozeilen Optionen.

Kommandozeilen Optionen Ihres pvserver anzeigen

```
./pvsexample --help
```

Starten eines pvserver

```
ihr_pvserver -cd=/pfad/zu/ihrem/projekt
```

Fall die '-cd' (change directory) Option nicht angegeben wird, verwendet der pvserver das aktuelle Verzeichnis.

7.1 Modbus

Falls Sie zunächst mehr über die Grundlagen von Modbus erfahren möchten, lesen Sie bitte `http://de.wikipedia.org/wiki/Modbus` .
Modbus hat den Vorteil, dass dessen Spezifikation öffentlich zugänglich ist und daher nicht auf 'reverse Engineering' zurückgegriffen werden muss, wenn man das Protokoll implementieren will. Das hat dazu geführt, dass eine Vielzahl von Hardware verfügbar ist, die dieses Protokoll beherrscht. Modbus ist deshalb die bevorzugte Methode zur Datenerfassung in vielen Systemen, so auch in pvbrowser.
Modbus gibt es in mehreren Varianten. Aber immer gibt es einen Master am Bus und bis zu 255 Slaves. Zunächst kann Modbus über serielle Verbindungen mit RS485 gefahren werden. Hier gibt es wieder die Methoden Modbus RTU und Modbus Ascii. Darüber hinaus kann Modbus inzwischen auch über TCP verwendet werden. Eine Kombination von Beidem ist möglich, indem ein Gateway verwendet wird, das auf der TCP Seite einen Modbus TCP Slave implementiert und auf der RS485 Seite einen Modbus RTU Master darstellt.
Die Klasse rlModbus aus der rllib implementiert alle dieser Modbus Varianten. Indem ein rlSerial bzw. ein rlSocket Objekt in rlModbus registriert wird, kann man auswählen, ob die serielle Schnittstelle oder TCP verwendet werden soll. rlModbus dient als Basis zur Implementierung der Daemonen für Modbus.
In rlModbus werden alle Daten Adressen in den Modbus Botschaften auf 0 bezogen, mit dem ersten Wert beginnend bei 0. Das Funktionscode Feld spezifiziert, welche Register Gruppe angesprochen werden soll (z.B. 0x, 1x, 3x, or 4x Referenz Adressen). Beispielsweise wird Holding Register 40001 als Register 0000 im Daten Adressen Feld angegeben. Der Funktions Code, der dieses Register anspricht bedeutet ein "Holding Registerïm Adressbereich "4xxxx". Damit wird Holding Register 40108 über die Daten Adresse 006B Hex (107 Dezimal) adressiert.

Modbus Register Zuordnung

```
0xxxx Read/Write diskrete Outputs oder Coils.
1xxxx Read discrete Inputs.
3xxxx Read Input Registers.
4xxxx Read/Write Output oder Holding Registers.
```

7.1.1 Zugriff über lesbare ASCII Zeichen

Im Verzeichnis pvbaddon/daemons/modbus/client in pvbaddon findet man den Modbus Daemon, der mit lesbaren ASCII Zeichen als Variablen arbeitet. In der INI Datei wird angegeben, was der Daemon lesen soll.

INI Datei für Modbus Daemon

```
# ini file for modbus_client
#
# USE_SOCKET := 1 | 0 # if 0 then USE_TTY
# DEBUG     := 1 | 0
# BAUDRATE  := 300   |
#              600   |
#              1200  |
#              1800  |
#              2400  |
#              4800  |
#              9600  |
#              19200 |
#              38400 |
#              57600 |
#              115200
# STOPBITS  := 1 | 2
# PARITY    := NONE | ODD | EVEN
# PROTOCOL  := RTU | ASCII
# CYCLE<N>  := <count>,<name>
# name      := coilStatus(slave,adr)    |
#              inputStatus(slave,adr)   |
#              holdingRegisters(slave,adr) |
#              inputRegisters(slave,adr)
# CYCLETIME in milliseconds
# SHARED_MEMORY_SIZE must be equal to SHARED_MEMORY_SIZE of pvserver
```

```
# MAX_NAME_LENGTH is maximum length of variable name in shared memory
#

[GLOBAL]
USE_SOCKET=1
DEBUG=1
CYCLETIME=1000
N_POLL_SLAVE=0 # number of cycles a slave will not be polled when it fails

[SOCKET]
IP=localhost
PORT=5502
xxxIP=169.254.200.9
xxxPORT=502

[TTY]
DEVICENAME=/dev/ttyUSB0
BAUDRATE=9600
RTSCTS=1
STOPBITS=1
PARITY=NONE
PROTOCOL=RTU

[RLLIB]
MAX_NAME_LENGTH=30
SHARED_MEMORY=/srv/automation/shm/modbus1.shm
SHARED_MEMORY_SIZE=65536
MAILBOX=/srv/automation/mbx/modbus1.mbx

[CYCLES]
NUM_CYCLES=4
CYCLE1=10,inputStatus(1,0)
CYCLE2=8,coilStatus(1,0)
CYCLE3=2,holdingRegisters(1,0)
CYCLE4=2,inputRegisters(1,0)
```

Über USE_SOCKET wird ausgewählt, ob die serielle Schnittstelle oder TCP verwendet werden soll. Bei USE_SOCKET=1 ist der Abschnitt [SOCKET] relevant und bei USE_SOCKET=0 der [TTY] Abschnitt. Über DEBUG können Meldungen des Daemon ein- und aus-geschaltet werden. Im Abschnitt [RLLIB] wird das Shared Memory und die Mailbox definiert. Beachten Sie, dass das Beispiel für unix-artige Betriebssysteme gilt. Unter Windows müssen [TTY][DEVICENAME], [RLLIB][SHARED_MEMORY] und [RLLIB][MAILBOX] in Windows Syntax angegeben werden.

Beispiel für Windows Syntax

```
DEVICENAME=COM1
SHARED_MEMORY=c:\automation\shm\modbus1.shm
MAILBOX=c:\automation\mbx\modbus1.mbx
```

Beachten Sie, dass die Verzeichnisse für das Shared Memory und die Mailbox bereits existieren müssen.
Im Abschnitt [CYCLES] werden in dem Beispiel 4 Zyklen angelegt. In CYCLE1 werden 10 aufeinanderfolgende inputStatus von Slave=1 und Adresse=0 gelesen. In CYCLE2 werden 8 aufeinanderfolgende coilStatus von Slave=1 und Adresse=0 gelesen. In CYCLE3 werden 2 aufeinanderfolgende holdingRegister von Slave=1 und Adresse=0 gelesen. In CYCLE4 werden 2 aufeinanderfolgende inputRegister von Slave=1 und Adresse=0 gelesen.
Nachdem Sie die INI Datei für Ihr System erstellt haben, starten Sie den Daemon bitte mit eingeschaltetem DEBUG=1 auf der Kommandozeile. Sie können dann anhand der DEBUG Ausgaben sehen, ob die Werte korrekt gelesen werden.
Im Verzeichnis pvbaddon/daemons/modbus/pvs in pvbaddon findet man einen pvserver, in dem gezeigt wird, wie auf das Shared Memory und auf die Mailbox zugegriffen werden muss.

Zugriff auf Shared Memory und Mailbox aus einem pvserver

```
rlDataAcquisition *acqui;
```

```
// snip
acqui = new rlDataAcquisition("/srv/automation/mbx/modbus1.mbx",
                              "/srv/automation/shm/modbus1.shm",65536);
// snip
int val = acqui->intValue("holdingRegisters(1,1)"); // read shared memory
                                                    // holdingRegister slave=1 adr=1
// snip
acqui->writeIntValue("coil(1,0)", value); // send a coil over mailbox to daemon
```

7.1.2 Zugriff über binär kodierte Werte

In pvdevelop gibt es einen Dialog, um einen Modbus Daemon zu generieren, der mit binär kodierten Werten arbeitet. In diesem Dialog kann man spezifizieren, was gelesen werden soll. Über 'communication=serial' bzw. 'communication=socket' können Sie wählen, welche Schnittstelle für die Modbus Kommunikation verwendet werden soll. Setzen Sie einen Kommentar vor der nicht zu benutzenden Methode. Auch hier können Sie wieder eine Reihe von 'cycle' definieren, mit denen Werte über Modbus zyklisch gelesen werden sollen und dann im Shared Memory abgelegt werden.

Sie müssen beachten, dass Sie unter Windows wieder die entsprechende Windows Syntax verwenden. Dabei ist darauf zu achten, dass für den '\' in den Pfaden 2 '\\' angegeben werden muss, denn dieser Text resultiert in C++ Quelltext, im dem der Text eingesetzt wird. Wenn Sie die Dialogbox beenden, wird die Datei modbusdaemon.cpp im aktuellen Verzeichnis generiert und compiliert. Das ergibt einen Modbus Daemon, der die in der Dialogbox eingegeben Konfiguration enthält und verwendet werden kann.

Starten Sie diesen Daemon zunächst auch in der Kommandozeile, um zu prüfen, ob alles wie gewünscht funktioniert. Die zyklisch gelesenen Werte werden dabei kompakt hintereinander ins Shared Memory geschrieben.

Der Offset für jeden Zyklus ist aus modbusdaemon.h zu entnehmen. modbusdaemon.h wird von pvdevelop generiert und enthält einige Definitionen, die in Ihrem Quelltext verwendet werden sollen. Das sind z.B. der Name und die Größe des Shared Memory und der Name der Mailbox. Darüber hinaus finden sich darin die Offsets im Shared Memory unter denen die Werte für die einzelnen Zyklen abgelegt sind.

Im Verzeichnis pvbaddon/demos/modbusserial in pvbaddon findet man ein Beispiel.

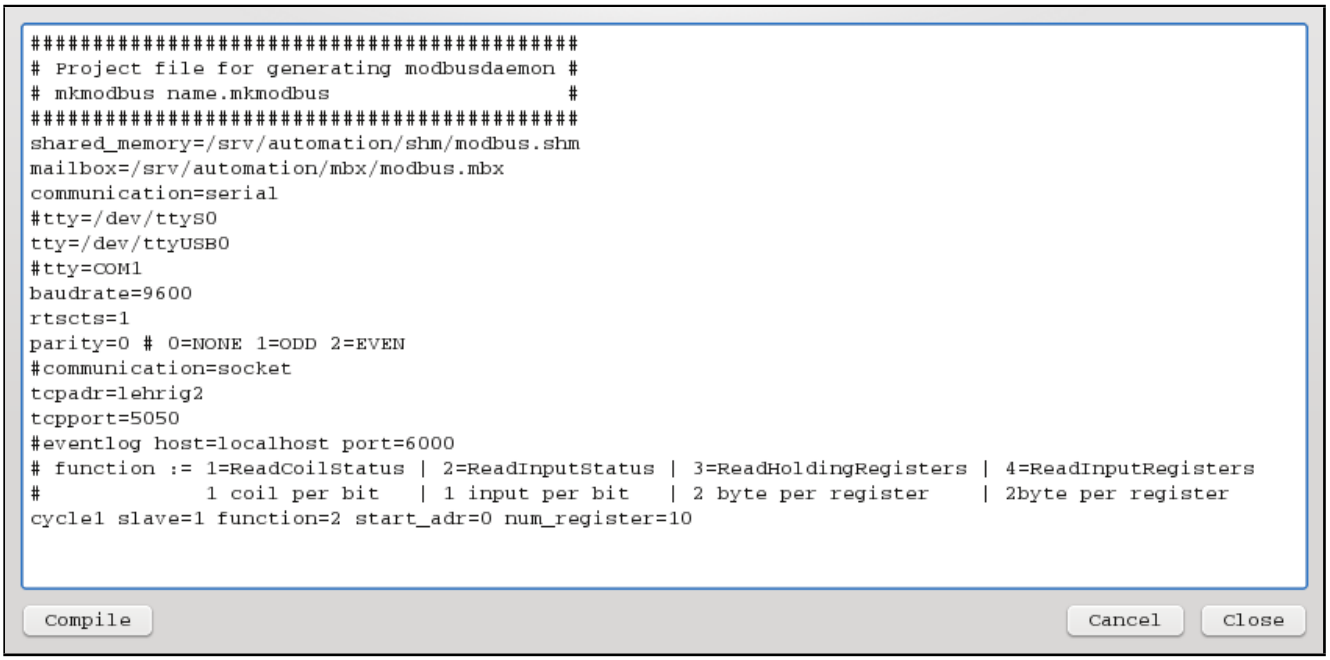

Abbildung 7.2: Dialog zum Generieren eines modbusdaemon in pvdevelop

Zugriff auf Shared Memory und Mailbox aus einem pvserver mit binär kodierten Werten

```
#include "rlmodbusclient.h"
#include "modbusdaemon.h"          // this is generated
rlModbusClient   modbus(modbusdaemon_MAILBOX,modbusdaemon_SHARED_MEMORY,
    modbusdaemon_SHARED_MEMORY_SIZE);
// snip
int val = modbus.readBit(modbusdaemon_CYCLE1_BASE,i+4); // the first 4 bits are outputs
// snip
modbus.writeSingleCoil(1,address,val); // write to modbus using the mailbox
                                       // slave=1
```

7.2 Siemens

Die Daemonen für Siemens S7 und S5 SPS sind analog zu den Modbus Daemonen aufgebaut. Lesen Sie daher zunächst den Abschnitt über Modbus.

7.2.1 Zugriff über TCP mit lesbaren ASCII Zeichen als Kodierung

Im Verzeichnis pvbaddon/daemons/siemenstcp/client in pvbaddon findet man den Siemens Daemon für TCP Verbindungen.
Hier kommt die Klasse rlSiemensTCP aus der rllib zum Einsatz.
Die INI Datei ist ähnlich wie die Modbus INI Datei aufgebaut. Es können aber beliebig viele SPS Verbindungen mit einem Daemon realisiert werden. HAVETO_SWAP sollte bei einer X86 CPU auf 1 gesetzt sein. Wenn der Prozessor eine andere Byte Order hat, kann man den Wert auf 0 setzen. Bei den SLAVE Werten gibt man zunächst die IP Adresse oder den Namen für die SPS an. Dann folgt ein Kommata und der SPS-Typ. Als optionalen Parameter kann noch auswählen, ob das Fetch/Write Protokoll oder das neuere Siemens TCP Protokoll verwendet werden soll. In der INI Datei ist NUM_CYCLES=1 gesetzt, weil zu Testzwecken nur der erste Zyklus verwendet werden soll. Wenn der zweite Zyklus ebenfalls durchgeführt werden soll, muss NUM_CYCLES=2 gesetzt werden.

INI Datei für Siemens SPS über TCP

```
# ini file for siemenstcp_client
#
# DEBUG      := 1 | 0
# SLAVE<N>   := IP,PLC_TYPE,FETCH_WRITE,FUNCTION,RACK_SLOT
# PLC_TYPE   := ANY | S7_200 | S7_300 | S7_400 | S5 | S7_1200 | LOGO
# FETCH_WRITE := 1 | 0 # default 1
# FUNCTION   := optional parameter for PLC (1=PG,2=OP,3=Step7Basic)
# RACK_SLOT  := optional parameter for PLC Byte(upper_3_bit_is_rack / lower_5_bit_is_slot)
# CYCLE<N>   := <count>,<name>
# name       := byte<ORG>(slave,dbnum,adr) |
#               float<ORG>(slave,dbnum,adr) |
#               dword<ORG>(slave,dbnum,adr) |
#               short<ORG>(slave,dbnum,adr) |
#               udword<ORG>(slave,dbnum,adr) |
#               ushort<ORG>(slave,dbnum,adr)
# ORG        := ORG_DB | ORG_M | ORG_E | ORG_A | ORG_PEPA | ORG_Z | ORG_T
# HAVETO_SWAP := 1 | 0 # must be 1 on intel machines
# CYCLETIME in milliseconds
# SHARED_MEMORY_SIZE must be equal to SHARED_MEMORY_SIZE of pvserver
# MAX_NAME_LENGTH is maximum length of variable name in shared memory
#

[GLOBAL]
DEBUG=1
CYCLETIME=1000
HAVETO_SWAP=1

[SOCKET]
NUM_SLAVES=1
SLAVE1=192.168.1.101,ANY,0
#SLAVE2=192.168.1.35,S7_200,0,1,2

# You may also specify the TSAPs explicitly.
# In that case the PLC_TYPE does not care. Use ANY.
[SLAVE1_CONNECT_BLOCK]
#S7-200
CB13='M' # remote TSAP     (not necessary to set explicitly)
CB14='W' # remote TSAP     (not necessary to set explicitly)
CB17='M' # local TSAP PG   (1=PG,2=OP,3=Step7Basic)
CB18='W' # local TSAP slot 1 (upper_3_bit_is_rack / lower_5_bit_is_slot)
#S7-300
#CB13=2 # remote TSAP     (not necessary to set explicitly)
#CB14=1 # remote TSAP     (not necessary to set explicitly)
#CB17=1 # local TSAP PG   (1=PG,2=OP,3=Step7Basic)
#CB18=2 # local TSAP slot 2 (upper_3_bit_is_rack / lower_5_bit_is_slot)
#S7-400
#CB13=2 # remote TSAP     (not necessary to set explicitly)
#CB14=1 # remote TSAP     (not necessary to set explicitly)
#CB17=1 # local TSAP PG   (1=PG,2=OP,3=Step7Basic)
#CB18=3 # local TSAP slot 3 (upper_3_bit_is_rack / lower_5_bit_is_slot)
#S7-1200
#CB13=2 # remote TSAP     (not necessary to set explicitly)
#CB14=1 # remote TSAP     (not necessary to set explicitly)
#CB17=1 # local TSAP PG   (1=PG,2=OP,3=Step7Basic)
#CB18=0 # local TSAP slot 0 (upper_3_bit_is_rack / lower_5_bit_is_slot)

[RLLIB]
MAX_NAME_LENGTH=30
SHARED_MEMORY=/srv/automation/shm/siemenstcp1.shm
SHARED_MEMORY_SIZE=65536
MAILBOX=/srv/automation/mbx/siemenstcp1.mbx

[CYCLES]
```

```
NUM_CYCLES=4
CYCLE1=10,byteORG_M(1,0,0)
CYCLE2=4,byteORG_E(1,0,0)
CYCLE3=4,byteORG_A(1,0,0)
CYCLE4=4,byteORG_DB(1,1,0)
#CYCLE2=1,byteORG_M(2,2,3)
```

7.2.2 Zugriff über PPI mit lesbaren ASCII Zeichen als Kodierung

Im Verzeichnis pvbaddon/daemons/siemensppi/client in pvbaddon findet man den Siemens Daemon für PPI Verbindungen (serielle Schnittstelle).
Als Kommunikationsbibliothek kommt bei PPI libnodave `http://libnodave.sourceforge.net/` zum Einsatz.
Eine Kopie von Libnodave wird mit unserem Paket mitgeliefert.
Die INI Datei ist ähnlich wie die Modbus INI Datei aufgebaut.

INI Datei für Siemens SPS über PPI

```
# ini file for siemensppi_client
#
# DEBUG     := 1 | 0
# BAUDRATE  := 300   |
#              600   |
#              1200  |
#              1800  |
#              2400  |
#              4800  |
#              9600  |
#              19200 |
#              38400 |
#              57600 |
#              115200
# CYCLE<N>  := <count>,<name>
# name      := sd(slave,dbnum,start_adr)   |
#              inputs(slave,dbnum,start_adr) |
#              outputs(slave,dbnum,start_adr) |
#              flags(slave,dbnum,start_adr) |
#              db(slave,dbnum,start_adr)   |
#              di(slave,dbnum,start_adr)   |
#              local(slave,dbnum,start_adr) |
#              v(slave,dbnum,start_adr)    |
#              counter(slave,dbnum,start_adr) |
#              timer(slave,dbnum,start_adr)
# CYCLETIME in milliseconds
# SHARED_MEMORY_SIZE must be equal to SHARED_MEMORY_SIZE of pvserver
# MAX_NAME_LENGTH is maximum length of variable name in shared memory
#

[GLOBAL]
DEBUG=1
DAVE_DEBUG=0
CYCLETIME=1000

[TTY]
DEVICENAME=/dev/ttyUSB0
BAUDRATE=9600

[RLLIB]
MAX_NAME_LENGTH=30
SHARED_MEMORY=/srv/automation/shm/siemensppi1.shm
SHARED_MEMORY_SIZE=65536
MAILBOX=/srv/automation/mbx/siemensppi1.mbx

[CYCLES]
```

```
NUM_CYCLES=2
CYCLE1=64,db(2,1,0)
CYCLE2=1,db(2,1,10)
```

7.2.3 Aus pvdevelop generierte Daemonen für Siemens TCP und PPI

Genau wie bei Modbus gibt für Siemens SPS auch die Möglichkeit einen Daemon mit pvdevelop zu generieren. Die generierte Datei wird als siemensdaemon.cpp in das aktuelle Verzeichnis geschrieben und compiliert.

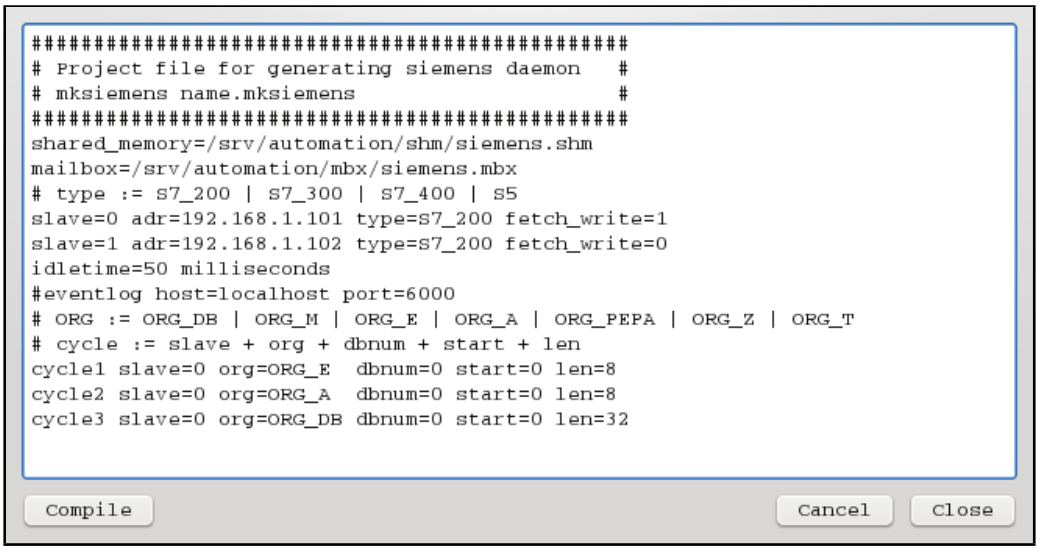

Abbildung 7.3: Dialog zum Generieren eines Siemens TCP daemons in pvdevelop

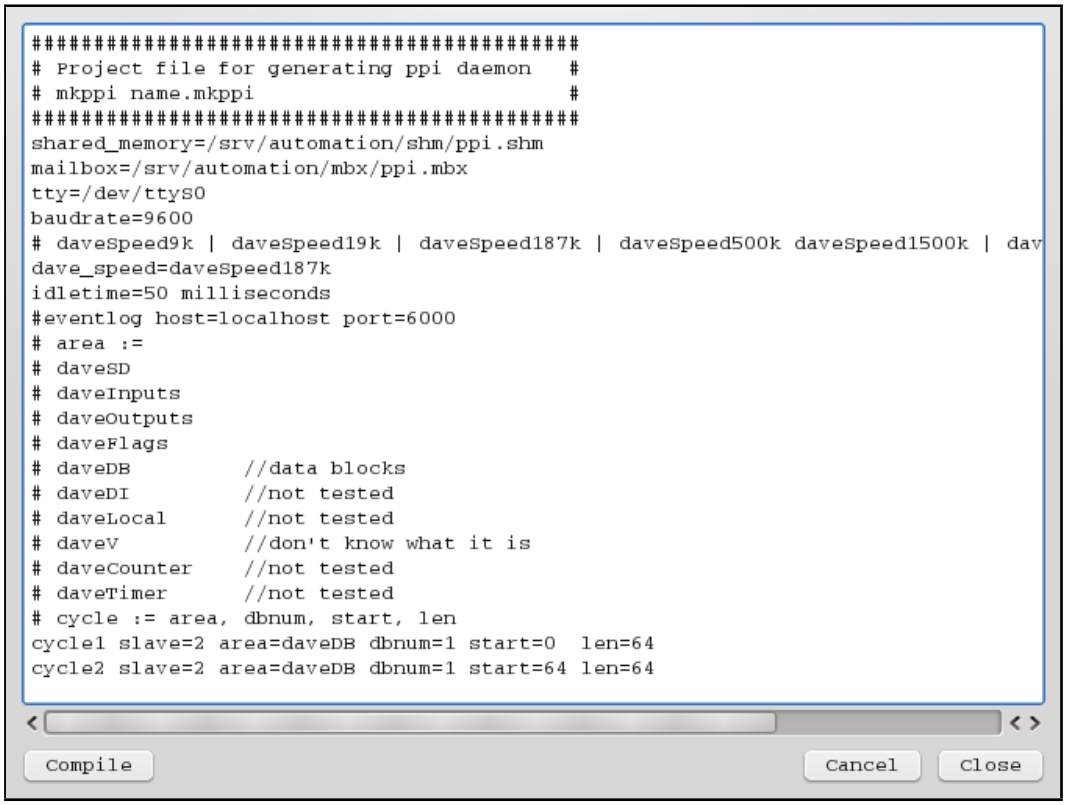

Abbildung 7.4: Dialog zum Generieren eines Siemens PPI daemons in pvdevelop

7.3 EIB Bus

Im Verzeichnis pvbaddon/daemons/eibnet/client in pvbaddon findet man den EIBnet Daemon für Verbindungen zum europäischen Installations Bus über ein TCP/EIB Gateway mit EIBnet.
Hier kommt die Klasse rlEIBnetIP als der rllib zum Einsatz.
Die INI Datei ist ähnlich wie die Modbus INI Datei aufgebaut. Es muss die IP Adresse des Gateway und die Adresse Ihres Computers angegeben werden.
Die EIB Bus Variablen müssen nicht explizit spezifiziert werden, weil der Daemon alle auf dem Bus übertragenen Daten in das Shared Memory einträgt.

INI Datei für EIBnet

```
# ini file for eibnet_client (EIBnet/KNX)
#
# DEBUG    := 1 | 0
# DEBUG_EIB := 1 | 0
# WATCH_EIB := 1 | 0
# SHARED_MEMORY_SIZE must be equal to SHARED_MEMORY_SIZE of pvserver
# MAX_NAME_LENGTH is maximum length of variable name in shared memory
#

[GLOBAL]
DEBUG=1
DEBUG_EIB=1
WATCH_EIB=1

[SOCKET]
GATEWAY_IP=192.168.1.102
CLIENT_IP=192.168.1.14
#CLIENT_IP=192.168.1.129

[RLLIB]
MAX_NAME_LENGTH=12
SHARED_MEMORY=/srv/automation/shm/eibnet1.shm
SHARED_MEMORY_SIZE=65536
MAILBOX=/srv/automation/mbx/eibnet1.mbx
```

7.4 Ethernet/IP

Im Verzeichnis pvbaddon/daemons/ethernetip/client in pvbaddon findet man den Daemon der das von Allen Bradley und Rockwell verwendete Ethernet/IP Protokoll implementiert.
Hier kommt das Open Source Projekt TuxEip zum Einsatz, welches leider nicht mehr gepflegt wird.
Die INI Datei ist ähnlich wie die Modbus INI Datei aufgebaut.

INI Datei für Ethernet/IP

```
# ini file for ethernetip_client
#
# PLC_TYPE := PLC5 | SLC500 | LGX
# CHANNEL := Channel_A | Channel_B
# CYCLE<N> := <count>,<name>
# CYCLETIME in milliseconds
# SHARED_MEMORY_SIZE must be equal to SHARED_MEMORY_SIZE of pvserver
# MAX_NAME_LENGTH is maximum length of variable name in shared memory
#

[GLOBAL]
USE_CONNECT_OVER_CNET=1
TNS=1234
DEBUG=1
CYCLETIME=1000
IP=192.168.1.115
```

```
[ConnectPLCOverCNET]
PLC_TYPE=SLC500
CONNECTION_ID=0x12345678
CONNECTION_SERIAL_NUMBER=0x6789
REQUEST_PACKET_INTERVAL=5000
PATH=1,0

[ConnectPLCOverDHP]
PLC_TYPE=PLC5
TARGET_TO_ORIGINATOR_ID=0x12345678
CONNECTION_SERIAL_NUMBER=0x6789
CHANNEL=Channel_B
PATH=1,1,2,2,1,3

[RLLIB]
MAX_NAME_LENGTH=8
SHARED_MEMORY=/srv/automation/shm/ethernetip1.shm
SHARED_MEMORY_SIZE=65536
MAILBOX=/srv/automation/mbx/ethernetip1.mbx

[CYCLES]
NUM_CYCLES=2
CYCLE1=8,H7:0
CYCLE2=8,H7:2
```

7.5 Profibus und CAN

Für Profibus und CAN bevorzugen wir die CIF Karten von Hilscher `http://hilscher.com`.
Diese Karten besitzen einen eigenen Controller, der das Feldbus Protokoll abwickelt. Die Karten speichern die Prozessvariablen in einen Dual Ported Memory ab, auf das der Controller und der PC zugreifen kann. Mit dem Konfigurationswerkzeug SyCon von Hilscher wird definiert, an welcher Stelle welche Prozessvariable liegen soll. Die neuen auf dem netX basierenden Karten arbeiten nach dem gleichen Verfahren, unterstützen aber praktisch alle geläufigen Feldbusprotokolle. Der darauf integrierte ARM Prozessor wird dazu mit der entsprechenden Firmware geladen.
In pvbrowser stellt die Klasse rlHilscherCIF einen Wrapper für den Hilscher Treiber dar. Da diese Karten die Prozessvariablen bereits in dem Dual Ported Memory speichern, verzichten wir auf unser Shared Memory und die Mailbox. Statt dessen starten wir in unserem pvserver einen zusätzlichen Thread, in dem die Karte ausgelesen werden kann und Variablen auf dem Feldbus gesendet werden können.
In dem Verzeichnis pvbaddon/demos/hilschercif von pvbaddon finden Sie ein Beispiel.
Die als global definierten Variablen sendData und receiveData können dann von der Visualisierung in den Masken verwendet werden. Mit pbus.lock() und pbus.unlock() wird der konkurrierende Zugriff auf die Variablen auch für mehrere Clients synchronisiert.

Separater Thread für Hilscher Karten

```
#include "rlhilschercif.h"
rlHilscherCIF cif;
unsigned char sendData[512];
unsigned char receiveData[512];
rlThread pbus;

void *profibus(void *arg)
{
#ifdef _RL_HILSCHER_CIF_H_
  THREAD_PARAM *p = (THREAD_PARAM *) arg;
  cif.debug = 1;
  if(cif.open() == DRV_NO_ERROR)
  {
    cif.debug = 0;
    while(p->running)
```

```
    {
      rlsleep(50);
      pbus.lock();
      cif.devExchangeIO(0,4,sendData,
                        0,4,receiveData,
                        1000);
      pbus.unlock();
    }
  }
  else
  {
    printf("failed␣to␣cif.open()\n");
    printf("Please␣run␣me␣as␣root␣or\n");
    printf("make␣/dev/cif␣readable␣by␣normal␣user\n");
  }
#else
  printf("WARNING:␣you␣will␣have␣to␣install␣the␣hilscher␣driver␣and␣link␣to␣it.␣Then␣you␣can␣remove␣
      the␣ifdef␣_WIN32\n");
#endif
  return arg;
}

// snip

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 */
  memset(sendData,0,sizeof(sendData));
  memset(receiveData,0,sizeof(receiveData));
  pbus.create(profibus,NULL);
  while(1)
  {
    s = pvAccept(&p);
    if(s != -1) pvCreateThread(&p,s);
    else       break;
  }
  return 0;
}
```

7.6 OPC XML-DA

OPC XML-DA ist die erste plattformunabhängige Variante von OPC. Im Gegensatz zu dem klassischen auf COM/DCOM von Microsoft basierenden OPC läuft OPC XML-DA über http Anfragen mit XML, die nicht nur unter Windows funktionieren.
In dem Verzeichnis pvbaddon/daemons/opcxmlda/client von pvbaddon finden Sie unseren Daemon, der einen OPC XML-DA Client implementiert
Mit diesem Client lesen Sie zunächst das Objektverzeichnis Ihres OPC XML-DA Servers aus.

Auslesen eines OPC XML-DA Objektverzeichnisses

```
./opcxmlda_client http://server/opcxmlda/xmldaserver Browse > opcxmlda.itemlist
```

Beispiel für den Inhalt einer opcxmlda.itemlist

```
#./opcxmlda_client http://192.168.208.128/opcxmlda/isopc.simopcserver.3 Browse
#
#
#Level1
```

```
Level1/DS_Devicename
Level1/DS_DeviceID
Level1/DeviceType
Level1/DS_Vendorname
Level1/ProfileID
Level1/SW_Rev
Level1/HW_Rev
Level1/Ser_Num
Level1/Descriptor
Level1/Dev_Instal_Date
Level1/Dev_Message
Level1/Out
Level1/Hi_Lim
Level1/Lo_LIM
#
#Level2
Level2/DS_Devicename
Level2/DS_DeviceID
Level2/DeviceType
Level2/DS_Vendorname
Level2/ProfileID
Level2/SW_Rev
Level2/HW_Rev
Level2/Ser_Num
Level2/Descriptor
Level2/Dev_Instal_Date
Level2/Dev_Message
Level2/Out
Level2/Target
#
#Pump1
Pump1/DS_Devicename
Pump1/DS_DeviceID
Pump1/DeviceType
Pump1/DS_Vendorname
Pump1/ProfileID
Pump1/SW_Rev
Pump1/HW_Rev
Pump1/Ser_Num
Pump1/Descriptor
Pump1/Dev_Instal_Date
Pump1/Dev_Message
Pump1/ThroughPut
Pump1/Revolutions
Pump1/Capacity
Pump1/Gain
#
#Pump2
Pump2/DS_Devicename
Pump2/DS_DeviceID
Pump2/DeviceType
Pump2/DS_Vendorname
Pump2/ProfileID
Pump2/SW_Rev
Pump2/HW_Rev
Pump2/Ser_Num
Pump2/Descriptor
Pump2/Dev_Instal_Date
Pump2/Dev_Message
Pump2/ThroughPut
Pump2/Revolutions
Pump2/Capacity
Pump2/Gain
```

```
#
#Command
Command/Enter
#
#test
test/Int16
test/Int32
test/float
test/double
test/string
```

In dieser itemlist können Sie die Variablen auskommentieren, die Sie nicht interessieren.

Benutzung von opcxmlda_client

```
user@host:~/pvbaddon/daemons/opcxmlda/client> ./opcxmlda_client
Usage: ./opcxmlda_client [URL] [METHOD] <-itemlist=filename> <-shm=filename> <-mbx=filename> <-sleep=
    milliseconds> <-max_name_length=char> <-shmsize=bytes> <-debug>

[URL] is the url of the OPC XML-DA server.
[METHOD] is the method to call. [METHOD] := GetStatus | Browse | Run
[URL] and [METHOD] are mandatory and must be the first 2 parameters.

Defaults:
-itemlist=opcxmlda.itemlist # may be created by Browse
-shm=/srv/automation/shm/opcxmlda.shm OR
    c:\automation\shm\opcxmlda.shm on windows
    # location of the shared memory
-mbx=/srv/automation/mbx/opcxmlda.mbx OR
    c:\automation\mbx\opcxmlda.mbx on windows
    # location of the mailbox
-sleep=1000        # time between read calls
-max_name_length=31 # max length of result name
-shmsize=65536     # total size of the shared memory

Example for creating opcxmlda.itemlist:
./opcxmlda_client http://server/opcxmlda/xmldaserver Browse > opcxmlda.itemlist
```

Wenn die angegebenen Default Werte ausreichen, könnte der Aufruf des opcxmlda_client folgendermaßen aussehen.

Beispielhafter Aufruf von opcxmlda_client

```
./opcxmlda_client http://192.168.1.13/opcxmlda/isopc.simopcserver.3 Run
```

In Ihrem pvserver verwenden Sie bei OPC XML-DA die Klasse rlOpcXmlDa aus der rllib. Ein Beispiel für einen solchen pvserver ist im Verzeichnis pvbaddon/daemons/opcxmlda/pvs von pvbaddon zu finden.

7.7 Benutzung von Gateways

Falls ein Feldbusprotokoll in pvbrowser integriert werden soll, das von hause aus nicht unterstützt wird, könnte es über entsprechende Gateways möglich sein. Hier ist es am wahrscheinlichsten auf der PC Seite Modbus zu verwenden.
Einige Beispiele:
Modbus LON Gateway
http://www.intellicom.se/lonworks_modbus_rtu.cfm
Profibus Modbus-TCP Gateway der Firma Comsoft
http://www.directindustry.de/prod/comsoft/ethernet-profibus-feldbus-gateway-36838-347665.html
http://www.directindustry.com/prod/comsoft/ethernet-profibus-fieldbus-gateway-36838-347665.html
CAN Modbus Gateway
http://www.adfweb.com/home/products/CAN_Modbus_RTU.asp?frompg=nav3_2
http://www.wachendorff.de/wp/Protokollwandler-Gateway-CAN-zu-Modbus-HD67014.html

7.8 Vorlage für weitere Protokolle

Der Modbus Daemon im Verzeichnis pvbaddon/daemons/modbus/client von pvbaddon kann als Vorlage für Ihre selbst geschriebene Datenerfassung dienen.
Einfacher ist es Ihre Datenerfassung aus dem von pvdevelop generierten modbusdaemon.cpp abzuwandeln.
Falls Sie auf das Shared Memory und die Mailbox verzichten wollen, können Sie auch einen separaten Thread in Ihrem pvserver verwenden, um die Prozessdaten zu erfassen. Sehen Sie sich dazu an, wie das unter 'Profibus und CAN' gemacht wird.

7.9 Vorlage für Arduino Integration über eine serielle USB Schnittstelle

Die Arduino Mikrocontroller Boards besitzen eine serielle USB Schnittstelle (tty), die benutzt werden kann, um mit einem Host Rechner zu kommunizieren. Unter Linux wird die Schnittstelle als /dev/ttyUSBx repräsentiert. In main.cpp eines pvserver kann man einen zusätzlichen Thread starten, der die Kommunikation mit dem Arduino Board übernimmt (über tty). Dort könnte man die Texte interpretieren, die von Arduino mit print Aufrufen gesendet werden (über die tty). Ebenso könnte man einen rlFifo verwenden, um mit slotFunktionen() zu kommunizieren, die die Clienten bedienen (anderer Thread). Über den rlFifo können die slotFunktionen() dann Kommandos senden, die zum Arduino-Kommunikations-Thread gesendet werden.

Zusätzlicher Thread, der mit dem Arduino Board über USB kommuniziert

```
// include this on top of main.cpp within your pvserver
rlThread arduinoThread;
rlFifo  arduinoFifo;
void   *arduinoFunc(void *arg)
{
  int ret;
  char line[128],buf[128];
  THREAD_PARAM *p = (THREAD_PARAM *) arg;
  if(p == NULL) return NULL;

  while(1) // forever
  {
    rlSerial tty;
    // ret = tty.openDevice("/dev/ttyUSB0",B9600,1,0,8,1,rlSerial::NONE); // because /dev/ttyUSBx
        will change, we use the tty /dev/serial/by-id
    ret = tty.openDevice("/dev/serial/by-id/usb-FTDI_FT232R_USB_UART_A104WLOP-if00-port0",B9600
        ,1,0,8,1,rlSerial::NONE); // openDevice without hardware control
    if(trace) printf("tty.openDevice␣ret=%d\n", ret);
    if(ret >= 0)
    {
      while(1) // while tty is plugged in
      {
        if(arduinoFifo.poll() == rlFifo::DATA_AVAILABLE)
        { // if we got something to send to arduino
          ret = arduinoFifo.read(line, sizeof(line)-1);
          if(trace) printf("arduinoFifo.read:%s\n", line);
          // TODO:        // send what arduino can interpret
          line[1] = 0;    // currently we send only 1 char to arduino
          ret = tty.writeBlock((const unsigned char*) line, strlen(line));
          if(ret < 0) break;
        }
        ret = tty.select(); // test if ardunio send something to the tty
        if(trace) printf("tty.select:␣ret=%d\n", ret);
        if(ret > 0)       // if data is available on the tty
        {
          // arduino will send lines terminated with 0x0d 0x0a
          ret = tty.readLine((unsigned char *) line,sizeof(line)-1);
          if(trace) printf("tty.readLine:␣␣ret=%d␣line=%s\n", ret, line); // should be line + 0x0d
          if(ret < 0) break;
          ret = tty.readBlock((unsigned char *) buf,1,1000);
```

```
        if(trace) printf("tty.readBlock: ret=%d char=%x\n", ret, buf[0]); // should be 0x0a
        if(ret < 0) break;
        // TODO:        // interpret the line that was send from arduino
      }
    }
  }
  if(trace) printf("WARNING: tty unplugged\n");
  tty.closeDevice();
  rlsleep(10000);
 }
 return NULL;
}
```

Start des zusätzlichen Thread in main()

```
// include this in main.cpp
int main(int ac, char **av)
{
PARAM p;
int   s;

 pvInit(ac,av,&p);
 arduinoThread.create(arduinoFunc,NULL);
 // snip ...
```

includes in pvapp.h

```
// include this in pvapp.h
#include "rlthread.h"
#include "rlserial.h"
#include "rlfifo.h"
```

slotFunktion() sendet etwas über den rlFifo

```
// include this in maskX_slots.h
extern rlFifo arduinoFifo;

// snip ...

static int slotButtonEvent(PARAM *p, int id, DATA *d)
{
 if(p == NULL || id == 0 || d == NULL) return -1;
 if(id == pbRequest)
 {
   arduinoFifo.printf("%d\n", d->i++);
   if(d->i > 9) d->i = 0;
 }
 return 0;
}

// snip ...
```

Kapitel 8

Distributed Control System (DCS)

pvbrowser kann um die Funktion eines DCS erweitert werden. Über unsere Bibliotheken können wir bereits Daten von SPS Systemen, entfernten IO Systemen über Feldbusse oder direkt im Computer vorhandenen IO erfassen und ausgeben. Diese Daten werden in pvbrowser in einem 'Shared Memory' gespeichert und können von der Visualisierung angezeigt werden bzw. von der Visualisierung beeinflusst werden.
Es gibt nun keinen Grund, warum neben der Visualisierung nicht auch ein DCS System in einem eigenen Prozess implementiert werden sollte. Sowohl Visualisierung als auch DCS System können gleichzeitig auf das 'Shared Memory' zugreifen.
Prinzipiell ist es möglich beliebig viele Prozesse oder Threads parallel auf das 'Shared Memory' zugreifen zu lassen. Aus praktischen Gründen sollte man aber nicht zu viele separate Prozesse verwenden.
Es erscheint beispielsweise sinnvoll die Datenerfassung und das DCS in einem Prozess zusammenzufassen.
Auf das 'Shared Memory' kann man mit read/write Operationen zugreifen, die den Zugriff über einem Mutex gegenseitig verriegeln. Es ist aber auch möglich die Basisadresse des 'Shared Memory' zu bekommen und dann selbst für die Synchronisierung zu sorgen.
Ein sinnvoller Ansatz wäre hier eine komplexe von Benutzer festgelegte Datenstruktur in dem 'Shared Memory' abzulegen, die alle notwendigen Informationen über Ein-/Ausgänge und berechnete Größen enthält.

Vom Benutzer festgelegte Datenstruktur

```
typedef struct
{ ... }
BENUTZER_DEFINIERTE_DATENSTRUKTUR;
```

Da die einzelnen Werte in der Datenstruktur aus Basisdatentypen bestehen (int/float) werden diese in atomaren Schreib-/Leseoperationen zum 'Shared Memory' transferiert. Es ist also unter bestimmten Randbedingungen möglich auf Verriegelungen des 'Shared Memory' zu verzichten, um so die Parallelität der Gesamtlösung zu verbessern. Die Gesamtlösung wird durch Prozessoren mit mehreren Kernen parallelisiert. Es wird also nicht nur Timesharing verwendet.
Ein DCS kann nun in einem eigenen Prozess implementiert werden und besteht aus mehreren parallel laufenden Threads.

- *Ein Thread mit einer Schleife zur Datenerfassung*
- *Ein Thread mit einer Schleife für das Logging in einer Datenbank*
- *Mehrere Zustandsmaschinen, die jeweils in einem eigenen Thread laufen*
- *Kontinuierliche Regler (P/PI/PID ...) die jeweils in einem eigenen Thread laufen*

In unserer rllib sind alle notwendigen Klassen vorhanden, um einen solchen Prozess mit wenig Aufwand zu erstellen. Der Benutzer kann dabei auf Vorlagen (Templates) zurückgreifen, die er als Ausgangspunkt für seine eigene Lösung verwenden kann. Der Programmieraufwand wird so stark verringert und der Benutzer muss nur noch die eigene Logik eingeben. Das Rahmenprogramm kann einfach kopiert werden.
Eine solche Vorlage für Datenerfassung mit Modbus + Zustandsautomaten ist beispielsweise unter 'pvbaddon/templates/statemachine' zu finden.

8.1 DCS Vorlage mit Datenerfassung und Zustandsautomaten

Die Header Datei 'plcapp.h' inkludiert einige weitere Header und definiert eine benutzerdefinierte Datenstruktur. In 'main()' erfolgt die Initialisierung, bei der die benutzerdefinierte Datenstruktur in das 'Shared Memory' gelegt wird.

Benutzer definierte Datenstruktur im Shared Memory

```
BENUTZER_DEFINIERTE_DATENSTRUKTUR *bs = (BENUTZER_DEFINIERTE_DATENSTRUKTUR *) shared_memory.getUsrAdr
    ();
```

Dann werden die Threads für die Zustandsautomaten gestartet. Nach der Initialisierung geht 'main()' in eine Endlosschleife, liest die Eingänge und schreibt die Ausgänge zyklisch. Dabei ist auf die korrekte Abtastrate zu achten. Die analogen Eingänge müssen bandbegrenzt sein und die Abtastrate muss hoch genug sein, um die analogen Eingänge mindestens 2 mal in der Periodendauer der Grenzfrequenz abzutasten. Die Zyklusdauer der Datenerfassung sollte höchstens halb so groß sein, wie die Zyklusdauer, mit der die Zustandsmaschinen arbeiten.
Sehen Sie hier den entsprechenden Quelltext:

main.cpp

```
//***************************************************************************
//                        main.cpp -  description
//                           -------------------
// begin          : Sa. Mai 4 09:29:07 2013
// generated by   : pvdevelop (C) Lehrig Software Engineering
// email          : lehrig@t-online.de
//***************************************************************************
#include "plcapp.h"

SHM_DATA     *shm_data;
rlSharedMemory shm("/srv/automation/shm/plc.shm", sizeof(SHM_DATA));
rlSerial      tty;
rlModbus      mb;
rlMutex       mb_mutex;
rlState       sm1, sm2;

// helper functions
int printBinByte(unsigned char val)
{
  if(val & BIT7) printf("1");
  else           printf("0");
  if(val & BIT6) printf("1");
  else           printf("0");
  if(val & BIT5) printf("1");
  else           printf("0");
  if(val & BIT4) printf("1");
  else           printf("0");
  printf(":");
  if(val & BIT3) printf("1");
  else           printf("0");
  if(val & BIT2) printf("1");
  else           printf("0");
  if(val & BIT1) printf("1");
  else           printf("0");
  if(val & BIT0) printf("1");
  else           printf("0");
  return 0;
}

int printBin(unsigned char *data)
{
  printf("BinData:␣");
  printBinByte(data[0]);
```

```
  printf("␣-␣");
  printBinByte(data[1]);
  return 0;
}

// Schneider PLC: first 4 bits are outputs then 6 bits input follow
static int readIO()
{
  unsigned char data[256];
  int ret;

  MB_readInputStatus(1,0,10,data);      // read all IO values from modbus
  shm_data->plc.in.in1 = mb.data2int(data); // store data in shared memory

  if(trace)
  {
    printf("readIO::␣ret=%d␣", ret);
    printBin(data);
    printf("␣in1=%x\n", shm_data->plc.in.in1);
  }
  return 0;
}

static int writeIO()
{
  unsigned char coils[8];
  int ret;

  coils[0] = shm_data->plc.out.out1 & 0x0ff;
  MB_forceMultipleCoils(1,0,4,coils);   // write the 4 output bits to modbus

  return 0;
}

int main()
{
  if(trace) printf("plc␣starting␣...\n");
  if(trace) printf("shm.status=%d\n", shm.status);

  // --- Initialize our DCS ---
  if(shm.status != rlSharedMemory::OK)
  {
    printf("ERROR:␣shared␣memory␣status␣is␣not␣ok\n");
    return -1;
  }
  shm_data = (SHM_DATA *) shm.getUserAdr();
  memset(shm_data,0,sizeof(SHM_DATA));
retry:
  if(tty.openDevice("/dev/ttyUSB0",B9600,1,1,8,1,rlSerial::NONE) < 0)
  {
    printf("ERROR:␣openDevice(\"/dev/tty/USB0\")\n");
    rlsleep(5000);
    goto retry;
  }
  mb.registerSerial(&tty);

  // --- Start our treads ---
  startStepsStm1(&sm1, 100); // start statemachine 1
  startStepsStm2(&sm2, 100); // start statemachine 2
  // TODO: eventually start a thread for logging data (into a database)
  // TODO: eventually start threads for continuous loop back controller with rlController from rllib

  // --- Continuous loop for data acquisition ---
```

```
  printf("going to IO loop\n");
  while(1)
  {
    readIO();
    writeIO();
    rlsleep(10);
  }
}
```

Statemachine 1

```
//*************************************************************************
//                        stm1.cpp -  description
//                           -------------------
//  begin          : Sa. Mai 4 09:29:07 2013
//  generated by   : pvdevelop (C) Lehrig Software Engineering
//  email          : lehrig@t-online.de
//*************************************************************************
#include "plcapp.h"
extern rlState sm2;

//TODO: define our states
static void stStart(rlState *sm);

//TODO: implement our states
static void stStart(rlState *sm)
{
  shm_data->plc.out.out2 = sm->stepCounter;
  if(shm_data->plc.state.stm2_running == 0)
  {
    if(shm_data->pvs.state.button_start_stm2 == 1)
    {
      startStepsStm2(&sm2, 100);           // start statemachine 2 thread
    }
    else if(shm_data->plc.in.in1 & BIT1)
    {
      startStepsStm2(&sm2, 100);           // start statemachine 2 thread
    }
  }
}

int startStepsStm1(rlState *sm, int cycletime) // start our statemachine
{
  if(trace) printf("Start stm1\n");
  shm_data->plc.state.stm1_running = 1;    // set running within shared memory
  sm->gotoState(stStart);                  // goto nextState
  sm->startSteps(cycletime);               // start a thread that will handle our statemachine
  return 0;
}
```

Statemachine 2

```
//*************************************************************************
//                        stm2.cpp -  description
//                           -------------------
//  begin          : Sa. Mai 4 09:29:07 2013
//  generated by   : pvdevelop (C) Lehrig Software Engineering
//  email          : lehrig@t-online.de
//                   A simple template for implementing your own statemachine
//                   See: pvbaddon/templates/statemachine
//*************************************************************************
#include "plcapp.h"
```

```
//TODO: define our states
//     Your states are defined by static functions which get a pointer to the statemachine
//     The pointer sm->user might be used to transfer the address of a user defined datastructure
//     A transition from one state to the next is done by sm->gotoState(theNextState);
//     Your statemachine runs within a separate thread and the current state is called within "
    cycletime" intervals
static void stStart(rlState *sm);
static void stProcess(rlState *sm);
static void stFinish(rlState *sm);

//TODO: implement our states
static void stStart(rlState *sm)
{
  shm_data->plc.out.out1 = 1;                  // set output 1 in shared memory
  if(sm->stepCounter > 20)
  {
    shm_data->plc.out.out1 = 2;                // reset output 1 in shared memory
    strcpy(shm_data->plc.state.stm2_name,"Process"); // set next state name in shared memory
    sm->gotoState(stProcess);                  // goto the next state
  }
}

static void stProcess(rlState *sm)
{
  shm_data->plc.out.out1 = sm->stepCounter;    // set output 1 in shared memory
  if(sm->stepCounter > 30)
  {
    strcpy(shm_data->plc.state.stm2_name,"Finish"); // set next state name in shared memory
    sm->gotoState(stFinish);                   // goto the next state
  }
}

static void stFinish(rlState *sm)
{
  shm_data->plc.out.out1 = 1;                  // set output 1 in shared memory
  if(sm->stepCounter > 30)
  {
    shm_data->plc.out.out1 = 0;                // reset output 1 in shared memory
    strcpy(shm_data->plc.state.stm2_name,"NULL"); // set next state name NULL
    shm_data->plc.state.stm2_running = 0;      // reset running in shared memory
    sm->gotoState(NULL);                       // goto NULL state
  }
}

int startStepsStm2(rlState *sm, int cycletime)  // start our statemachine
{
  if(trace) printf("stm2␣starting\n");
  shm_data->plc.state.stm2_running = 1;        // set running in shared memory
  strcpy(shm_data->plc.state.stm2_name,"Start"); // set next state name in shared memory
  sm->gotoState(stStart);                      // goto nextState
  sm->startSteps(cycletime);                   // start a thread which handles the statemachine
  return 0;
}
```

Header plcapp.h

```
//***************************************************************************
//                      plcapp.h -  description
//                         -------------------
//  begin          : Sa. Mai 4 09:29:07 2013
//  generated by   : pvdevelop (C) Lehrig Software Engineering
//  email          : lehrig@t-online.de
//***************************************************************************
#ifndef _PVAPP_PLC_H_
```

```
#define _PVAPP_PLC_H_

static int trace=1; // todo: set trace=0 if you do not want printf() within event loop

#include <string.h>
#include "rlthread.h"
#include "rlstring.h"
#include "rlmodbus.h"
#include "rlsharedmemory.h"
#include "rlcutil.h"
#include "rlstate.h"

// define the global macros and variables that will be used in readIO() and writeIO()

#define MB_forceMultipleCoils(slave,start_adr,number,data) \
  mb_mutex.lock();                                         \
  rlsleep(10);                                             \
  ret = mb.forceMultipleCoils(slave,start_adr,number,data); \
  mb_mutex.unlock();                                       \
  if(ret < 0)                                              \
  {                                                        \
    printf("WARNING: writeIO Modbus does not response\n"); \
  }

#define MB_readInputStatus(slave,start_adr,number,data)   \
  mb_mutex.lock();                                        \
  rlsleep(10);                                            \
  ret = mb.readInputStatus(slave,start_adr,number,data);  \
  mb_mutex.unlock();                                      \
  if(ret <= 0)                                            \
  {                                                       \
    printf("WARNING: readIO Modbus does not response\n"); \
  }

#define MB_readCoilStatus(slave,start_adr,number,data)    \
  mb_mutex.lock();                                        \
  rlsleep(10);                                            \
  ret = mb.readCoilStatus(slave,start_adr,number,data);   \
  mb_mutex.unlock();                                      \
  if(ret <= 0)                                            \
  {                                                       \
    printf("WARNING: readIO Modbus does not response\n"); \
  }

//int ret = mb.readHoldingRegisters(1,0,1,registers); // read all bits (4 output bits + 6 input bits)
//int ret = mb.readInputRegisters(1,0,1,registers); // read all bits (4 output bits + 6 input bits)


// PLC_DATA;
typedef struct
{
  int in1, in2, in3;
}PLC_INPUT;

typedef struct
{
  int out1, out2, out3;
}PLC_OUTPUT;

typedef struct
{
  int st1, st2, st3;
  int stm1_running, stm2_running;
```

```
  char stm2_name[32];
}PLC_STATE;

typedef struct
{
  PLC_INPUT in;
  PLC_OUTPUT out;
  PLC_STATE state;
}PLC_DATA;

// PVS_DATA
typedef struct
{
  int button_start_stm2;
}PVS_STATE;

typedef struct
{
  PVS_STATE state;
}PVS_DATA;

// SHM_DATA;
typedef struct
{
  PLC_DATA plc;
  PVS_DATA pvs;
}SHM_DATA;

extern SHM_DATA    *shm_data;
extern rlSharedMemory shm;
extern rlSerial     tty;
extern rlModbus     mb;
extern rlMutex      mb_mutex;

int startStepsStm1(rlState *sm, int cycletime);
int startStepsStm2(rlState *sm, int cycletime);

#endif
```

Eine genauere Beschreibung der Vorlagen finden Sie in deren Verzeichnissen bzw. in der Referenz der Klassen in der rllib. Kontinuierliche Regler lassen sich beispielsweise mit Hilfe der Klasse 'rlController' realisieren. Damit kann man P/PI/PID Regler parametrieren und als parallel laufenden Thread starten/stoppen.
Die Zustandsmaschinen arbeiten mit Zeigern auf Funktionen. Daher entfallen 'switch()' Befehle für die Zustandsübergänge.

8.2 Visualisierung mit Hilfe der Zustandsautomaten

Es hat sich als komfortabel erwiesen vorher Zustandsdiagramme mit `http://graphviz.org` zu erstellen und eine SVG Grafik erzeugen zu lassen. Die SVG Grafik enthält dabei id's für die Objekte, die in der 'Eingangsdatei.dot' spezifiziert werden können. Damit kann die SVG direkt in die Visualisierung übernommen werden, wie Sie auch in der Vorlage sehen. Die in der SVG enthaltenen id's dienen zur Adressierung der graphischen Objekte mit unserer rlSvgAnimator Klasse.

Graphviz Datei zur Generierung des Zustandsgraphen als SVG

```
digraph state_machine_2 {

  subgraph statemachine
  {
    rankdir=LR;
    size="8,5"
    node [shape = doublecircle, id="PV.start", style=filled, fillcolor=grey]; stStart;
    node [shape = doublecircle, id="PV.finish", style=filled, fillcolor=grey]; stFinish;
```

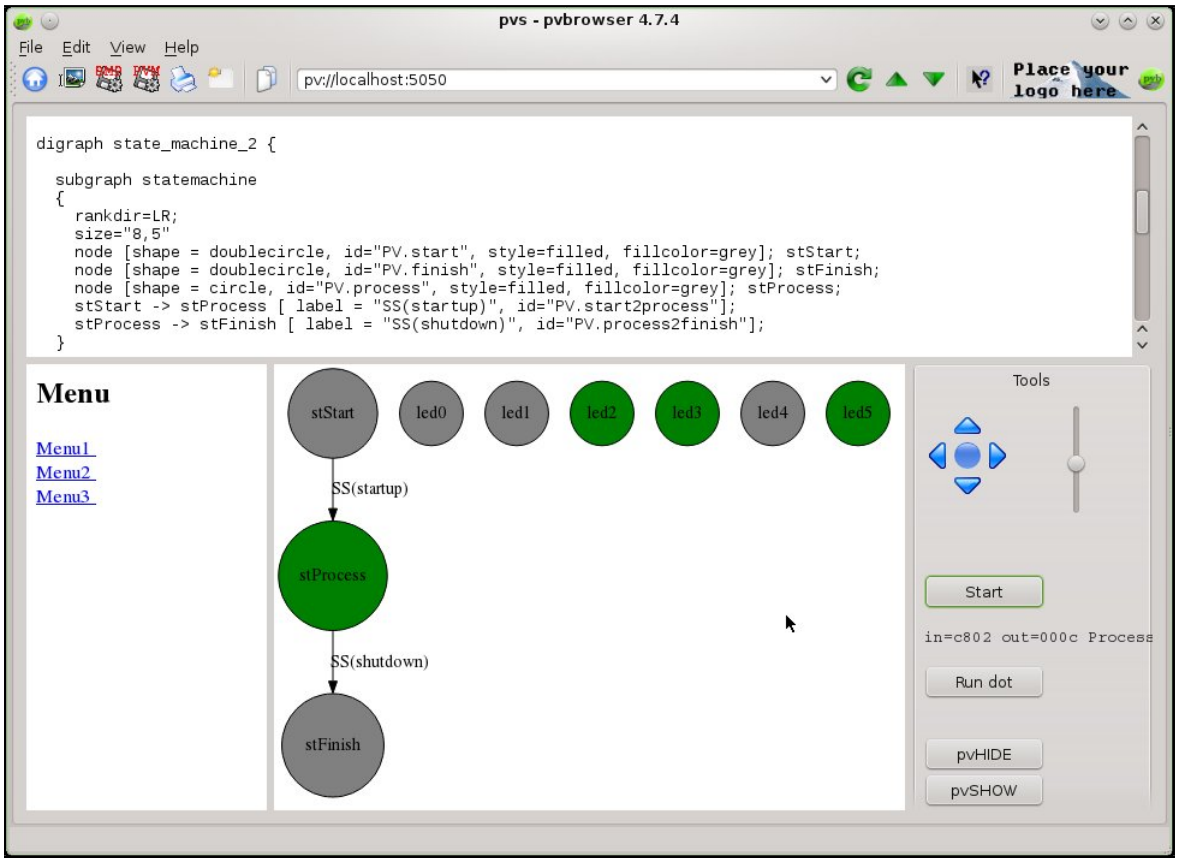

Abbildung 8.1: Statemachine visualization

```
    node [shape = circle, id="PV.process", style=filled, fillcolor=grey]; stProcess;
    stStart -> stProcess [ label = "SS(startup)", id="PV.start2process"];
    stProcess -> stFinish [ label = "SS(shutdown)", id="PV.process2finish"];
  }

  subgraph led
  {
    rankdir=LR;
    size="2,5"
    node [shape = circle, id="PV.led0", style=filled, fillcolor=grey]; led0;
    node [shape = circle, id="PV.led1", style=filled, fillcolor=grey]; led1;
    node [shape = circle, id="PV.led2", style=filled, fillcolor=grey]; led2;
    node [shape = circle, id="PV.led3", style=filled, fillcolor=grey]; led3;
    node [shape = circle, id="PV.led4", style=filled, fillcolor=grey]; led4;
    node [shape = circle, id="PV.led5", style=filled, fillcolor=grey]; led5;
  }
}
```

Code for the visualization

```
// for very lazy people like me
#define RPF(a,b,c) d->svgAnimator.svgRecursivePrintf(a,b,c)

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

  rlString newState(shm_data->plc.state.stm2_name);
```

```
  pvPrintf(p,labelState,"in=%04x␣out=%04x␣%s", shm_data->plc.in.in1, shm_data->plc.out.out1, newState
      .text());

  if(newState != d->oldState)
  {
    modified = 1;
    RPF("PV.start","fill=","grey");
    RPF("PV.process","fill=","grey");
    RPF("PV.finish","fill=","grey");
    if     (newState == "Start") RPF("PV.start","fill=","green");
    else if(newState == "Process") RPF("PV.process","fill=","green");
    else if(newState == "Finish") RPF("PV.finish","fill=","green");
    d->oldState = newState;
  }

  int input = shm_data->plc.in.in1;
  if(input != d->oldInput)
  {
    modified = 1;
    RPF("PV.led0","fill=","grey");
    RPF("PV.led1","fill=","grey");
    RPF("PV.led2","fill=","grey");
    RPF("PV.led3","fill=","grey");
    RPF("PV.led4","fill=","grey");
    RPF("PV.led5","fill=","grey");
    if(input & BIT15) RPF("PV.led3","fill=","green");
    if(input & BIT14) RPF("PV.led2","fill=","green");
    if(input & BIT13) RPF("PV.led1","fill=","green");
    if(input & BIT12) RPF("PV.led0","fill=","green");
    if(input & BIT1) RPF("PV.led5","fill=","green");
    if(input & BIT0) RPF("PV.led4","fill=","green");
    d->oldInput = input;
  }

  if(modified) drawSVG1(p,centerWidget,d);
  return 0;
}
```

Kapitel 9

Einrichtung eines pvserver zum Start im Hintergrund

Ein pvserver sollte beim Booten des Rechners im Hintergrund gestartet werden können. Die dafür verwendetet Methode hängt vom Betriebssystem ab.
Bei einem pvserver kann man über die Präprozessor Direktive USE_INETD entscheiden, ob man einen Multi Threaded Server erhalten möchte oder ob der pvserver mit Hilfe des Inetd Mechanismus gestartet werden soll.
Der Multi Threaded Server besteht aus mehreren Threads. Der Haupt-Thread wartet dabei auf neue Clienten. Wenn sich ein Client verbindet, startet der Haupt-Thread einen neuen Thread, in dem der Client bedient wird. Der Haupt-Thread kann auf weitere Clients warten.
Inetd ist ein Super Server, der auf Clienten verwaltet und dann den eigentlichen Server startet. Dabei wird der Standard Input und Output als Kommunikationskanal verwendet. Inetd leitet die vom Server verwendeten Standard Input und Output Kanäle über das Netzwerk weiter.

9.1 Linux

Für den Multi Threaded pvserver kann man ein 'startscript' aus pvdevelop heraus mit 'Linux->WriteStartscript' generieren.

Aus pvdevelop generiertes Startscript für einen Server

```
#!/bin/sh
# generated by pvdevelop. Please adjust DAEMON_PATH and DAEMON to your needs.
# copy this file to /etc/init.d and link it to runlevel 5 .
DAEMON_PATH=/home/username/directory
DAEMON=pvs
. /etc/rc.status
rc_reset
case "$1" in
    start)
      echo -n "Starting␣$DAEMON"
      startproc $DAEMON_PATH/$DAEMON -sleep=100 -cd=$DAEMON_PATH > /dev/null
      rc_status -v
      ;;
    stop)
      echo -n "Shutting␣down␣$DAEMON"
      killproc -TERM $DAEMON_PATH/$DAEMON
      rc_status -v
      ;;
    try-restart)
      $0 status >/dev/null && $0 restart
      rc_status
      ;;
    restart)
      $0 stop
      $0 start
```

```
    rc_status
    ;;
  force-reload)
    echo -n "Reload␣service␣$DAEMON"
    killproc -HUP $DAEMON_PATH/$DAEMON
    rc_status -v
    ;;
  reload)
    echo -n "Reload␣service␣$DAEMON"
    killproc -HUP $DAEMON_PATH/$DAEMON
    rc_status -v
    ;;
  status)
    echo -n "Checking␣for␣service␣$DAEMON"
    checkproc $DAEMON_PATH/$DAEMON
    rc_status -v
    ;;
  *)
  echo "Usage:␣$0␣{start|stop|status|try-restart|restart|force-reload|reload}"
  exit 1
  ;;
esac
rc_exit
```

Darin können die Variablen DAEMON_PATH und DAEMON angepasst werden. Das 'startscript' muss nun nach /etc/init.d kopiert werden, wo alle Startscripte von Servern gespeichert werden. Nun kann man folgendes tun:

Verwendung des Startscript

```
su
cd /etc/init.d
./startscript status
./startscript start
./startscript stop
```

Um IHREN_PVS in dem richtigen Runlevel zu starten, müssen Sie einen Link in 'rc5.d' anlegen. Unter openSUSE kann man dies mittels YaST im Runlevel Editor tun. Wählen Sie IHREN_PVS und aktivieren Sie Ihn einfach.
Falls Sie inetd oder xinetd verwenden möchten, tun Sie das Folgende. Installieren und aktivieren Sie (x)inted. In /etc/services fügen Sie eine Zeile hinzu.

Definition eines pvserver auf Port 5051

```
pvsuper       5051/tcp      # pvs super server
```

Dies definiert einen Service pvsuper auf Port 5051. In /etc/xinted.d braucht man folgende Datei.

/etc/xinetd.d/pvsuper

```
# default: off
# description: pvsuper ProcessViewServer daemon
service pvsuper
{
      socket_type    = stream
      protocol       = tcp
      wait           = no
      user           = root
      server         = /your/directory/pvsuper
      server_args    = -port=5051 -cd=/your/directory/
      disable        = no
}
```

Um den pvserver zu aktivieren, müssen Sie xinted bzw. inted neu starten.

xinetd neu starten

```
cd /etc/init.d
./xinetd stop
./xinetd start
```

9.2 Windows

Um pvserver als Windows Service im Hintergrund zu starten, können Sie xyntservice `http://www.codeproject.com/KB/system/xyntservice.aspx` benutzen.
Da xynetservice anscheinend aus dem Internet verschwunden ist, stellt pvbrowser nun das kleine Werkzeug 'pvservice' als Ersatz zur Verfügung. `http://pvbrowser.de/pvbrowser/index.php?lang=en&menu=6&left=3`.
pvserver als inetd laufen zu lassen ist unter Windows nicht möglich, da es keinen inetd mitbringt.
Wenn Sie den pvserver aus Windows Autostart Ordner starten, erscheint eine DOS Box in der der pvserver läuft. Um die DOS Box zu vermeiden können Sie unserer Hilfswerkzeug pvb/win-mingw/bin/start_pvbapp.exe benutzen.

9.3 OpenVMS

Sie können loginout.exe benutzen, um Ihren pvserver im multithreaded mode zu starten. Dieses Kommando kann in SYS$MANAGER:SYSTARTUP_VMS.COM plaziert werden.

loginout

```
$ @loginout dka0:[your.server]your_server.com
```

your_server.com

```
$ set default dka0:[your.server]
$ your_server := dka0:[your.server]your_server.exe
$ your_server -sleep=100 -port=5050
```

loginout.com

```
$ loginout:
$
$   device   = f$parse("''p1'",,,"device")
$   directory = f$parse("''p1'",,,"directory")
$   name     = f$parse("''p1'",,,"name")
$   type     = f$parse("''p1'",,,"type")
$   run   /dump                                    -      !
          /detach                                  -      !
          /proc='name'                             -      !
          /prio=8                                  -      !
          /noswap                                  -      !
          /working_set=1500                        -      !
          /maximum_working_set = 3600              -      !
          /page_file=60000                         -      !
          /uic=[200,1]                             -      !
          /out='device''directory''name'.out       -      !
          /err='device''directory''name'.err       -      !
          /input='device''directory''name'.com     -      !
          sys$system:loginout.exe
$
```

Sie können UCX SET SERVICE benutzen, um einen pvserver im inetd mode zu starten.

pvserver_setup.com

```
$ ucx set service pvserver /file=dka100:[lehrig]pvserver_startup.com -
                       /user=lehrig                          -
```

```
                        /protocol=tcp                          -
                        /port=5050                             -
                        /process=pvserver                      -
                        /limit=10
$ ucx enable service pvserver
$ ucx show service pvserver /full
```

Die Startup Datei für ucx set service sieht dann wie folgt aus.

pvserver_startup.com

```
$ set default dka100:[lehrig.cc.processviewserver]
$ run dka100:[lehrig.exe]pvserver.exe
```

9.4 pcontrol

In der Prozessüberwachung haben Sie eine Anzahl von Prozessen, die Ihre Automation ausmachen. Die meisten dieser Prozesse laufen im Hintergrund. Es muss eine Methode geben, um diese zu überwachen und zu steuern. Ebenso sollten die Prozesse Ereignismeldungen an eine zentrale Instanz senden können. Die Ereignismeldungen müssen online und offline für historische Aufzeichnungen ausgewertet werden können.
pcontrol ist ein pvserver, der auf rllib aufbaut und pvbrowser verwendet, um diese Aufgaben wahrzunehmen. Der 'pvserver - pcontrol' startet die Hintergrundprozesse und überwacht Sie. Wenn Sie ein komplexeres Automationssystem automatisch im Hintergrund starten wollen, kann man pcontrol verwenden. pcontrol wird dazu wie zuvor beschrieben im Hintergrund gestartet. Die weiteren Prozesse werden dann von pcontrol gestartet und überwacht.

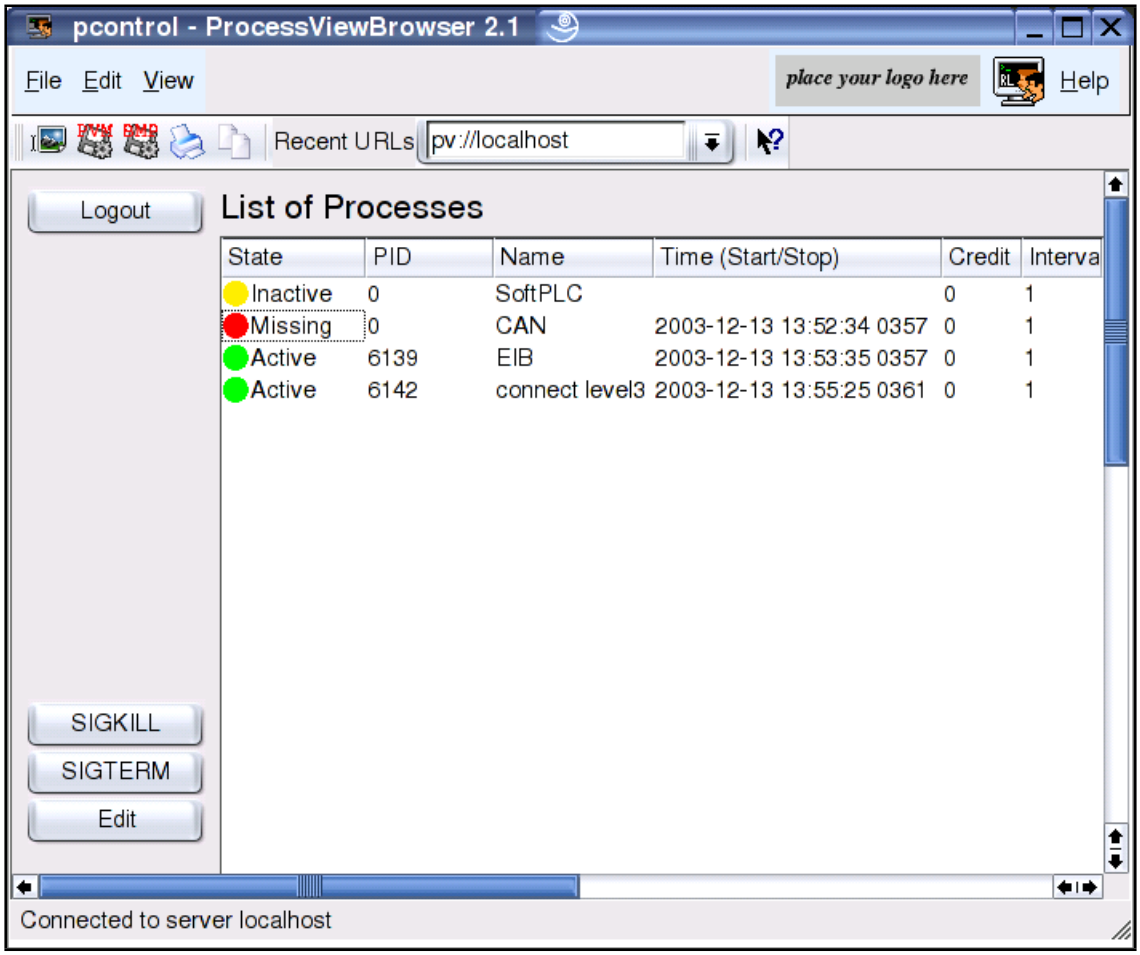

Abbildung 9.1: pcontrol steuert Hintergrundprozesse

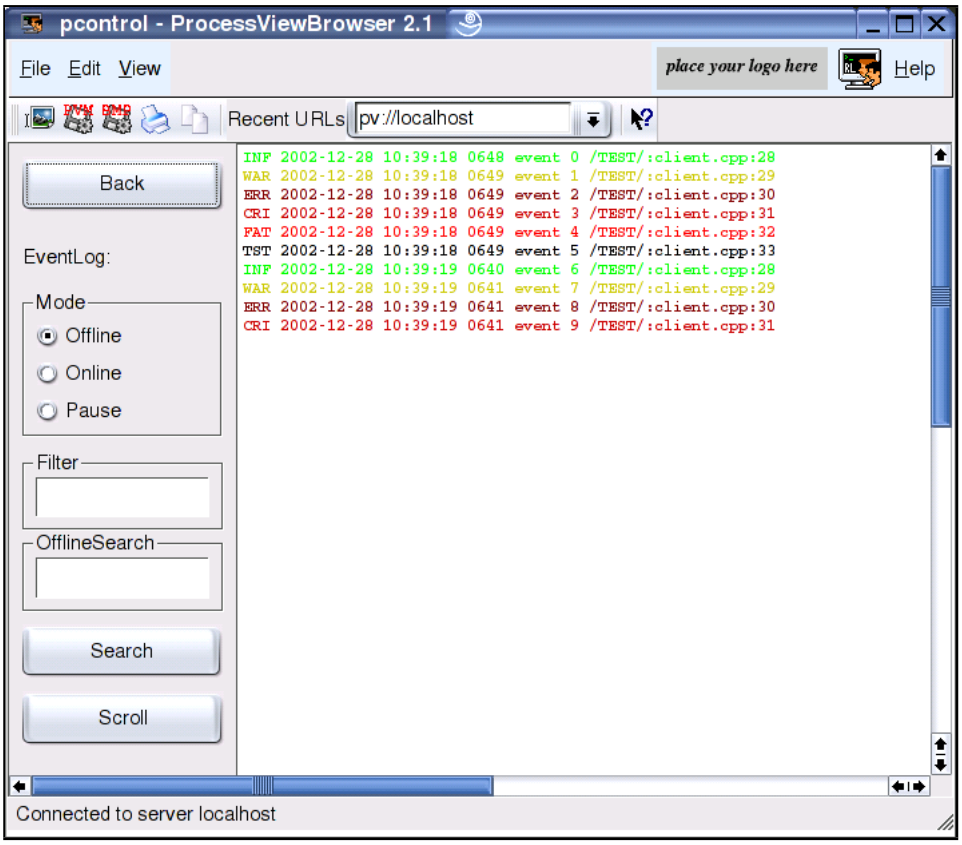

Abbildung 9.2: Ereignisse können von pvbrowser eingesehen werden

Von pcontrol können Ereignismeldungen anderer Prozesse (auch über das Netzwerk) empfangen werden. In der rllib gibt es ein paar Funktionen, die es solchen Prozessen erlauben die Ereignismeldungen zu versenden. Zunächst wird der Server pcontrol mit rlEventInit() definiert. Dann kann man mit rlEvent() Botschaften ähnlich einem printf() ausgeben. Dabei wird das Datum/Uhrzeit, die Quelltextdatei und Zeile in der die Botschaft gesendet wird mit in die Ereignismeldung mit aufgenommen.

Ausgabe von Ereignismeldungen

```
#include "rlevent.h"

int main()
{
  char *argv[] = {"","-eventhost=localhost","-eventport=6003"};
  int i = 0;

  rlEventInit(3,argv,"/TEST/");
  while(1)
  {
    rlEvent(rlInfo   ,"event␣%d",i++);
    rlEvent(rlWarning ,"event␣%d",i++);
    rlEvent(rlError  ,"event␣%d",i++);
    rlEvent(rlCritical,"event␣%d",i++);
    rlEvent(rlFatal  ,"event␣%d",i++);
    rlEvent(rlTest   ,"event␣%d",i++);
    rlsleep(1000);
    if(i > 100*6) break;
  }
  return 0;
}
```

9.5 Zugriffssteuerung

Mit den folgenden Funktionen kann man einstellen, wie viele Clients sich von einer einzigen IP Adresse aus verbinden können und wie viele Clients sich insgesamt mit dem pvserver verbinden können. Der Wert von 'max_clients' muss zwischen 1 und MAX_CLIENTS liegen. Diese Funktionen können in main() eingefügt werden. Standardmäßig stehen die Werte auf MAX_CLIENTS = 100.

Einstellung der maximal erlaubten Clients

```
int pvSetMaxClientsPerIpAdr(int max_clients);
int pvSetMaxClients(int max_clients);
```

Damit man eine sichere Verbindung eines pvbrowser über das Internet mit einem pvserver bekommt, ist es sinnvoll bestimmte IP Adressen explizit zu erlauben oder andere IP Adressen zu verbieten. Wenn Sie diese Filter aktivieren möchten, speichern Sie die Dateien 'deny.ipv4' und 'allow.ipv4' in dem Verzeichnis aus dem Ihr pvserver läuft.
Wenn keine dieser Dateien existiert, kann man sich von beliebigen IP Adressen aus verbinden.
Wenn nur die Datei 'allow.ipv4' existiert, kann man sich nur von den IP Adressen aus verbinden, die explizit erlaubt worden sind.
Wenn nur die Datei 'deny.ipv4' existiert, kann man sich von allen Adressen aus verbinden, die nicht explizit gesperrt worden sind.
Wenn sowohl 'allow.ipv4' und 'deny.ipv4' existieren, verhält es sich so, als wäre nur die Datei 'allow.ipv4' vorhanden.

Beispiel: allow.ipv4

```
# Ein Kommentar
# Hier geben Sie bitte eine Liste von IP Adressbereichen ein, denen erlaubt wird sich zu verbinden.
# Eine gueltige Adresse / Adressbereich hat die Form aaa.bbb.ccc.ddd/anzahl_signifikante_bits
192.168.1.0/24 # erlaube den Zugriff aus den privaten Subnetz 192.168.1.X
192.168.0.14/32 # erlaube den Zugriff von IP Adresse 192.168.0.14
# Geben Sie beliebig viele Adressen / Adressbereiche ein.
```

Beispiel: deny.ipv4

```
# Hier geben Sie bitte eine Liste von IP Adressbereichen ein,
# denen nicht erlaubt wird, sich zu verbinden.
# Eine gueltige Adresse / Adressbereich hat die Form aaa.bbb.ccc.ddd/anzahl_signifikante_bits
100.101.0.0/16 # verbiete den Zugriff aus dem Adressbereich 100.101.xxx.yyy
200.201.0.0/16 # verbiete den Zugriff aus dem Adressbereich 200.201.xxx.yyy
```

Ein mögliches Szenario wäre es, nur dem privaten Subnetz Zugriff auf den pvserver zu geben. Wenn eine externe Wartung erfolgen soll, könnte der Operator die Adresse des Wartungstechnikers explizit freischalten und damit den Zugriff über das Internet nur für diesen Fall erlauben.
Die entsprechenden Funktionen für IPv6 verwenden die Dateien 'allow.ipv6' und 'deny.ipv6' und werden analog zu IPv4 benutzt.

Beispiel: allow.ipv6

```
# Hier geben Sie bitte eine Liste von IP Adressbereichen ein, denen erlaubt wird sich zu verbinden.
# Eine gueltige Adresse / Adressbereich hat die Form ipv6_adresse/anzahl_signifikante_bits
::1/128
0001:0002:0003:0004:0005:0006:0007:0008/128
0000:0000:0000:0000:0000:0000:1000:0000/128
# Geben Sie beliebig viele Adressen / Adressbereiche ein.
```

Kapitel 10

Weiterführende Möglichkeiten

pvbrowser stellt ein Framework für die Prozessvisualisierung dar, das mit C/C++ arbeitet. Sie sind nicht auf die oben beschriebenen Möglichkeiten eingeschränkt, sondern können weitere Möglichkeiten durch eigene oder fremde Bibliotheken mit einbauen.

10.1 Diverse Datenbanken einbinden

Zur Einbindung diverser Datenbanken bietet es sich an, die Qt Bibliothek zu verwenden, da diese Bibliothek schon mit dem von pvbrowser verwendeten Entwicklungssystem mitgeliefert wird.
Das 'SQL Module' von Qt bietet ein komfortables Programmierinterface für SQL Datenbanken. Durch Plugins werden alle gängigen Datenbanksysteme unterstützt. Um die Qt Bibliothek aus einem pvserver verwenden zu können, muss lediglich das 'CONFIG -= qt' aus der Projektdatei Ihres pvserver entfernt werden. Dann wird qmake die Qt Bibliothek mit anbinden und Sie können die Qt SQL Klassen verwenden.
Ein Beispiel für die Verwendung des Qt SQL moduls in einem pvserver findet man in pvbaddon.tar.gz im Verzeichnis:
pvbaddon/templates/qtDatabase

10.2 Tabellenkalkulation auf dem Client verwenden

Wenn Sie Tabellenkalkulationsprogramme benutzen wollen, um den Benutzern der Visualisierung die Möglichkeit zu geben eigene Auswertungen zu machen, können Sie auf dem Server eine CSV Datei erzeugen und sie mit pvDownloadFile() auf den Client Computer laden. Mit der Funktion pvClientCommand kann die Tabellenkalkulation dann mit dieser CSV Datei gestartet werden.

pvClientCommand

```
int  pvClientCommand (PARAM *p, const char *command, const char *filename);
// Beispiel:
// pvClientCommand(p, "csv", "test.csv");
```

Der 'csv-viewer' wird unter den Optionen im pvbrowser Client definiert. Hier können Sie z.B. OpenOffice oder Excel einsetzen.

10.3 Generieren von Reports mit LaTeXund PDF Datei erzeugen

Zur Dokumentation und Qualitätssicherung in der Produktion werden häufig Reports erstellt. Ein gutes Format für Reports ist PDF. Sie könnten z.B. auf dem Server LaTeXinstallieren, Dokumenten-vorlagen in LaTeXerstellen und darin Ergebnisse aus der Produktion eintragen. Nachdem Sie mit Hilfe von LaTeXeine PDF Datei erzeugt haben, kann die PDF Datei mit pvDownloadFile() auf den Client Computer geladen werden.
Mit der Funktion pvClientCommand kann ein Viewer mit dieser PDF Datei gestartet werden.

pvClientCommand

```
int  pvClientCommand (PARAM *p, const char *command, const char *filename);
// Beispiel:
```

```
// pvClientCommand(p, "pdf", "test.pdf");
```

Der 'pdf-viewer' wird unter den Optionen im pvbrowser Client definiert. Hier können Sie z.B. Okular oder Acrobat Reader einsetzen.

10.4 Generieren von Reports mit HTML und PDF Datei erzeugen

Mit dem Kommandozeilen Werkzeug rlhtml2pdf können Sie HTML Reports in PDF Format konvertieren. rlhtml2pdf basiert allerdings auf der QTextDocument Klasse und kann nicht alle HTML und CSS Features umsetzen. Mit dem folgenden Befehl bekommen Sie eine Hilfe zu rlhtml2pdf.

Hilfe zu rlhtml2pdf

```
rlhtml2pdf --help
```

10.5 Statistische Auswertungen

Es gibt eine große Anzahl von Statistikprogrammen, die kommandozeilen-orientiert arbeiten und Bitmap Dateien als Ergebnis erzeugen. Sie können diese Statistik Programme genau wie die externen Plot Werkzeuge einbinden. Unter unixoiden Betriebssystemen auf der Serverseite empfiehlt es sich die Klasse rlSpawn aus der rllib zu verwenden. Diese Klasse startet externe Anwendungen und verbindet deren STDIN und STDOUT über eine Pipe mit der rlSpawn Instanz. Sie können die externe Anwendung auf diese Weise über rlSpawn von pvserver bedienen lassen. Die Ergebnisse der statistischen Auswertungen lassen sich dann in die Visualisierung oder in die mit LaTeXgenerierten Reports einbinden.

10.6 Ramdisk für 'temp' Verzeichnis von pvbrowser

pvbrowser Clients verwenden ein temporäres Verzeichnis in das Dateien geschrieben werden können. Das temporäre Verzeichnis wird in den Optionen des pvbrowser Client angegeben. Sie könnten das Verzeichnis in eine Ramdisk legen, um weniger Plattenzugriffe zu haben. Ausserdem hätte das den Vorteil, dass dieses Verzeichnis nach einem Neustart der Client Computer geleert ist.

Ramdisk mit 512MB Kapazität unter Linux im Unterverzeichnis ram anlegen

```
mount -t tmpfs -o size=512m none ram
```

Kapitel 11

Zusammenfassung

In dieser Dokumentation wurde pvbrowser dargestellt. Mit diesem Framework können sehr schnell und relativ einfach Visualisierungen mit der dazugehörigen Datenerfassung erstellt werden. Der Zeitaufwand für eine Visualisierung lässt sich mit dem Aufwand für die Erstellung von Webseiten vergleichen. Die mit pvbrowser realisierten Visualisierungen sind von hause aus netzwerkfähig. Ähnlich einem Webbrowser können die einzelnen Visualisierungen/pvserver ausgewählt werden und man kann eine ganze Anlage mit diesem verteilten System überwachen und steuern. Da die Kommunikation über TCP/IP erfolgt, ist pvbrowser sogar über das Internet einsetzbar.
Es sollte jedoch ausdrücklich davor gewarnt werden, anlagenkritische Teile damit für Internetzugriffe zu öffnen. Die für interne Zwecke realisierte Visualisierung sollte z.B. Ports verwenden, die von einer Firewall gegen Internetzugriffe abgeschottet werden. Die öffentlich zugänglichen Teile sollten lediglich eine Beobachtung erlauben aber keine Eingriffe in die Anlage zulassen. Wenn jedoch trotzdem Zugriffe über das Internet freigegeben werden, sollte man das pvssh:// Protokoll verwenden (Secure Shell) und ein starkes Passwort verwenden. Sinnvoll wäre es, die Verbindung zu trennen, wenn jemand eine falsches Passwort eingibt. Damit können 'Brute Force Attacken' auf das Passwort verhindert werden.
Da pvbrowser auf allen gängigen Betriebssystemen lauffähig ist, sind Ihnen kaum Grenzen gesetzt. Das Fachpersonal kann z.B. die Vorteile unix-artiger Betriebssysteme nutzen, um die Server zu erstellen und zu verwalten, während normale Benutzer weiter mit Ihrem gewohnten Betriebssystem arbeiten. Es können aber auch Visualisierungen im Vollbildmodus eingestellt werden, bei denen die Benutzer gar nicht wissen, welches Betriebssystem dahinter steht.
Wir hoffen, dass Sie pvbrowser erfolgreich einsetzen und würden uns freuen, wenn Sie uns helfen könnten pvbrowser weiter zu verbessern, sei es durch Anregungen für neue Funktionen, Fehlerrückmeldungen, Beiträge zur Dokumentation, Einsendung von Patches oder gar der zur Verfügungstellung von Treibern/Daemonen für weitere Protokolle.
Wir wünschen Ihnen viel Erfolg bei der Benutzung von pvbrowser.

Ihre pvbrowser Gemeinschaft

`http://pvbrowser.org` 19. Juli 2016

Anhang

Abbildungsverzeichnis