pvbrowser manual

http://pvbrowser.org

July 19, 2016

Contents

Credits

The idea for pvbrowser was developed at SMS Siemag AG.
There they wanted a process visualization with sourcecode available in order to be able to make adjustments if necessary. Solutions on base of Java and Qt were proposed. The Java variant was preferred at SMS Siemag AG but the Qt variant was also started as open source project under the name pvbrowser. The main features of pvbrowser should be independence from platform and strict client/server architecture.
The Java variant was not traced for a long time but pvbrowser was soon used at SMS Siemag AG. We have to thank the staff of the company for early testing of the software and using it up to now. Without their support we could not have afforded the license fees for the commercial version of Qt.
After publishing pvbrowser the community of users and developers contributing to pvbrowser grew. Without these users and developers it would have not been possible the reach the current status. Many ideas were developed and errors were eliminated.
The students and their professors doing master thesis in cooperation with our project are to appreciated for their excellent work.
The company Nokia/Trolltech must be honoured for putting Qt under LGPL license. The support staff at Nokia/Trolltech helped to solve difficult problems. Furthermore we are very happy with the rapid progress Qt has taken.
pvbrowser uses some components developed by other open source projects. We thank all these projects from the open source community without which the whole solution would not have been possible.
Our special thanks goes to our families who had big patience with us.

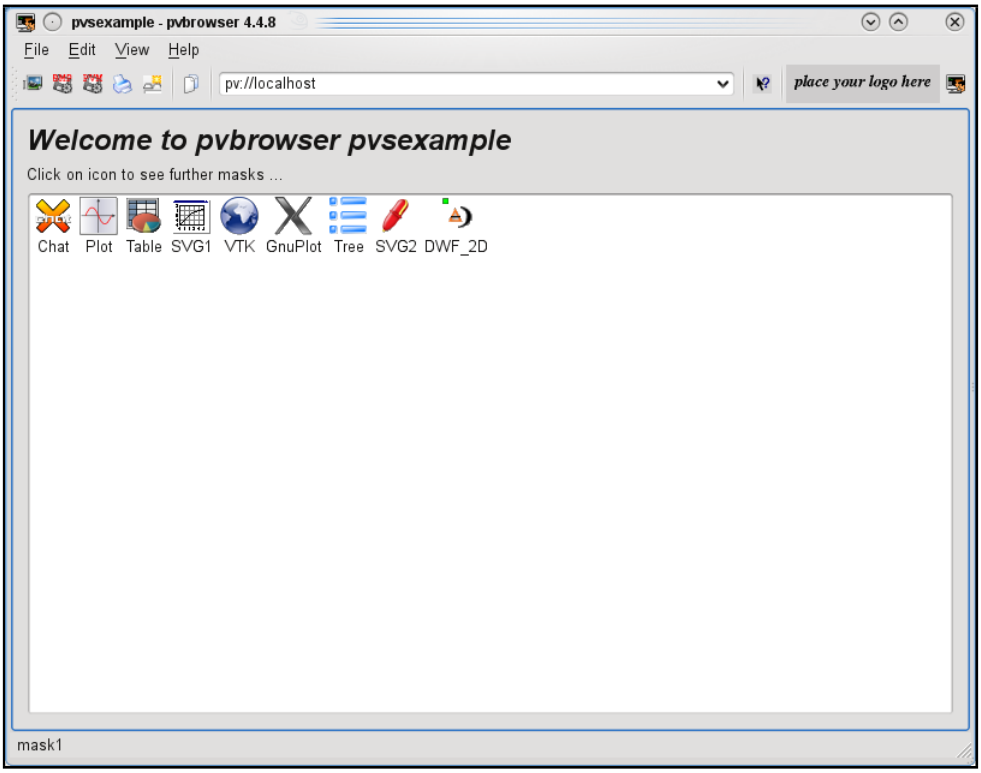

Figure 1: pvsexample start mask

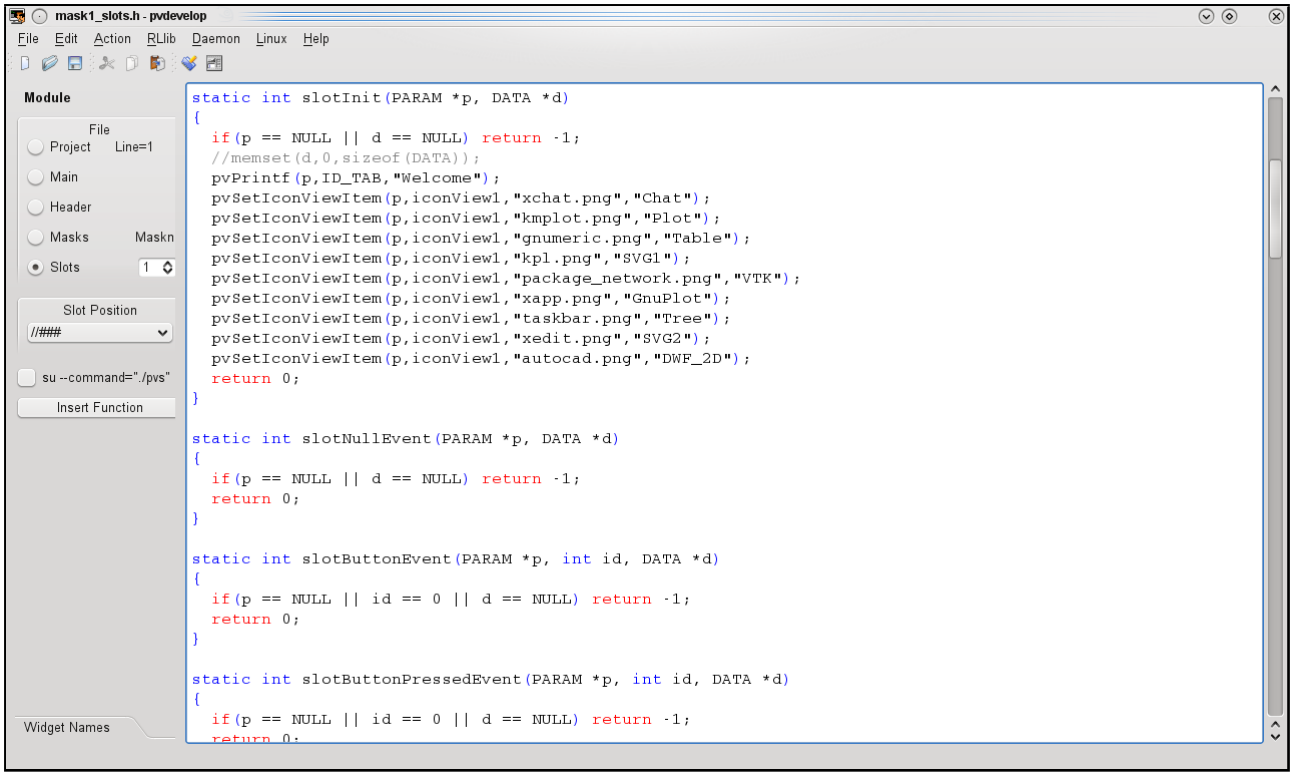

Figure 2: pvsexample start mask in pvdevelop

Preface

What is SCADA ?

From wikipedia: SCADA stands for supervisory control and data acquisition. It generally refers to an industrial control system: a computer system monitoring and controlling a process. The process can be industrial, infrastructure or facility-based as described below:

- Industrial processes include those of manufacturing, production, power generation, fabrication, and refining, and may run in continuous, batch, repetitive, or discrete modes.
- Infrastructure processes may be public or private, and include water treatment and distribution, wastewater collection and treatment, oil and gas pipelines, electrical power transmission and distribution, civil defense siren systems, and large communication systems.
- Facility processes occur both in public facilities and private ones, including buildings, airports, ships, and space stations. They monitor and control HVAC, access, and energy consumption.

project file for a visualization with pvbrowser. Using qmake a Makefile is generated from this file.

```
#######################################################################
# 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
```

Chapter 1

Introduction

pvbrowser is a SCADA software under GPL. With GPL license you have to contribute your solutions to our project for publishing.
pvbrowser provides a framework for process visualization. These are the characteristics of pvbrowser.

- Client/Server
- Qt Widgets
- Custom Widgets
- platform independent
- SVG graphic
- xy graphic
- 3D graphic
- Webpages using WebKit
- IDE support
- graphical design
- C/C++
- Lua, Python
- Multithreaded or Inetd
- Unicode support (Chinese, Arabic, Cyrillic, ...)
- Support of ssh-urls
- Connections to fieldbuses
- Connections to PLC
- Control of background processes
- Central event log
- You can code your own authorization
- GPL license

The pvbrowser client can be compared to the normal web browser. But not almost static content is presented. Instead it shows dynamic content as is necessary with SCADA and in other areas. With pvbrowser any visualization can be implemented which are distributed across the plant or even over the internet and can be browsed using hyperlinks.
The frame of the pvbrowser windows is controlled by pvbrowser itself but the content is completely designed by the developer of the visualization who do it at the remote computers. If a visualization is modified nothing is necessary to do at the client computers.
The usage of the window is determined by the custom made implementation of the visualization. The usage of the client window should be self explaining. With a click on "?" next to the URL input and a click on the according element in the toolbar you will get a context sensitive help. The help menu on the upper right corner is in fact a HTML page starting with index.html . This page must be downloaded at start up using pvDownloadFile() . If index.html does not exist there is no help available for the according visualization. The help can use all features provided by WebKit.
pvbrowser can be customized using the editor under File-Options. Please notice that some changes are applied after a restart of the client only.
The following command line options are available.

command line options for 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
```

The pvbrowser client will connect to a pvserver via TCP/IP. The pvserver will then start sending ASCII text to the client which will be interpreted there and result in a call to a Qt method.
In the opposite direction the pvbrowser client will also send ASCII text to the pvserver when the user triggers an event like clicking a button for example. This text will be interpreted within an event loop in pvserver. This will result in a call to the according 'slot functions'.
The task of the developer of a visualization is to code these 'slot functions'. But the skeleton of the pvserver will be generated and cared about by pvdevelop. Thus the developer can concentrate on the 'slot functions'. The layout of the masks is designed graphically.

A typical function in pvserver which sends ASCII text to the pvbrowser client.

```
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;
}
```

As you can see by the above function the developer of the visualization does not need to know the ASCII text. There is a library that encapsulates this. The parameter PARAM is used as first parameter in all these functions and describes the connection to the client. Especially within PARAM there is the socket for communication with the pvbrowser client over TCP/IP.
pvbrowser implements it's individual protocol pv:// but supports the http:// protocol also and thus can show normal websites. This is implemented by WebKit which is integrated into Qt. You can link between normal web pages and pvservers and vice versa by hyperlinks. Web pages could even be part of a mask in pvbrowser. There are solutions for connecting a PLC or fieldbus system which support many different protocols. It is also possible to use different protocols at the same time.

Chapter 2

Installation

pvbrowser runs on Linux / Unix / Windows and Mac OS-X. A pvserver additionally can run on OpenVMS.

The sources of pvbrowser are included within the archive `http://pvbrowser.de/pvbrowser/tar/pvb.tar.gz` . Please load this archive if you want to compile pvbrowser yourself. Binary packages for Windows and OS-X are available from our homepage. Binary packages for the most popular Linux distributions are on the openSUSE buildservice. There is a link to the buildservice on our homepage.

Additionally there is the package `http://pvbrowser.de/pvbrowser/tar/pvbaddon.tar.gz` , in which you find some demos and templates for pvbrowser. Especially you will find examples for data acquisition using different protocols.

We will not describe the installation of the binary packages because this is done with the standard procedure for installing software on your operating system. But you have to notice that some additional software is needed if you want to develop your own visualizations with our IDE pvdevelop. You can run the pvbrowser client without needing to install these packages. Only the Qt libraries are needed. For OS-X please install them from the Nokia/Trolltech page. On Windows these libraries are already included within the binary package. On Linux these libraries should already be installed by standard. From the following boxes you see which additional software is necessary for developing you own pvservers on you operating system.

Building and testing the software on Linux and OS-X

```
#
# Linux:
#   qt4-devel inclusive WebKit , make and gcc must be installed
# OS-X:
#   xcode, X11User and the Qt SDK must be installed
#
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 must be adjusted a little bit that it can find the Qt libraries on your operating system. Please edit build.sh and adjust it according to the instructions found in there.

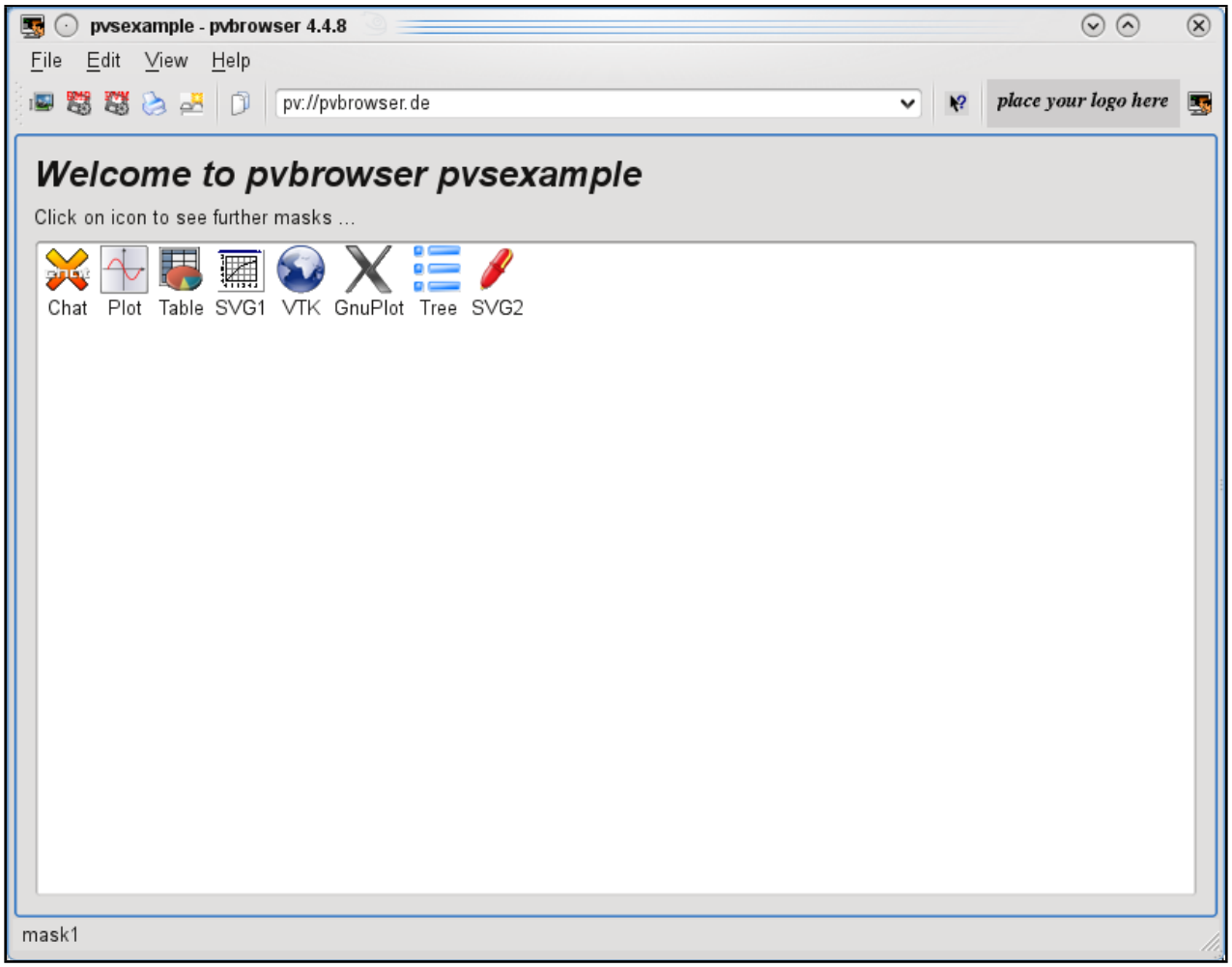

Figure 2.1: pvbrowser after calling 'pvbrowser pv://pvbrowser.de'

Building and testing on Windows

```
#
# The Qt SDK for Windows and MinGW must be installed
# The Qt SDK asks if you want to install MinGW also if you run that installation program
# The environment variables QTDIR and MINGWDIR must be set correctly
# thus these tools can be found.
#
Download and and unpack http://pvbrowser.org/tar/pvb.tar.gz
Open a DOS-Box and go to the directory pvb.
cd win-mingw
CreatePvbWithMinGW.bat
cd bin
pvbrowser pv://pvbrowser.de
```

Building the libraries for pvserver on OpenVMS

```
#
# the CXX C++ Compiler must be installed
#
Download and unpack http://pvbrowser.org/tar/pvb.tar.gz
Go to directory pvb.
@vms_build.com
```

Chapter 3

pvbrowser client

The pvbrowser client works like a web browser but besides the http:// protocol it primarily uses it's own protocols pv:// and pvssh://. The own protocols are connection oriented that is the connection to the server stays alive until the session is terminated. In contrast to that the http:// protocol is connectionless that is a connection is opened and it will read a web page and close the connection immediately. When clicking a link the connection will be opened again. For the purpose of process visualization a connection oriented protocol it suited better because you can do a better authentication and do not have the overhead of the http:// protocol.

Similar to a web browser the pvbrowser client has a address line in which you can input the URL (internet address). pvbrowser supports the above 3 protocols. The http:// protocol is supported by WebKit which is integrated in Qt. You can link between these protocols and surf from one page to the other.

Linking between masks and websites

```
pvHyperlink(p,"pv://pvbrowser.org"); // Linking from a pvserver to our example test server
pvHyperlink(p,"pvssh://user@pvbrowser.org"); // Linking using secure shell
pvHyperlink(p,"http://google.com"); // Linking from a pvserver to a search engine

<a href="pv://pvbrowser.org">Linking to the pvserver on our homepage</a>
<a href="pvssh://user@pvbrowser.de">Linking to a pvserver using secure shell</a>
```

The usage of the client should be self explaining. But we want to say something about the customization of the client. The configuration is stored in a file located in the HOME directory of the user. This file can be edited from the file menu of pvbrowser. Please notice that some settings only take effect after restarting the client.

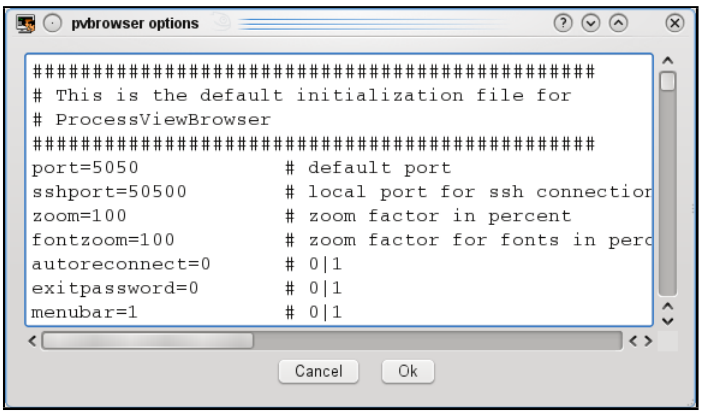

Figure 3.1: Dialog for pvbrowser options

Configuring the pvbrowser client

```
port=5050                   # Standard port
sshport=50500               # Local port for ssh connections
zoom=100                    # Zoom factor in percent
fontzoom=100                # Zoom factor for letters in percent
autoreconnect=0             # 0|1 automatic reconnection after connection lost
exitpassword=0              # 0|1 determines if the user must input a password
                            # in order to leave pvbrowser
menubar=1                   # 0|1 switch on/off menubar
toolbar=1                   # 0|1 switch on/off toolbar
statusbar=1                 # 0|1 switch on/off statusbar
scrollbars=1                # 0|1 switch on/off scrollbars
fullscreen=0                # 0|1 switch to fullscreen mode
maximized=0                 # 0|1 you can start with a maximized window
cookies=1                   # 0=No 1=Yes 2=Ask handling of cookies
echo_table_updates=0        # 0|1 determines if table updates should be echoed when cells are changed
                            # by the server
temp=/tmp                   # directory in which pvbrowser will store temporary files
customlogo=/opt/pvb/custom.png # logo in the menubar that can be set by the user
newwindow=pvbrowser         # command for starting a new window
ssh=ssh                     # tool for secure shell. Under Windows putty.exe
initialhost=pv://localhost  # starting page with which to connect
# The pvserver can store some files in the temporary directory of the client and
# instruct pvbrowser to call an external program to handle such a file.
# This can be usefull for printing protocols on the client computer or calling
# a spreadsheet program with a CSV file in order to do evaluations by the user.
view.pdf=okular             # program for PDF
view.img=gimp               # program for bitmap images
view.svg=inkscape           # program for SVG images
view.txt=kwrite             # program for text files
view.csv=ooffice            # program for CSV files (under Windows probably Excel)
view.html=firefox           # web browser
# Additionally the language setting follow.
# Currently english, german, french and spanish are included.
# You can add any additional language in there (also non latin languages)
```

URL's in pvbrowser

```
pv://server:portnumber/mask?parameter

Examples:
pv://pvbrowser.org
pv://pvbrowser.org:5050
pv://192.168.1.14:6000
pv://localhost/mask1?testvalue=1
pvssh://user@pvbrowser.org
http://www.google.com
```

Chapter 4

pvdevelop IDE

pvdevelop is a IDE for the development of pvservers where C/C++ or Lua and Python can be used. When using pvdevelop you do not have to care about the project file, the Makefile and the skeleton of a pvserver.
The project file, the Makefile and the skeleton of the pvserver are automatically generated and updated. You can easily browse and edit the sourcecode and graphically design the masks the visualization is consisting of.
In editor mode you can see and edit the project file. On the left side of the windows you can see a toolbox. Using the toolbox you can switch between the different files your pvserver is consisting of. If you click 'Insert Function' a selection tree appears from which you can choose the functions for your pvserver. Please notice that you must position the text cursor to the place where you want to insert the function.
If the toolbox is switched to 'Widget Names' you see a list of the widgets you have designed. You can choose from this list and insert the most frequently used functions.
In designer mode the widgets can be designed graphically. Please choose from the popup menu available over a right mouse click. After inputting a widget you can position and resize the widget with the mouse (lower right corner of the widget). With the properties dialog (click widget, right mouse) you can set the properties of the widget.
In designer mode the mouse is grabbed (grab mouse/release mouse). If you choose 'release mouse' you can test your design. Then it is acting like in the finished application. To continue with the design choose 'grab mouse'.
Within the file projectname.pvproject you can define in which steps the widgets can be positioned.
If you prefer to design with Qt Designer you can use the import/export function for UI files from the 'Action' menu in editor mode.
The development could be done in another IDE like Eclipse also. Especially Qt Creator from Nokia/Trolltech is very usefull because it uses the same project files as pvdevelop is using.
You can concentrate on coding the 'slot functions'. The rest of the sourcecode a pvserver is consisting of is generated and updated by pvdevelop.
Because a Makefile is generated for compiling you can even use any ASCII editor for development and compile from the command line using make.

compiling from command line

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

With the help menu in pvdevelop you can open the reference manual for the libraries used for controlling the pvbrowser window and for the rllib used for serverside programming.

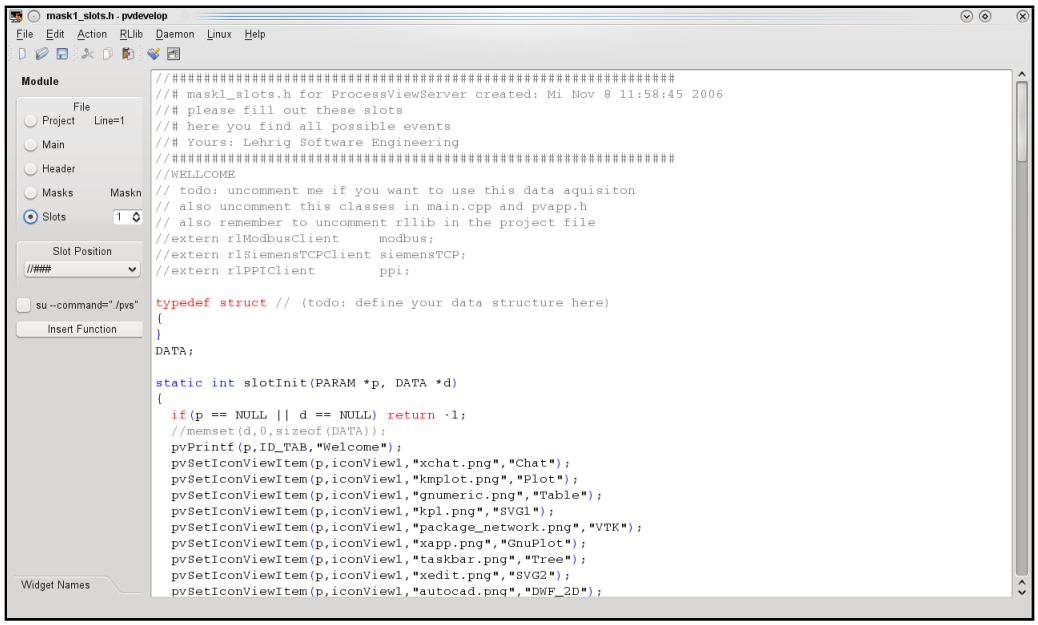

Figure 4.1: main window of pvdevelop in editor mode

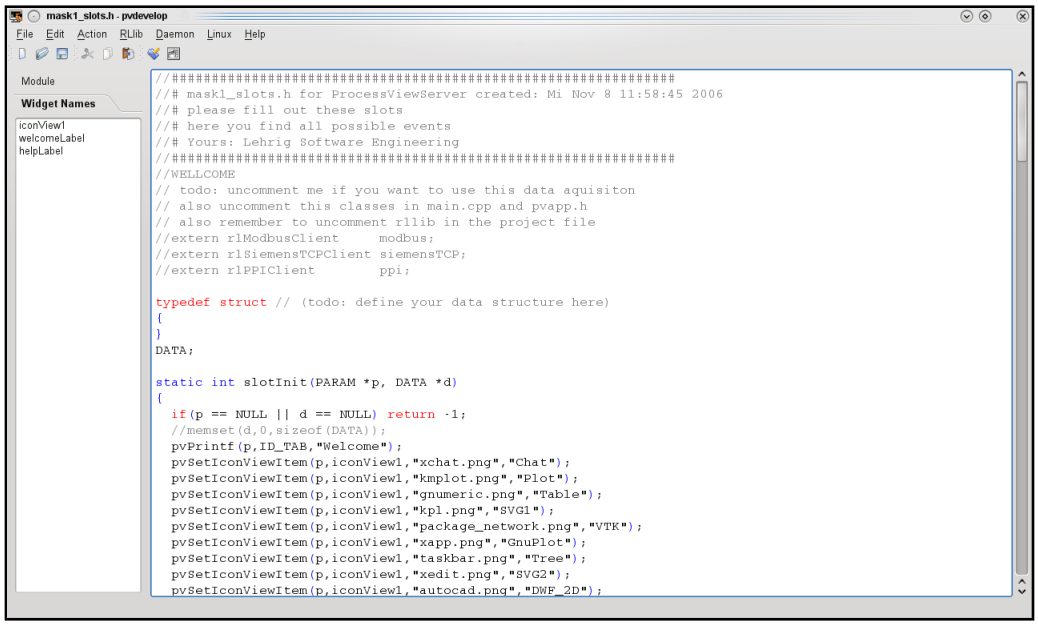

Figure 4.2: main window of pvdevelop in editor mode with selected toolbox

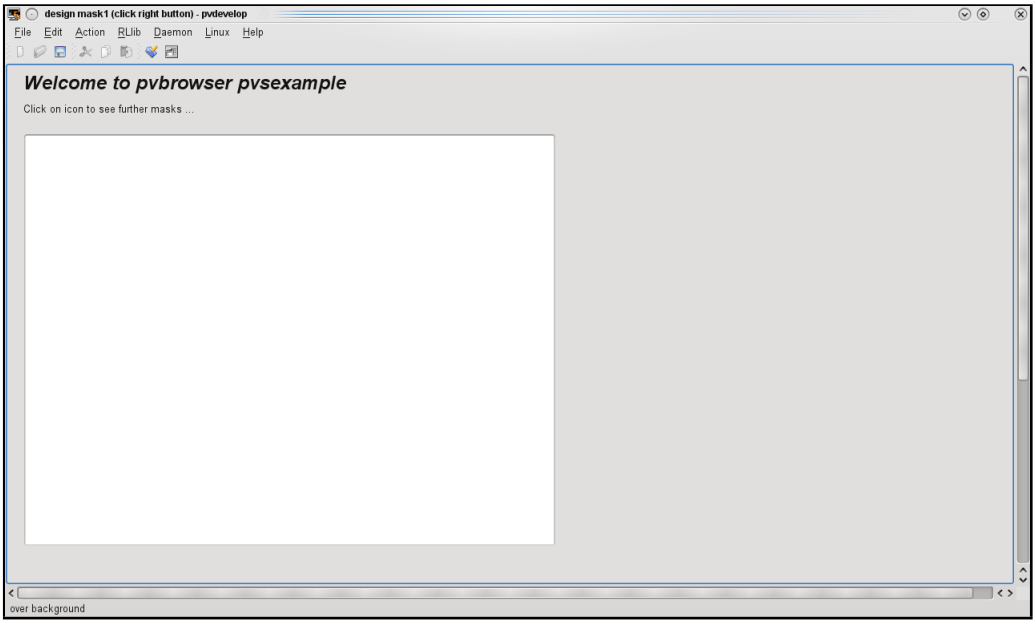

Figure 4.3: main window of pvdevelop in designer mode

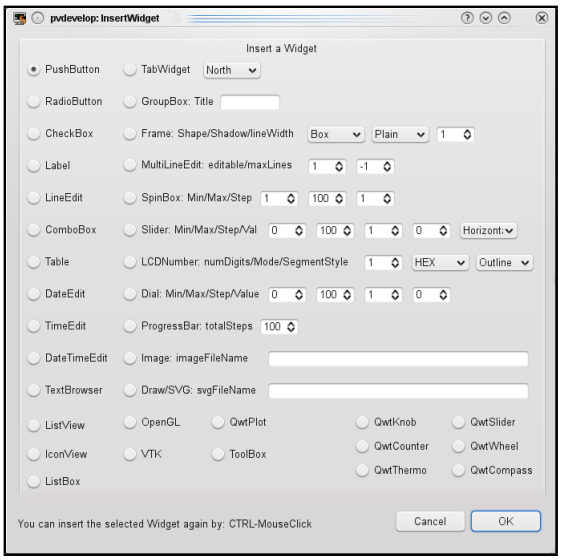

Figure 4.4: dialog box for inserting a widget

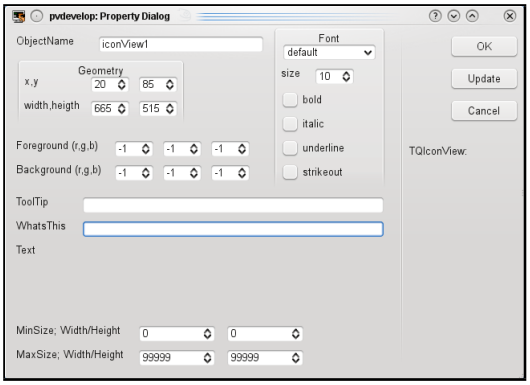

Figure 4.5: dialog box for widget properties

Chapter 5

Using Qt Creator as IDE

Qt Creator is ideal for programming your pvservers because it uses the pro file from qmake also.

Some usefull shell scripts You can create some shell scripts within the directory '~/bin' which will support your work.
The first shell script creates a new project and starts Qt Creator.

~/bin/pvnew

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

The second shell script imports a mask from Qt Designer 'maskN.ui' into the pvserver.

~/bin/pvimport

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

The third shell script will add a mask to an existing project.

~/bin/pvinsert

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

Now create a new directory and input the command 'pvnew'. A new pvserver will be created and Qt Creator will be started with that project.

Create a new pvserver and start Qt Creator

```
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 you must accept the import of the given settings. In 'Projects-Run Settings-Show Details' please select 'execute in terminal' because the pvserver will make outputs to the console which otherwise could not be watched. But these outputs are very helpfull in debugging. On Windows the steps are very similar but the shell scripts must be exchanged by according batch files.

Plugins for Qt Designer In order to use the special widgets from pvbrowser within Qt Designer you must copy the plugins to the Qt Designer plugin directory.

This can be done with the following commands which are a little different for 32 and 64 bit linux systems.

Copy plugins

```
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 has to be copied into the Qt Designer plugin
    directory.
```

Help Files for Qt Creator On `http://pvbrowser.org/pvbrowser/download.php?file=pvslib.qch` and `http://pvbrowser.org/pvbrowser/download.php?file=rllib.qch` you can find help files for our libraries that can be used in Qt Creator.

Using pvdevelop as external tool from within Qt Creator Within Qt Creator you can define pvdevelop as external tool. See the following screenshots for the parameters you should use.

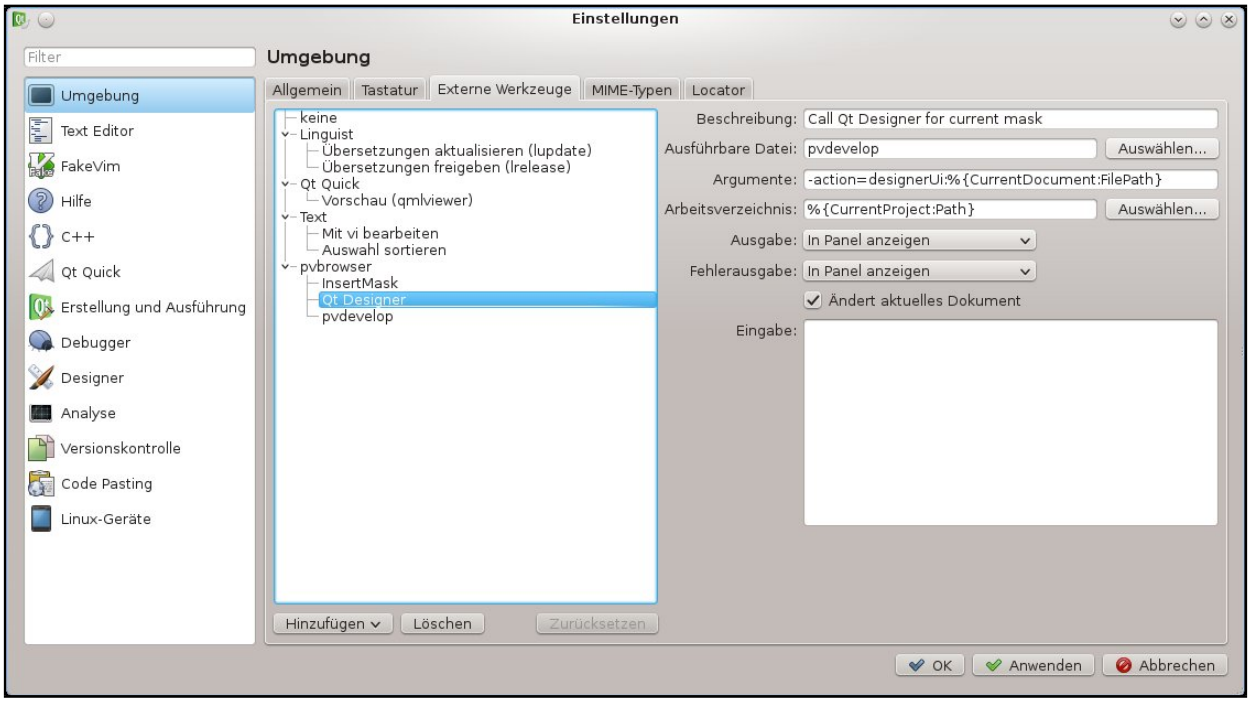

Figure 5.1: Calling Qt Designer. Within Qt Creator you open the according maskX.cpp or maskX_slots.h and select this menu. Then you can design the mask within Qt Designer. pvdevelop will export/import the definition of the mask to/from maskX.ui file.

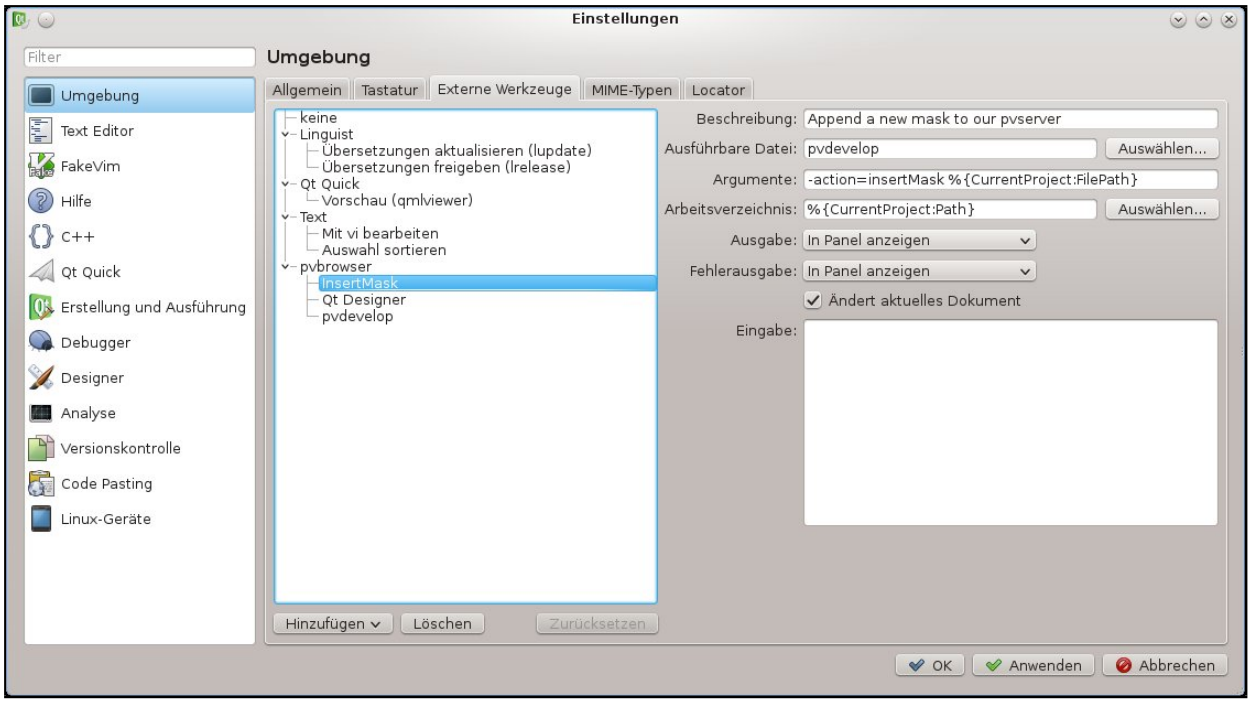

Figure 5.2: You can add a new mask to your project when you have defined an external command which calls pvdevelop with the above parameters

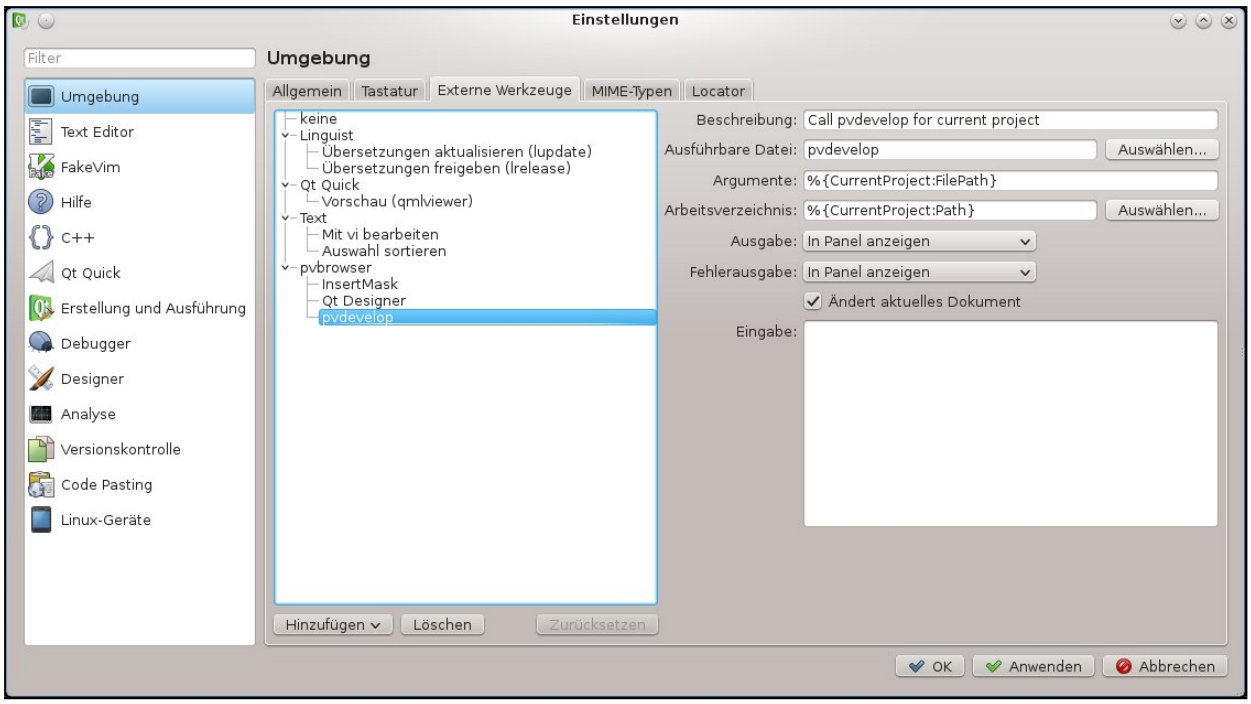

Figure 5.3: You may also want to define an external command which calls pvdevelop.

You can't input the layout in Qt Designer. This must be done within the editor or pvdevelop.

Chapter 6

Programming

If you choose 'File-new pvserver' in pvdevelop you are prompted for the name and the location for your pvserver. Then the integrated designer will become visible. It is used by the popup menu available with a right mouse click. Now you can input the layout within the designer. After you are finished you have to save the layout and switch back to editor mode.
The skeleton for the new pvserver will be generated. Please have a look at the sourcecode from within pvdevelop in order to understand it's structure.
After these initial steps you can start the visualization. 'Action-start server' and 'Action-start browser' are the according commands.
Your task is now to define the data structure DATA and to code the mask∗_slots.h in order to define the logic of your pvserver.

6.1 Structure of a pvserver

There is a project file in which the sources and the libraries of the pvserver are defined.
There is one main file. In this file you can find two different versions of main() one for INETD and one for the MULTI THREADED pvserver which are selected by a #ifdef preprocessor symbol. By standard the MULTI THREADED server is chosen. If you want to start your pvserver by the superserver xinetd you must uncomment USE_INETD within the project file and select 'libpvsid' instead of 'libpvsmt' of the pvbrowser library.
pvMain is the main function for serving a client. In this subroutine you see a loop in which the available masks are called. The return value of a mask determines which mask to show next.
There is one header file which is included in all the sourcecode of a pvserver.
There exist as much masks as you have designed. The sourcecode is generated automatically by pvdevelop. Normally you do not have to care about the skeleton.
There exist as much mask_slots as you have designed. This header file is included within the according mask. There the local data structure DATA and the slot functions are defined.

6.1.1 Project file for qmake

From the project file qmake from the Qt SDK will generate a Makefile. Within that project file all sources and libraries your pvserver is consisting of are specified. Please notice that the project file automatically selects the libraries for the operating system you are using. If you want to use additional libraries within your pvserver you can specify that within this project file also.

projectfile

```
#######################################################################
# 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 function

The main() function is different if you want to use the MULTI THEADED or the INETD version. This is selected by the preprocessor symbol USE_INETD .
If a new client connects to the pvserver main() will start a new thread that will call pvMain() in order to serve

the client. In case of USE_INETD pvMain() will be called directly.
In pvMain() you can first evaluate the URL the client is using. Then pvMain() will run a endless loop in which the different masks which are defined by you are called. The return value from a mask determines which mask to call next. If the the client terminates the connection the thread with pvMain() will be terminated automatically. It is possible to set a pointer to a cleanup() function within the PARAM structure if it is necessary to release resources when the pvserver is terminated. Normally this should not be necessary.
The function getParams() shown in the sourcecode was inserted by the developer of pvsexample and has not been generated. But the rest of the skeleton has been completely generated by pvdevelop.

main function

```
//***************************************************************************
//                        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 masks

From the endless loop within pvMain() the individual masks which implement your visualization are called. We will now show the start mask from pvsexample.
In show_mask1() there is an event loop. The events are parsed and the according 'slot functions' in which you have to insert your code will be called. This file you will never have to change by yourself. This task will be done by pvdevelop.
In generated_defineMask() commands for the widgets you have designed are send to the pvbrowser client. pvbrowser will parse the commands and call the appropriate constructors for these widgets.
All 'pv-functions' use the structure PARAM as the first parameter. Within one element of this structure there is the socket used for communicating with the client. The 'pv-functions' send ASCII text to the client which will interpret the text and call a method from the Qt library.
The enum at the beginning of the file lists the widget names which have been designed by you. These names are used as id within the 'pv-functions' and address the widget within the client. For this there is an array with pointers to widgets within the client. The enum is a index to this array.
The 'slot functions' are included with #include into the mask.

sourcecode of the completely generated part of a mask

```
////////////////////////////////////////////////////////////////////////////
//
// 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 functions

The 'slot functions' must be coded by the developer of the visualization. At the beginning of the file there is the definition of a structure DATA. Within this structure the developer can define his own values that are private to a mask. This can be compared to a C++ class although this is expressed in ANSI C at this place.

The slot functions that must be coded by the developer of the visualization

```
//#################################################################
//# 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 file

The header file pvapp.h is included in every mask. There you can define something for your task. The variable trace can switch on/off the printf() outputs of the pvserver. During development these outputs are usefull and should be switched on. If you use INETD you must switch off these outputs because they would be send to the pvbrowser client and could disturb it.

header file pvapp.h which is included in all masks

```
//***************************************************************************
//                        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 structure PARAM

The structure PARAM is used as first parameter by all 'pv-functions' . Please have a look at it and see which information can be found in there.

The structure PARAM describes the connection to the 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 programming

In the previous sections you have already seen the 'slot functions'. The skeleton has been generated by pvdevelop. Now it is your task to code the 'slot functions' in order to define the logic of your visualization.
slotInit() is responsible for initializing the variables within the DATA structure.
slotNullEvent() is called cyclically in the interval of (PARAM *) p-sleep milliseconds. You can insert continuous updates of the pvbrowser window in there.
The other 'slot functions' are called when the user triggers an event in pvbrowser. For example clicking a button.

Example for a slot function

```
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 functions

The 'Util functions' are used for controlling the pvbrowser window from pvserver. These functions will send ASCII text to the pvbrowser client which will be interpreted there and result in a call to a Qt method. In the mask∗.cpp files which have been generated by pvdevelop the constructors for the widgets you have designed are called in 'generated_defineMask()'. The name of the 'Util functions' starts with 'pv'. The first parameter of the 'Util functions' is (PARAM ∗p) which describes the connection to the pvbrowser client. The second parameter of most of the 'Util functions' is a 'id' which addresses the according widget. The id is one of the names which are listed in the enum at the beginning of the files of mask∗.cpp and have been created from your design.
The reference manual of the 'Util functions' can be found on our homepage and within the help manual within pvdevelop. Please look especially in the topic 'Construction' because there you can find all functions that are available for an individual widget.

6.4 rllib

The rllib allows the platform independent programming of a lot of system functions on the server side. There you also find classes for supporting numerous PLC and fieldbus protocols. In order to use rllib within your pvserver please select the menu 'RLlib-Uncomment rllib' in pvdevelop.

- *rl3964R* implements the Siemens 3964R (Dust) protocol
- *rlbussignaldatabase* class for reading/writing process variables in a MySQL database.
- *rlController* This class implements closed loop control according to 'F. Doerrscheid/W. Latzel, Grundlagen der Regelungstechnik, B.G. Teubner Stuttgart, Regelalgorithmen mit der Trapezregel'
- *rlCorbaClient* used for encapsulating corba communication.
- *rlCorbaServer* used for encapsulating corba communication.
- *rlcutil* contains some simple ANSI C functions.
- *rlDataProvider* container for process variables.
- *rlDataProviderClient* client for accessing process variables that are provided over the network by a server using *rlDataProviderThreads*.
- *rlDataProviderThreads* Provides *rlDataProvider* containers over the network. For this a new thread is started which handles an unlimited number of *rlDataProviderClient*.
- *rlDataAcquisition* is used for data acquisition according to the 'pvbrowser principle'. From a pvserver or even other applications you can access the shared memory and the mailbox used for exchanging data with the daemon that connects to PLC or fieldbus systems. This class encodes the process variables as readable ASCII text. Please also see *rlDataAquisitionProvider*.
- *rlDataAcquisitionProvider* This class is used on side of the daemon for writing process variables into the shared memory and for waiting for output commands over the mailbox. Please also see *rlDataAquisition*
- *rlEIBnetIp* implements the EIBnetIp (European Installation Bus) over network.
- *rlEventLogServer* is used for implementing a server that reads event log messages from other processes over the network. Such a pvserver is already available in 'pcontrol' (see pvbaddon).
- *rlEventLogServerThreads* is used for implementing a server that reads event log messages from other processes over the network. Such a pvserver is already available in 'pcontrol' (see pvbaddon).
- *rlFifo* first in first out buffer for communication between different threads.
- *rlFileLoad* loads an ASCII file which can be iterated within RAM.
- *rlHilscherCIF* Wrapper for drivers for Hilscher CIF cards (Profibus, CAN, ...).
- *rlHistoryLogger* is used for logging history in ASCII files and within RAM.

- *rlHistoryReader* is used for reading a history that has been logged with *rlHistoryLogger*.
- *rlIniFile* implements INI files as known from Windows. You can read, write and manipulate these files. Furthermore you can use the method i18n for translating your text to different languages.
- *rlInterpreter* is used for implementing an interpreter for command lines.
- *rlIpAdr* IP address for use with *rlUdpSocket*
- *rlMailbox* implements a mailbox that is a mechanism for communication between processes on the same computer where several processes are allowed to write and only one process is allowed to read.
- *rlModbus* implements the modbus protocol. Modbus RTU, Modbus ASCII over serial line and Modbus TCP are supported.
- *rlModbusclient* Class for use within a pvserver for accessing the shared memory and the mailbox with a modbusdaemon that has been generated from within pvdevelop. In this case the process variables are encoded binary.
- *rlMutex* implements a mutex.
- *rlOpcXmlDa* implements a client for the OPC XML/DA protocol (HTTP/SOAP/XML). In pvbaddon you can find a daemon that is based on this class.
- *rlPcontrol* Provides a class for starting and controlling processes.
- *rlPlcMem* Implements variables for a soft PLC.
- *rlPlcState* Implements arrays of variables for a soft PLC.
- *rlPPIClient* Implements the Siemens PPI protocol using libnodave.
- *rlSerial* Class for serial lines (RS232/RS485)
- *rlSharedMemory* Implements a shared memory (RAM that is shared between several processes).
- *rlSiemensTCP* Implements the Siemens PLC protocol for S7 series and S5.
- *rlSiemensTCPClient* Class for accessing the shared memory and the mailbox from within a pvserver that is provided by a siemensdaemon which has been generated from within pvdevelop. The data is encoded binary.
- *rlSocket* Socket for communicating over TCP using IPv4 and IPv6.
- *rlSpawn* Starts another process and connects it's STDIN and STDOUT over a pipe with this class. This can be used to control command line orientated applications from a graphical user interface. ATTENTION: This is only available on unix like operating systems.
- *rlSpreadsheetCell* Cell of a table.
- *rlSpreadsheetRow* Row of a table.
- *rlSpreadsheetTable* Table. Tables can read and write CSV files.
- *rlSpreadsheetWorkbook* Several tables. Workbook can read and write a series of CSV files.
- *rlString* Simple string class.
- *rlSvgAnimator* Class for animating SVG graphics from within a pvserver.
- *rlSvgCat* Class for 'normalization' of SVG respective XML code. The individual Tags are split to separate lines and written left justified to STDOUT.
- *rlSvgPosition* Used together with *rlSvgAnimator*. It represents the position of graphical objects within the SVG graphic.
- *rlThread* Wrapper for threads based on pthread respective Windows thread routines.

- *rlTime* implements time and date.
- *rlUdpSocket* Network communication with UDP.
- *rlWebcam* Implements a client for Motion JPEG Webcams that are connected over http.
- *rlwthread* C functions for encapsulating threads.

The reference manual of rllib can be found on our homepage and within the help manual in pvdevelop.

6.5 Lua

Lua `http://www.lua.org/` is a scripting language for embedding in C/C++ host programs. In our pvserver a new thread is created for each new client that connects. Handling the client starts in function 'pvMain()'. When using Lua we call the Lua script 'main.lua' with the function 'luaMain()' from there. The complete handling of the client can now be coded in Lua. Our libraries are wrapped by Swig `http://swig.org/` for using them from Lua.
An advantage of Lua is the small size of the library which implements the language. This allows embedding the library statically into a host program. The end user does not need to install Lua at all. You can develop our pvservers without installing additional packages. You will neither need a C/C++ compiler nor the Qt SDK.
A further advantage of this solution is that you can modify the code while the server is running. Each new client will see the current Lua code without the need to restart the server. This can be compared to the usage of PHP within web servers.
When creating a new project from within pvdevelop you must choose the programming language. If you select 'Lua' pvdevelop will create the whole framework of the Lua script. The part of the script that is dedicated to the design of the mask is completely generated by pvdevelop. The 'slotFunctions' are also initially generated. The programmer of the visualization can concentrate on coding the 'slotFunctions'.
In pvbaddon directory 'pvbaddon/demos/lua/' you will find examples in Lua. Here we show a 'Hello World' pvserver in Lua which shows the usage of Modbus and a SQL Database system.

6.5.1 main.lua

The Lua main program

```
-------------------------------------------------------------------------------
-- 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
```

The Lua main program 'luaMain()' is called by our C/C++ main program 'pvslua' when a new client connects with 'pvslua'. In 'luaMain()' the handling of the client starts.
As you can see from the code we call functions from our C/C++ libraries from Lua. In 'main.lua' we define a global class for data acquisition with our shared memory and mailbox and a class for a MySQL database. The database class is based on the Qt database classes. Thus a wide range of database types is supported.

Supported databases

```
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
```

After this there is a loop where all the masks of our visualization are called. Currently there is only 1 mask in there.

6.5.2 maskN.lua

Code for a mask of the visualization

```
-------------------------------------------------------------------------------
-- 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
```

The code for the masks of the visualization is created by pvdevelop. Within the contained event loop the 'slotFunctions' are called. These 'slotFunctions' must be coded by the programmer of the visualization.

6.5.3 maskN_slots.lua

slotFunctions for a mask of the visualization

```
-------------------------------------------------------------------------------------
-- 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
```

Within the 'slotFunctions' the programmer of the visualization codes the logic of the pvserver.
The example shows the connection to Modbus with our shared memory and mailbox. Furthermore we see a database query. The comments in the code should be sufficient to understand the code.

6.5.4 modbus.ini

INI file for 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)
```

The daemon 'modbus_client' uses a INI file which defines what should be read from Modbus, what name the shared memory and mailbox have and which interface is to use.

Start modbus_client

```
modbus_client modbus.ini
```

6.6 Python

For Python there is a language binding to the 'Util functions' and to rllib which is generated with Swig `http://swig.org`. Thus Python developers can use these libraries.
In pvdevelop you can choose to use Python when creating a new pvserver. Then pvdevelop will generate a skeleton in C++ that calls 'slot functions' written in Python. The developer can code in Python and use the above libraries.

slot functions in Python

```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 the following sections we will show the programming of the individual widgets. The following 'Util functions' are available for all kinds of 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);*

The other 'Util functions' refer to individual widgets only. These you can find under 'Construction' within the help on pvslib below each widget.

6.7.1 PushButton

PushButton can have a optional icon.

Figure 6.1: PushButton

They deliver events in the following 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

Several RadioButtons work exclusive if they share the same parent widget.

Figure 6.2: RadioButton

They deliver events in the following 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

CheckBoxes can be marked checked.

Figure 6.3: CheckBox

They deliver events in the following 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 are used for one line of text output.

Figure 6.4: Label

They deliver events in the following 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 are used to input one line of text.

Figure 6.5: LineEdit

They deliver events in the following 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 are used for input of several lines of text. A MultiLineEdit can also be set 'read only'.

Figure 6.6: MultiLineEdit

They deliver events in the following 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

ComboBoxes are used to select one option from several. ComboBoxes can be editable.

Figure 6.7: ComboBox

They deliver events in the following 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

A LCDNumber can show values as LCD display.

Figure 6.8: LCDNumber

They deliver events in the following 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 are used for input in a analog fashion.

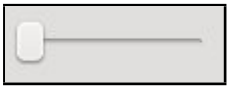

Figure 6.9: Slider

They deliver events in the following 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

A Frame can be used to surround and group other widgets.

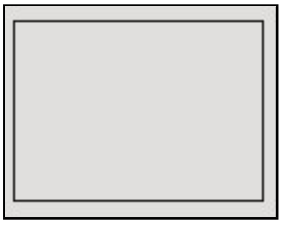

Figure 6.10: Frame

They deliver events in the following 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

A GroupBox groups several widgets and provides a title.

Figure 6.11: GroupBox

They deliver events in the following 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

A ToolBox provides several tabs with which the user can switch between several areas. In each area widgets can be paced.

Figure 6.12: ToolBox

They deliver events in the following 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

A TabWidget is used to organize several sub windows which the user can select.

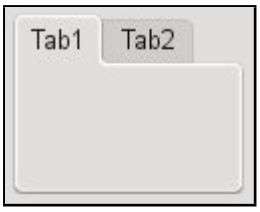

Figure 6.13: TabWidget

They deliver events in the following 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

A ListBox provides a selection of several entries.

Figure 6.14: ListBox

They deliver events in the following 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

A Table is used for input and output of tabular data.

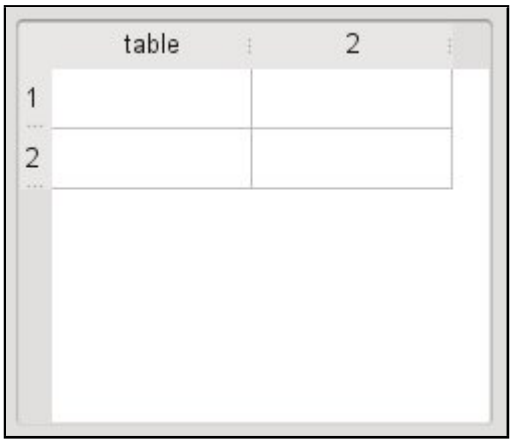

Figure 6.15: Table

They deliver events in the following 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

A SpinBox is used for input and output of values which are limited to a minimum and maximum value.

Figure 6.16: SpinBox

They deliver events in the following 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

A Dial is used for input and output of values.

Figure 6.17: Dial

They deliver events in the following 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

A Line is used for graphical organization.

Figure 6.18: Line

They deliver events in the following 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

A ProgressBar can be used for longer lasting actions.

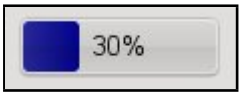

Figure 6.19: ProgressBar

They deliver events in the following 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

A ListView implements a tree like view.

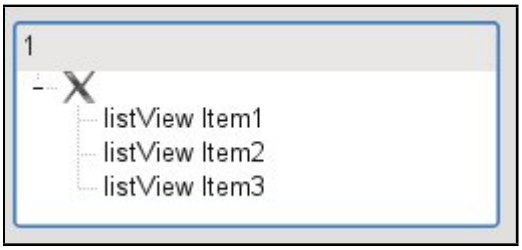

Figure 6.20: ListView

They deliver events in the following 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

A IconView lets you choose using icons.

Figure 6.21: IconView

They deliver events in the following 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 allows the integration of any html file or webpage using a http URL with the use of WebKit. Clicking a hyperlinks results in a TextEvent with that URL.

Figure 6.22: TextBrowser/WebKit

They deliver events in the following 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

DateTimeEdit is used for input of date and time.

Figure 6.23: DateTimeEdit

They deliver events in the following 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

DateEdit is used for input of date.

Figure 6.24: DateEdit

They deliver events in the following 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

TimeEdit is used for input of time.

Figure 6.25: TimeEdit

They deliver events in the following 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

A QwtThermo can be used as analog display.

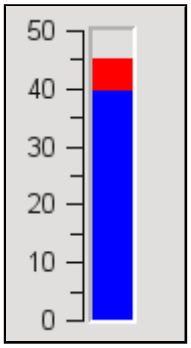

Figure 6.26: QwtThermo

They deliver events in the following 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)*

The Qwt widgets deliver float values. In order to use float values you must extend the event loop of the mask.

```
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;
    // In slotSliderEvent you would only get the integer part of the value.
  if((ret=slotTextEvent(p,i,&d,text)) != 0) return ret;
    // additionally deliver event to slotTextEvent and
    // read the float value in there.
    // sscanf(text,"(%f)",&fval);
  break;
```

6.7.27 QwtKnob

A QwtKnob is used for input and output values over a rotary knob.

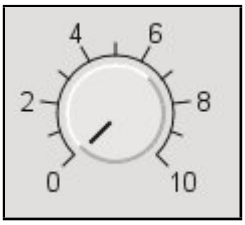

Figure 6.27: QwtKnob

They deliver events in the following 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

A QwtCounter is used for input and output limited to a minimum and maximum value.

Figure 6.28: QwtCounter

They deliver events in the following 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

A QwtWheel is used for input of values.

Figure 6.29: QwtWheel

They deliver events in the following 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

A QwtSlider is used for input and output of values limited to a minimum and maximum value.

Figure 6.30: QwtSlider

They deliver events in the following 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

A QwtDial is used for display of analog values.

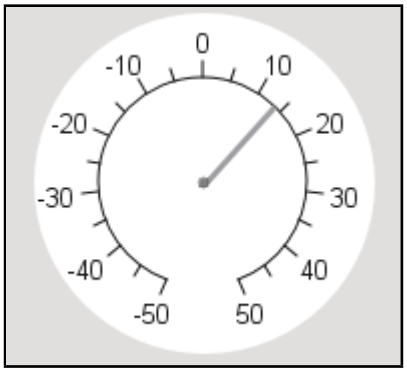

Figure 6.31: QwtDial

They deliver events in the following 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

A QwtAnalogClock is used for display of analog time.

Figure 6.32: QwtAnalogClock

They deliver events in the following 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

A QwtCompass is used for input and output of values with a compass.

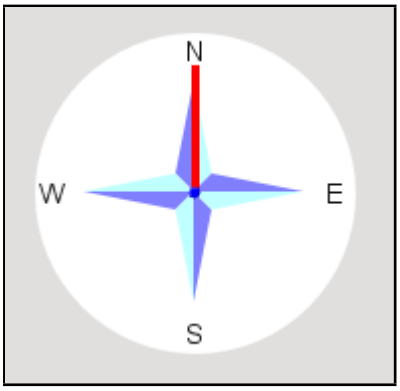

Figure 6.33: QwtCompass

They deliver events in the following 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 within pvbrowser

pvbrowser provides a rich set of standard widgets. But it is possible to add your own custom widgets as well. These custom widgets are defined within a shared object library. On Unix like systems the shared object library is named libNAME.so. On OS-X it is named libNAME.dylib. On Windows it is named NAME.dll.

An example of such custom widgets can be found within pvbaddon.tar.gz

directory: pvbaddon/custom_widgets/embedded_widgets/embedded_widgets_1

Figure 6.34: Screenshot of ewidgets

How users can get and install custom widget plugins

The above custom widgets can be downloaded in binary form from the download section of our web page. Please store them under the following directory on your computer / mobile device.

Directory where widget plugins are stored

```
Unix like systems: /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 # You must place the dll at the correct location.
                                         # The sys folder is hidden and/or protected.
```

The above path is defined within the options of pvbrowser.
pvb_widget_plugindir=/opt/pvbrowser
Thus it is possible to change the plugindir to a different location.
For installation a user simply downloads the binary of the widget plugin and stores it under the above directory.
When you call a pvserver that uses custom widgets not installed yet pvbrowser will show a message.

How widget plugins are used

Within the designer of pvdevelop you select "Custom Widget" when inserting the widget and set the whatsThis property to "/libname/widgettype:arguments".

In case of the above ewidgets plugin the following combinations are possible

```
"/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"
```

Notice that the libname does not include the leading "lib" nor the trailing extensions ".so ||.*dylib*||.*dll*".
The widgettypes supported by the plugin will be known from the documentation of the widget plugin (see above).
In the above examples we do not use meaningfull arguments. Ewidgets do not use these arguments but other custom widget plugins might do.
The standard functions like for example pvSetGeometry() can be applied to all custom widgets. Functions special to a custom widget might be done with pvSetText(). The custom widget would use the convention "if text starts with @" then interpret the following text as a special command to this custom widget.
An example pvserver that uses the ewidgets custom widget plugin can be found in
"pvbaddon/custom_widgets/embedded_widgets/pvsEmbeddedWidgetsSample"

How to develop your own widget plugins

First you will create widgets as described in the qt documentation and test them from a normal library. Then you will need some glue code to make the widgets loadable as a plugin.
See:
pvbaddon/custom_widgets/embedded_widgets/embedded_widgets_1/mywidgets.h
pvbaddon/custom_widgets/embedded_widgets/embedded_widgets_1/mywidgets.cpp
For example the "myQtMultiSlider" plugin widget would be derived from your normal "QtMultiSlider" widget as follows:

Class definition of a custom widget

```
#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;
};
```

We have added the protected "int *sock" and "int ident" which correspond to the socket used in pvbrowser and the widget id used in pvbrowser. Then we have a "virtual bool event(QEvent *event)" and some slot functions.

Definition of pvb events

```
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;
};
```

Whenever a command is send from the pvserver to your custom widget you will receive 2 strings that you can interpret. When we look to the implementation of "myQtMultiSlider" we see how this can be done.

Constructor implementation

```
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)));
}
```

The constructor receives the object name of the widget, the network sock of pvbrowser and the ident (id) used by pvbrowser. Optionally you can evaluate the argument you have defined for your widget within the designer of pvdevelop. In this example the argument is ignored. Then we connect some slotFunctions we want to use.

Slot functions

```
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));
}
```

Within the slot functions we can send the text for the according event to the server. In our example we use "user" events.

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);
  }
}
```

The method event will handle all events.
Either the events are triggered by user interaction or they were send from the pvserver to the pvbrowser. If the event was send from the pvserver it will have type pvbEventNumber. Here we interpret the strings used to represent that event and call the desired function. We only need to handle the commands that are special to our widget. All commands that are common to all widgets derived from QWidget do not need special handling because this is already done within pvbrowser itself. All events that are triggered by user interaction will be handled in the "else" part.
The export functions of the shared library are as follows.

Export functions

```
//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;
}
```

The export functions are of type extern "C".
When pvbrowser loads the widget plugin it will expect to be able to resolve the export functions above.

6.8 Graphics

pvbrowser provides several methods for output of graphics.

First it is possible to output simple bitmap graphics in the common image formats supported by Qt. It is possible to output XY graphics with which measurement curves can be shown for example. SVG graphics can be dynamically animated and you can receive events when the user clicks graphical objects. External plot tools like gnuplot can be easily integrated when they are able to produce bitmap graphics as output. It is even possible to use 3D graphics within pvbrowser. This can be done with OpenGL. Thus it is possible to include Autocad DWF files in pvbrowser for example and be animated. On the other side you can use the Visualization Toolkit VTK `http://vtk.org/` in pvbrowser. There TCL scripts are send to the pvbrowser client that control VTK graphics. Please notice that VTK is not compiled into pvbrowser by standard. If you want to use VTK you must install it, edit the file pvb/pvbrowser/pvbrowser.pro and remove the comment before CONFIG += USE_VTK in there. Finally you have to recompile pvbrowser.

6.8.1 Bitmap graphics

Bitmap graphics can be used in JPG, PNG, GIF, TIFF and BMP format.

Figure 6.35: Bitmap graphic

Inserting an image into an image widget

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

They deliver events in the following 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 graphics

xy graphics can be alternatively created using the Draw or QwtPlot widget.
They deliver events in the following 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)*

Examples for xy graphics using the Draw and 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)
  {
    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;
```

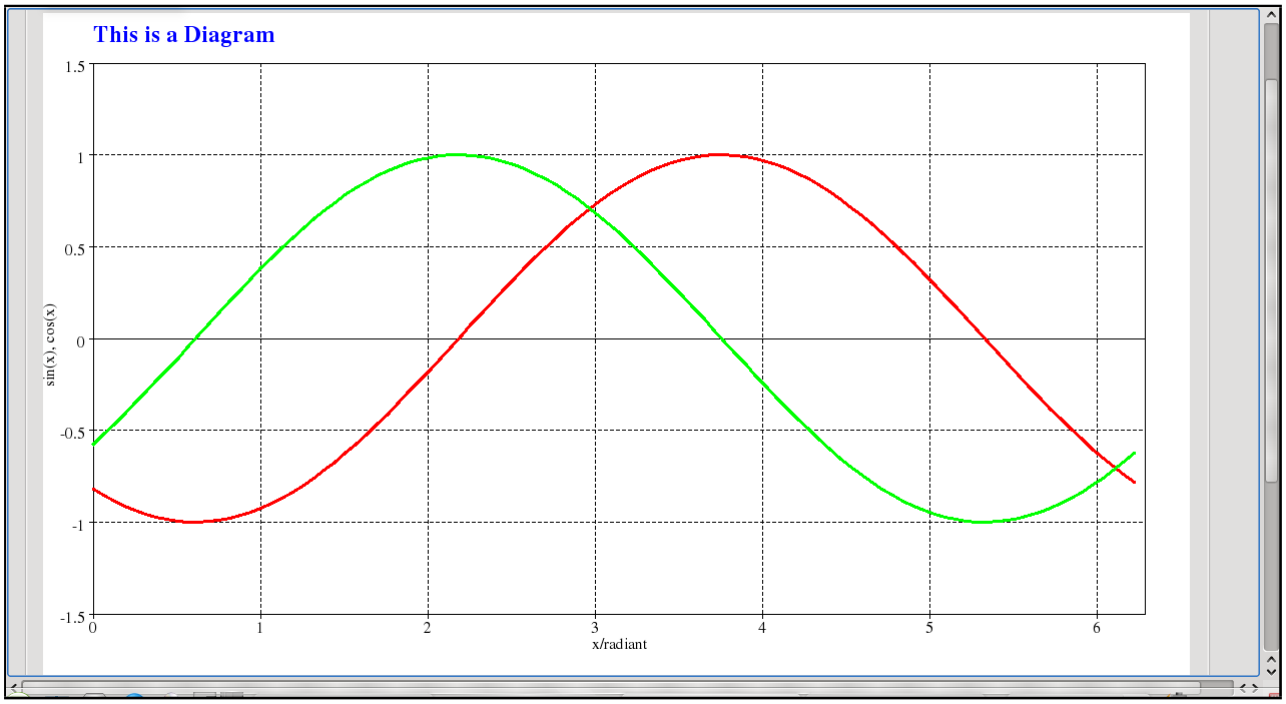

Figure 6.36: xy graphic using the Draw widget

```
  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;
}
```

Within the function 'drawGraphic1' an diagram with the Draw widget is drawn. The functions with 'qpw' at the beginning are calls for the QwtPlot widget.
The *Draw widget* provides some 'g-functions' (starting with the letter 'g') with which any shape can be drawn. Especially there are some functions for drawing axises and curves for diagrams.
The *QwtPlot widget* can draw more elaborate diagrams. Even diagrams scaled logarithmic are possible.
Please review the topics under *Graphics* and *QwtPlotWidget* within the reference of the pvslib.

6.8.3 external plotting tools

External plotting tools can be called from your pvserver. If the plotting tool is capable to create bitmap graphics as output you can easily insert that graphic into pvbrowser. We show this within an example for Gnuplot.

Gnuplot as example for an external plotting tool

```
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
```

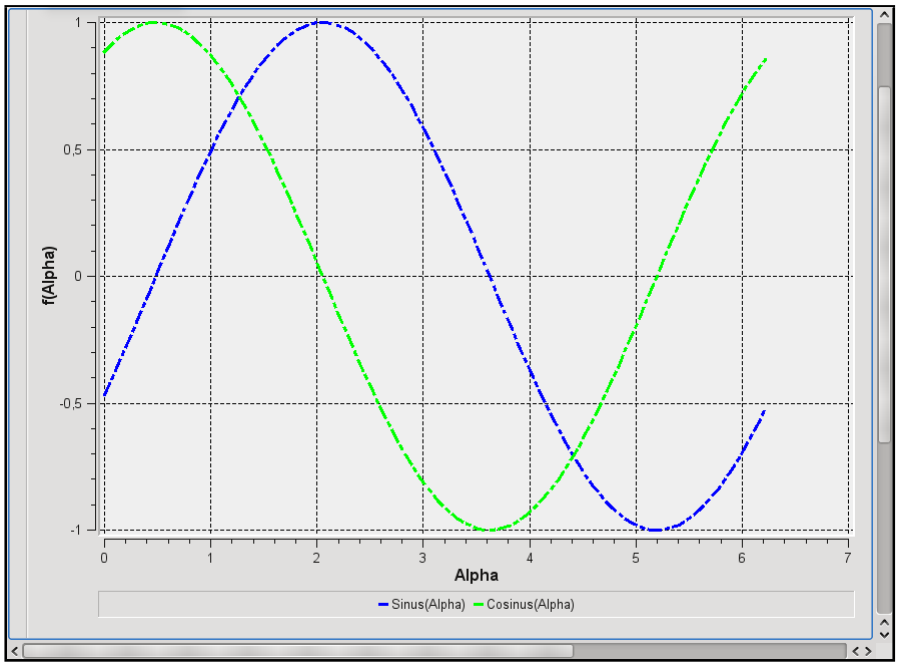

Figure 6.37: xy graphic using the QwtPlot widget

```
#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;
}
```

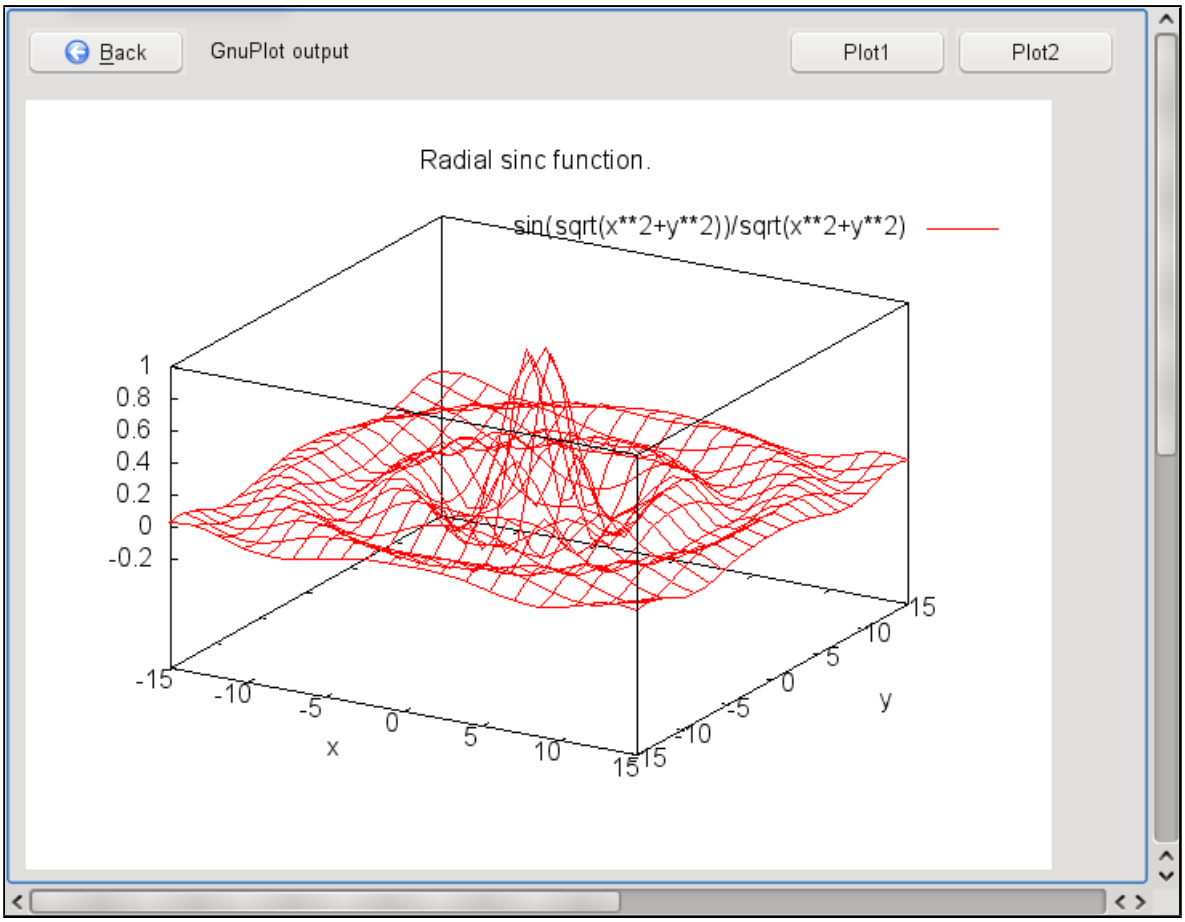

Figure 6.38: Gnuplot output within pvbrowser

6.8.4 SVG graphics

A Scalable Vector Graphic is a XML specification and file format for two dimensional vector graphics that can be static or animated. It is an open standard which is maintained by the W3C SVGWorking Group.
SVG graphics can be created with special drawing tools like inkscape. Alternatively you may use tools for converting CAD formats like DXF into SVG.

SVG example 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>
```

These graphics can be used in pvbrowser. Simply insert a Draw/SVG widget and use rlSvgAnimator from rllib to draw and animate the SVG. Using this class the SVG graphic is send from the server to the pvbrowser client. Once the graphic is in the pvbrowser client rlSvgAnimator is used to modify/animate the graphic. The

server will receive events when you click objects within the SVG graphic. Additionally some mouse events will be send.

Definition of rlSvgAnimator

public part of 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:
```

Loading and drawing a SVG graphic

The following code shows how to load a SVG graphic into the pvbrowser client and draw it.
First an instance of the rlSvgAnimator class is defined in DATA. In 'slotInit' the socket used for communication with the pvbrowser client and the object id is registered in svgAnimator. Using the method 'read' the SVG file is send to the pvbrowser client. With calls to 'pvSetZoomX' and 'pvSetZoomY' pvbrowser is instructed to respect the aspect ratio of the graphic when it is zoomed. The helper function 'drawSVG1' now redraws the SVG graphic after some animation steps have been applied.

loading and drawing a SVG graphic

```
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;
}
```

Animating a SVG graphic

After the SVG graphic is loaded and you are able to redraw it you can animate/modify the graphic by using rlSvgAnimator.
The object name is used to address the graphical objects within the SVG (id='objectname'). Using the SVG drawing tool you can set the id as wanted.

Changing the text of an SVG object

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

Hiding and showing an object

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

Setting object properties

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

Each object can be scaled, rotated and moved using it's SVG transformation matrix. Using rlSvgAnimator you can modify the matrices for each object.

Changing the transformation matrix for an object

```
// 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);
```

Attention:
During the first call of setMatrix() the object 'objectname' will be grouped. Within this operation all children of 'objectname' that have been hidden will be shown again.

Definition of 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);
};
```

After defining an instance of rlSvgPosition the matrix is set to the identity matrix (no movement, no rotation, zoom factor 1.0). By changing rlSvgPosition you can zoom, rotate and move the object. After you have changed rlSvgPosition you must call 'setMatrix' from rlSvgAnimator to apply the transformation to the object.
The method 'svgSearchAndReplace' from rlSvgAnimator can replace parts of properties. This is usefull for example if you want to modify the style properties. The 'before' parameter can include wildcards like *(any characters) ? (any 1 character).

Modify properties

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

Additionally there are methods for recursively applying changes to all parts below a graphical object.

Setting modifying properties of an object recursively

```
  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);
```

After you have modified all objects as desired you must redraw the SVG using 'drawSVG1'.
The method 'read' which sends the SVG graphic to the pvbrowser client can have an optional parameter ini file. The ini file will then be filled with the properties of the SVG graphic. The id's of the graphical object will correspond to the sections within the ini file. Tip: You may use your own properties like your:tag='any value' within the SVG. These properties will be ignored by the SVG renderer. You can use this properties for your own purposes.

Optional INI file parameter when calling read

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

Handling events from SVG graphics

SVG graphics are implemented with the Draw widget. The Draw widget will deliver the following events when it contains a SVG graphic.

- *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)*

Especially the TextEvent is interesting.
If you want an event when the user clicks a graphical object within the SVG graphic the id of the object must start with 'pv.' or 'PV.' .
Example: id='pv.testclick'
Then a 'slotTextEvent' will be triggered. If the name starts with 'PV.' id='PV.textclick' pvbrowser will additionally do a visual feedback to the user by changing the mouse shape to Qt::PointingHandCursor when the user moves the mouse over the object. This might be used to implement 'Buttons' with visual feedback.
slotTextEvent is not only used for this click event but there are more functions that can trigger slotTextEvent. For this we provide some helper functions that will interpret the text.

slotTextEvent when using SVG graphics

```
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;
}
```

The helper function 'svgObjectName' will return the name of the SVG object. The helper function 'textEventType' will detect a normal TextEvent or special events for use with SVG. You will know if the object has been clicked and which mouse button has been used.

The 'case SVG_BOUNDS_ON_ELEMENT:' is a response to the following request.

Requesting the surrounding rectangle for an SVG object

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

The 'case SVG_MATRIX_FOR_ELEMENT:' is a response to the following request.

Requesting matrix for an SVG object

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

Zooming and panning the whole SVG graphic

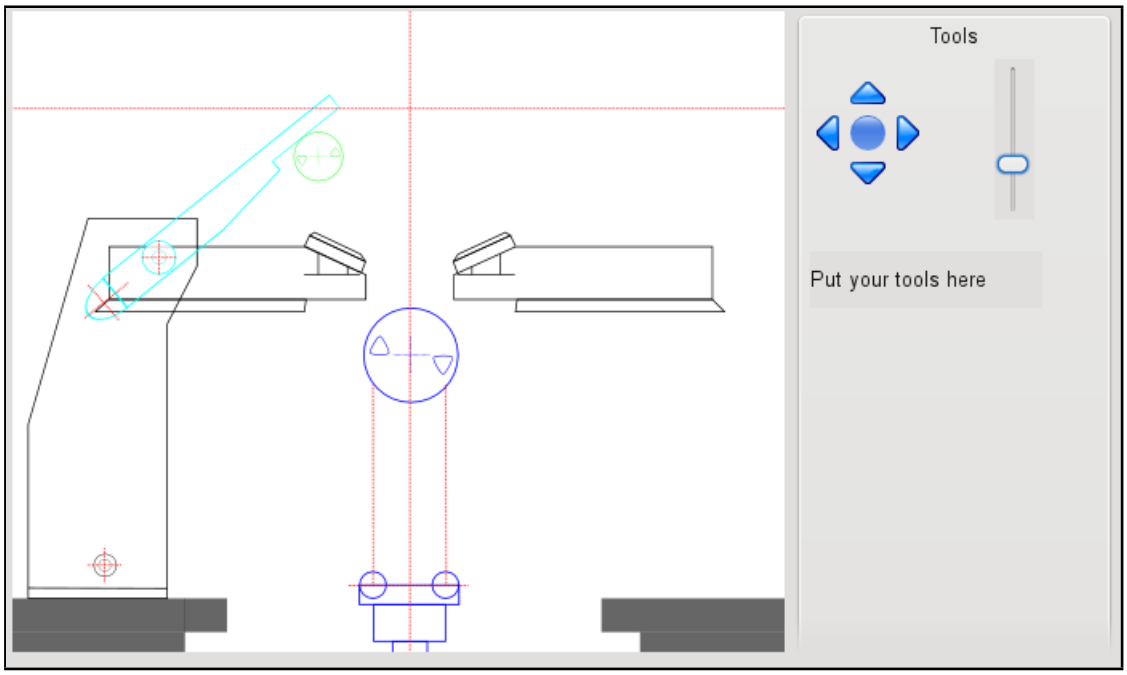

Figure 6.39: Zooming and panning the whole SVG graphic

In directory pvbaddon/template/weblayout/ within pvbaddon you can find a template where zooming and panning the whole SVG graphic is implemented.
First see how the name (id) of the whole SVG graphic is called within the SVG XML code. We suggest to call it 'main' because this is the default name rlSvgAnimator assumes.
The following slot functions are responsible for zooming and panning.

Zooming the whole SVG graphic in percent

```
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;
}
```

Adjusting to the actual window size

```
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;
}
```

Panning the graphic with the mouse

```
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;
}
```

Panning the SVG graphic with 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);
```

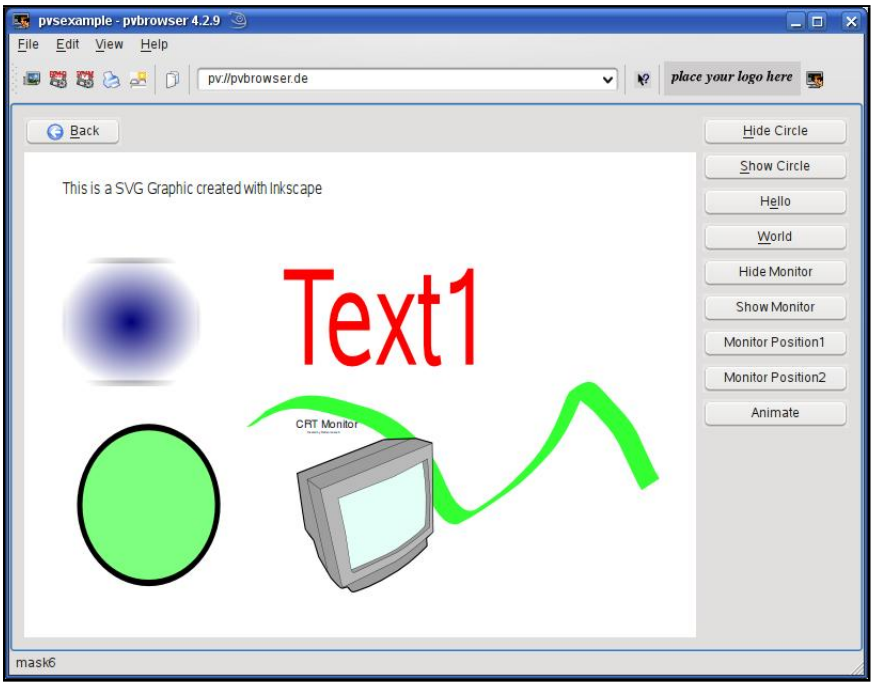

Figure 6.40: Simple SVG

```
  }
  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;
}
```

Examples for SVG graphics

The first 2 examples are from pvsexample which comes with pvbrowser.

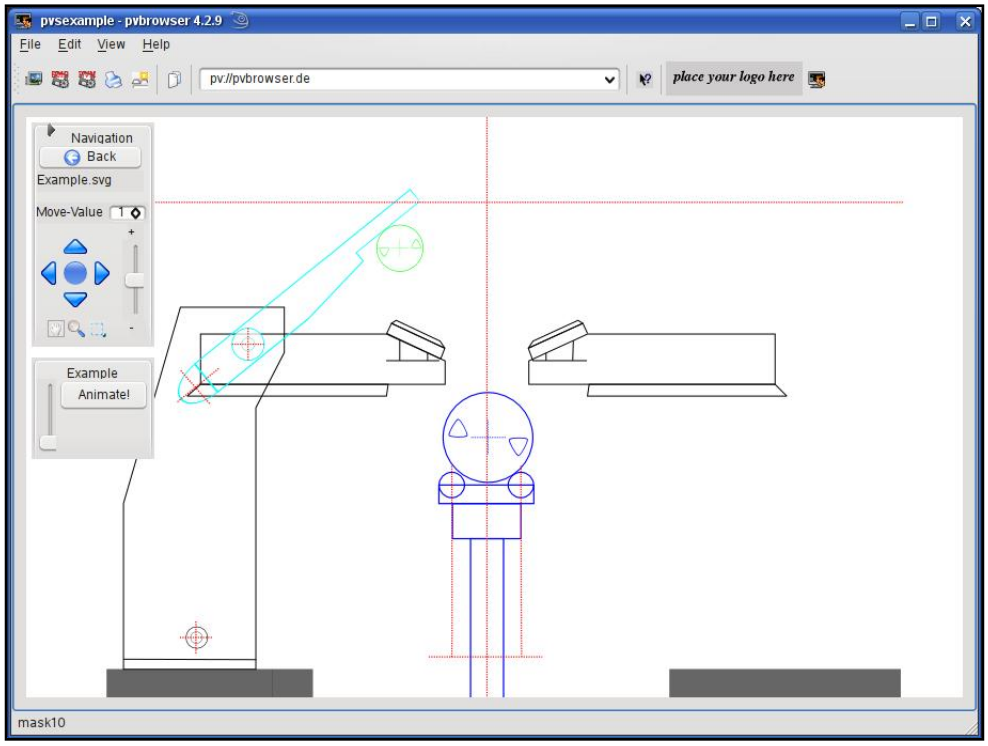

Figure 6.41: A mechanical drawing

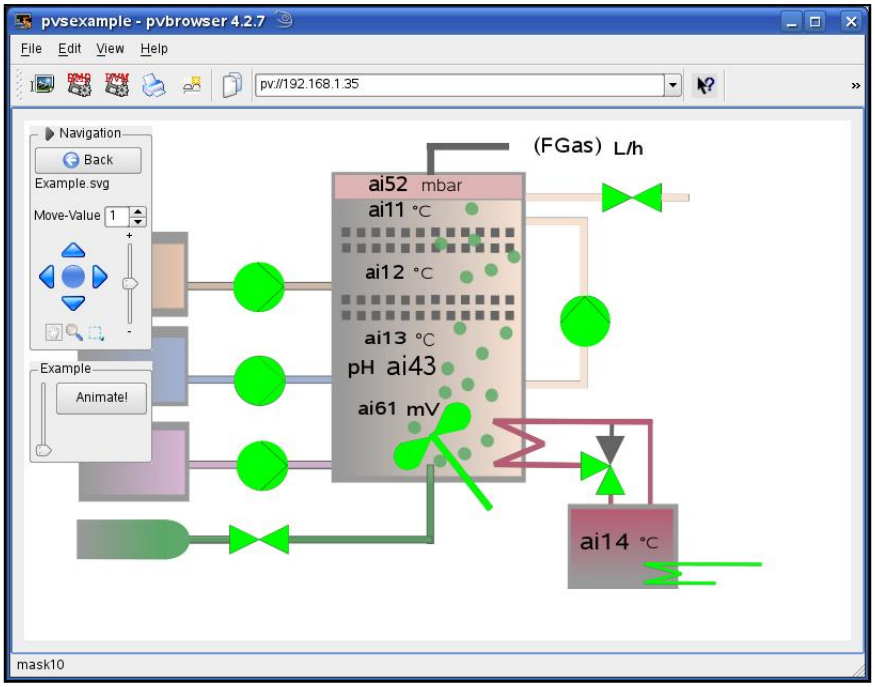

Figure 6.42: A SVG of a process

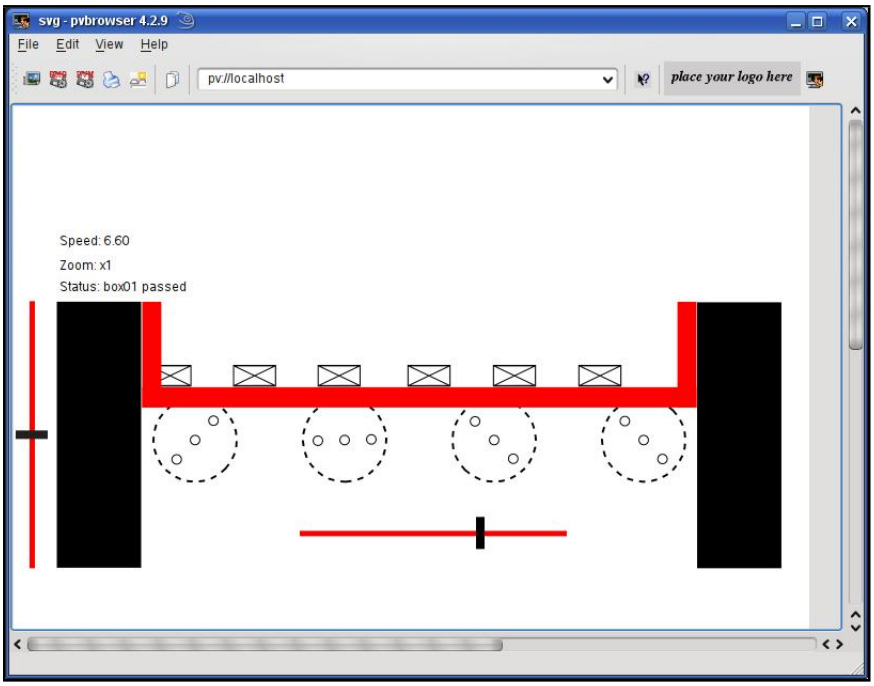

Figure 6.43: A SVG showing moving packages on a conveyor

6.8.5 OpenGL

OpenGL widgets are available in pvbrowser. The pvserver sends OpenGL commands to the pvbrowser client where they are executed. A use of OpenGL might be the embedding of CAD drawings into pvbrowser. For example Autocad uses the DWF file format to exchange CAD drawings with other applications. For this file format there exists a converter which transforms DWF to OpenGL and can then be used in pvbrowser. The generated drawing is send to the pvbrowser client with 'pvSendOpenGL' which will display it. The objects included in the drawing are hold in display lists (listarray). With this display lists graphics can be animated. OpenGL widgets deliver events in the following 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)*

A Autocad DWF file was exported to a OpenGL file and is now shown in pvbrowser and is animated

```
//#################################################################
//# 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 is a powerfull 3D visualization tool which has been written in C++ and is based on OpenGL. VTK can be scripted using Tcl.
In pvbrowser client there is a VTK widget if you compile pvbrowser with VTK support. A pvserver can now send Tcl scripts to such a widget which will be interpreted there. Thus VTK can be embedded into a

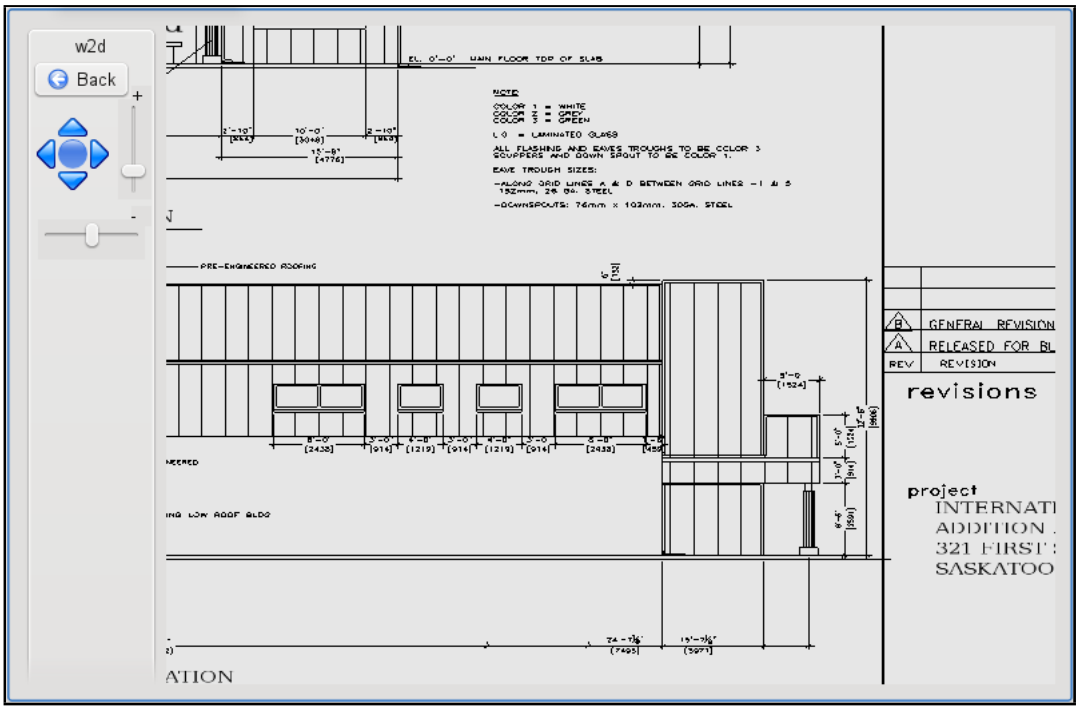

Figure 6.44: Autocad drawing in pvbrowser

visualization without much cpu load on the pvserver. The rendering of 3D scenes is completely done on the client. The pvserver only sends some Tcl commands.
Attention: You must compile and install VTK yourself. Please test if the examples that come with VTK work correctly. For this you must set the environment variables corectly. You can use "wish" (see Tcl) for testing.

Example for visualization of a 2D dataset 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"

    set dims [[reader GetOutput] GetDimensions]
    set dim1 [lindex $dims 0]
    set dim2 [lindex $dims 1]
```

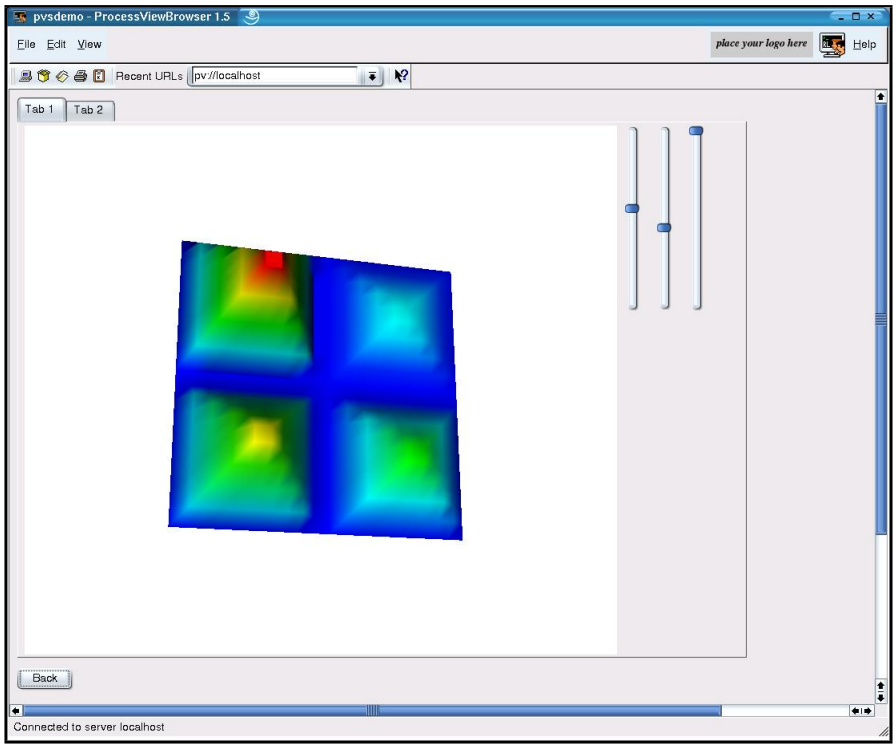

Figure 6.45: Display of a 2D datasset data1.vtk

```
    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 could be a profile measurement for example. You pvserver could let generate data1.vtk and update the screen with the new measurements as they get available.

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
```

The slot functions for 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 Graphics interface for the server

The class rlSvgVdi from rllib provides a virtual device interface for the server which uses SVG. It includes some conveniance methods that work similar to the methods described in chapter "xy graphics" above.
The methods produce valid SVG text that can be collected and/or send the SVG to a destination.

Setting the output destination of rlSvgVdi

```
int setOutput(int *socket_out, int idForPvbrowser=0); // output the SVG to a pvbrowser client
int setOutput(FILE *fout);                            // output the SVG to a FILE stream
int setOutput(const char *outputfilename);            // output the SVG to a file.svg
int setOutput(rlSpawn *pipe);                         // send the SVG over a pipe to a SVG converter
int endOutput();                                      // terminate the output
```

Use SVG converters with a commandline interface to convert SVG to png, jpg, pdf ...
Some SVG converters:
`https://wiki.gnome.org/action/show/Projects/LibRsvg`
`http://imagemagick.org/script/index.php`

6.9 Dialogs

In pvbrowser some simple dialog boxes are available. But you can also create complex dialogs.

6.9.1 MessageBox

Figure 6.46: A MessageBox

Message Boxes can be programmed with the function 'pvMessageBox'. id_return determines under which id the result of the MessageBox will be delivered. You should choose negative values in order not to conflict with the widgets you have designed. The value of the clicked button is delivered in 'slotSliderEvent'. if you want to use less than 3 buttons the remaining values are set to 0 or MessageBoxNoButton.

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

Figure 6.47: A Input Dialog

Input Dialogs can be programmed with the function 'pvInputDialog'. id_return determines under which id the result of the InputDialog will be delivered. You should choose negative values in order not to conflict with the widgets you have designed. The text that has been input by the user is delivered in a 'slotTextEvent'.

pvInputDialog

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

6.9.3 FileDialog

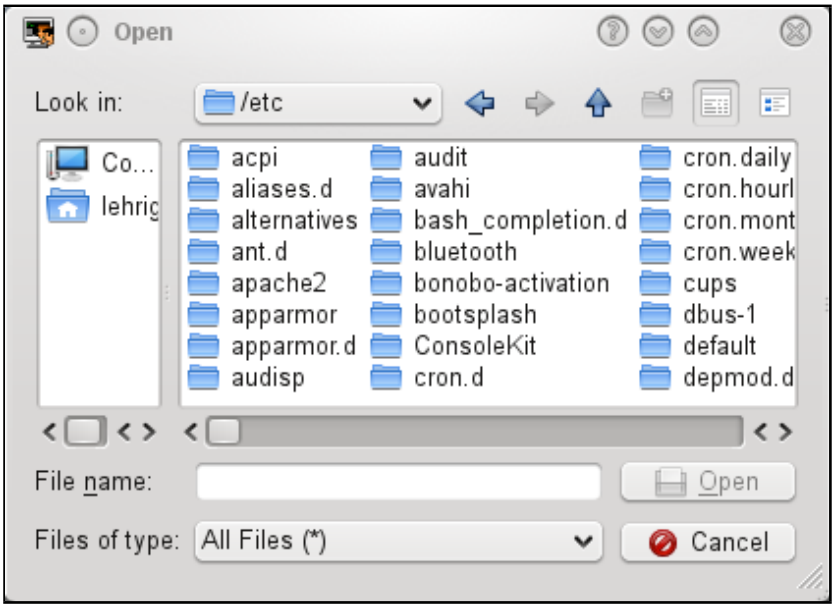

Figure 6.48: A File Dialog

File Dialogs can be programmed with the function 'pvFileDialog'. id_return determines under which id the result of the FileDialog will be delivered. You should choose negative values in order not to conflict with the widgets you have designed. The result will be delivered in a 'slotTextEvent'.

File Dialog

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

6.9.4 ModalDialog

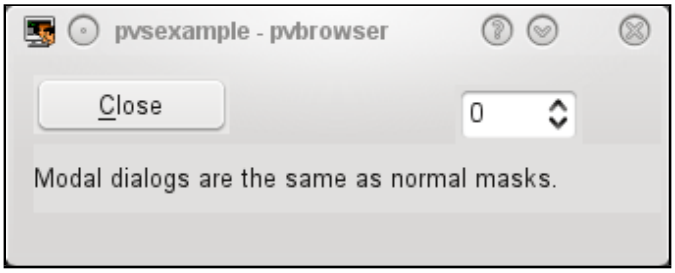

Figure 6.49: A modal dialog

modals dialogs can be designed like normal masks. In the calling mask you use the function 'pvRunModalDialog'. The size of the dialog window is given with width, height. 'showMask' is the show function of the modal dialog. 'userData' is the address of a data structure that can be used to return values from the modal dialog. If you wish to update the base mask from the modal dialog you must supply the 'slotNullEvent' for the 'showData' parameter. Otherwise you can set 'readData' and 'showData' to NULL. The parameter 'd' at the end is needed for 'showData'.

modal 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);
```

Call dialog from a mask

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

In 'slotNullEvent' the modal dialog can update the base window.

update mask of base window from within a modal dialog

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

terminate modal dialog and return values to the calling mask

```
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; // corresponds to 'userData' at creating the dialog
    *ptr = d->input;                     // read return values
                                         // instead of a simple int it could be the address
                                         // of a data structure with many values
    pvTerminateModalDialog(p);           // terminate modal dialog
  }
  return 0;
}
```

6.9.5 DockWidget

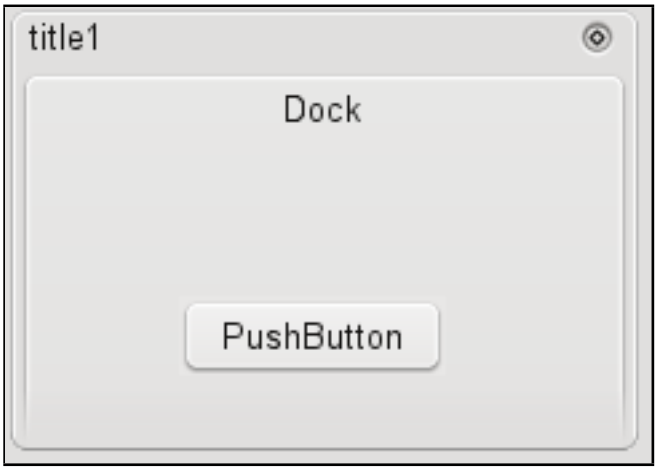

Figure 6.50: Dock Widget

Dock Widgets in principle work like modeless dialogs. But they can be docked at the borders of the pvbrowser window. The content of a Dock Widgets can be designed within the mask. You could place this design on the right most edge of the mask. Then you set the root object of this design (example: GroupBox) as root_id in the DockWidget. Then this object disappears in the mask and is set in the Dock Widget.

adding a Dock Widget

```
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      := title of the dialog
// dock_id    := ID_DOCK_WIDGETS + n . with n = 0...31 MAX_DOCK_WIDGETS
// root_id    := id of the root object. root_id is the id of the designed widgets.
//               The root object is inserted into the Dock Widget and
//               thus disappears in the mask.
//               allow_close := 0|1 allow the user to hide the dialog
//               floating   := 0|1 movable by the user
//               allow_X    := 0|1 Docking positions
//               functions that apply to Dock Widgets:
//               pvSetGeometry();
//               pvMove();
//               pvResize();
//               pvHide();
//               pvShow();
```

6.9.6 PopupMenu

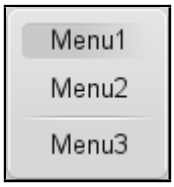

Figure 6.51: Popup Menu

A Popup Menu can be created with the following code. Popup Menus are programmed with the function 'pvPopupMenu'. id_return determines under which id the result of the PopupMenu will be delivered. You should choose negative values in order not to conflict with the widgets you have designed. The text input by the user is delivered in 'slotTextEvent'. Two comma within the text create a separator within the menu.

Creating a PopupMenu

```
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 Language translations

pvbrowser uses UTF-8. Thus you can even use non latin languages. It is possible to use cyrillic or chinese letters within your visualization. That is you can use all letters that can be represented with UTF-8. The menu text in pvbrowser can be adjusted to new languages within the INI file '.pvbrowser.ini' .
During pvbrowser is running you can switch from one language to the other if you have taken care for this in your pvserver. In processviewserver.h the macro '#define pvtr(txt) txt' is defined. The macro will simply return the original "txt". If you create a pvserver you should use 'pvtr("Any text")' for any text.
When you now want to translate the pvserver to several languages you can use a INI file (rlIniFile class). You include rlinifile.h within pvapp.h. rlinifile.h will redefine the macro 'pvtr(txt)' and will try to translate "any text" using the INI file. In main.cpp you insert 'rlSetTranslator' at the beginning of main().

Setting the default language for 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;
}
```

With this code the INI file 'translation.ini' is loaded and the default section is set to "GERMAN". Each call to 'pvtr("Any text")' will now try to translate the text to german.
During a client is connected you can switch from one language to the other. For example the user could hit a button and the pvserver would call 'pvSelectLanguage(p,"ENGLISH");' setting the current language for the user to english.

Sietting the language for a individual 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;
}
```

After you switched the language with 'pvSelectLanguage()' the example does a 'return 1' in order to return to 'pvMain()' and show the mask again now with the new language. The language corresponds to a section in the INI file. You can define any number of languages within one INI file. When you code your pvserver you should simply do this in the default language and enclose every text with 'pvtr("Any text in the default language")'. Then you would do the translation.
There is a special commandline argument to pvdevelop which will help you to extract the text you will have to translate.

Extract text to translate

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

This will search the sources of your pvserver for 'pvtr(' and dump the text to dump.ini.

Example dump.ini

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

From this you edit the INI file and translate to the target language

Translation to english

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

Please note that you can use the text as a format string to 'printf()' also. If the text to translate includes a '=' character this character must be quoted with '\' in order to distinguish it from the '=' charcter that stands between name and value in the INI file.
pvdevelop inserts the macro 'pvtr(txt)' when it generates a mask. If there is no translation the original text is returned. In pvbaddon 'pvbaddon/templates/myeventhandler' you will find a example for the language translation.
The PARAM structure contains a variable 'int language' which may be used to remember the chossen language. You can evaluate this variable for example when you want to use language specific units.

Setting the language in pvserver

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

6.11 Converting units

In the PARAM structure there is a member 'convert_units'. You can set p-convert_units to 0 or 1. (standard: 0).
Now use the function 'float unit(PARAM *p, float val, int conversion);' for unit conversion. If (p-convert_units == 0) the original value is returned.

Unit conversion

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

These are the available conversions. You should set p-convert_units depending on the language selection.

Unit conversions

```
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

The Layout Management can be defined in pvdevelop. Choose the according menu (right mouse button) within the graphical designer of pvdevelop. When designing the mask you can set min and max values for the dimension of widgets to arrange widgets so that they can be changed only within certain limits. If min and max values are the same the size of the widget is fixed.
Here is an example code for layout management.

Layout Management

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

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 remaining widgets from left to right
pvAddWidgetOrLayout(p,layout2,centerWidget,-1,-1);
pvAddWidgetOrLayout(p,layout2,rightWidget,-1,-1);
```

6.13 Setting the TAB order

The Tab order can be set in pvdevelop. Choose the according menu (right mouse button) within the graphical designer of pvdevelop. Then click the objects in the desired sequence. The clicked objects are hidden.

6.14 Using stylesheets within pvbrowser

Both Qt Widgets and HTML elements can be redesigned using stylesheets. This might be a possibility to adapt a visualization to different demands on desktop computers and mobile devices.
If you want to design Qt Widgets within pvbrowser please use the function

pvSetStyleSheet

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

Example: Set a button to red background color

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

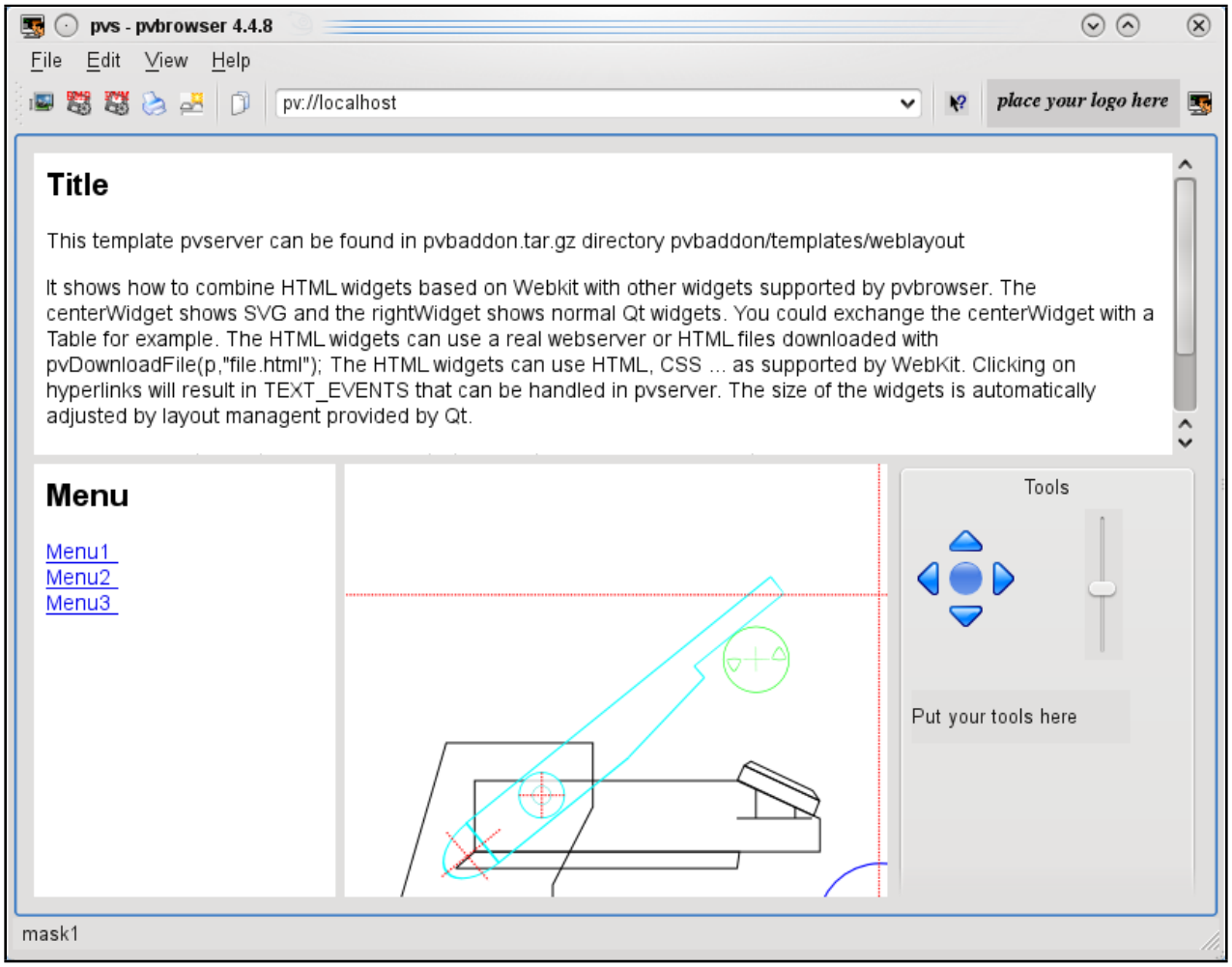

Figure 6.52: Layout for example code

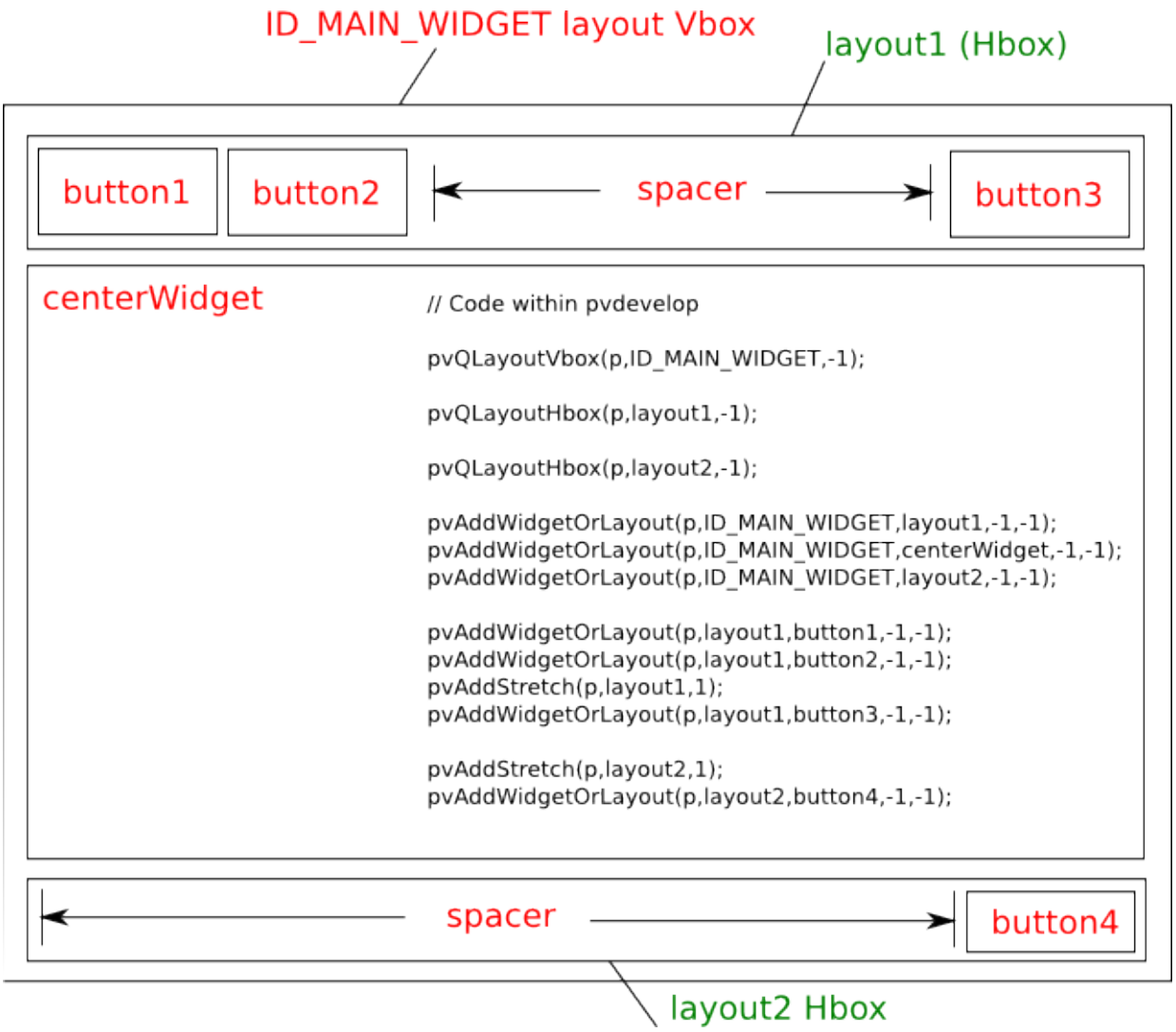

Figure 6.53: Layout example

For text you might use a rlString object to add longer stylesheet definitions. With

read method of rlString

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

you may read the stylesheet text from an external file and then call

Setting style from file

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

Examples and a detailed reference for Qt stylesheets can be found at:
`http://doc.qt.io/qt-5.5/stylesheet-examples.html`
`http://doc.qt.io/qt-5.5/stylesheet-reference.html`
pvbrowser can also use standard HTML elements within the TextBrowser object. See the constructor of the TextBrowser.

CSS stylesheets can be used for HTML content

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

There are a lot of resources on the internet where you can find information about CSS stylesheets.
For example:
`https://wiki.selfhtml.org/wiki/CSS`

6.15 Webcam

Webcams with Motion JPEG streams over http can be used with the class rlWebcam.
If you do not know under which URL the webcam delivers the M-JPEG video stream you can figure this out with tcpdump.

Using 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 integration 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 are short informations that are stored on the client computer if the pvbrowser client is configured to accept cookies.

Example for cookies

```
pvPrintf(p,ID_COOKIE,"%s=%s","cookie_name","cookie_values"); // setting cookie "cookie_name"
// snip
pvPrintf(p,ID_COOKIE,"cookie_name"); // requesting cookie "cookie_name"
                                     // As result you will receive a text event under
                                     // ID_COOKIE
```

6.17 Adding httpd functionality to your pvserver

Besides working as a pvserver for the pvbrowser client over the pv protocol a pvserver might also act as a httpd for a standard web browser over the http protocol. In order to do this please start your pvserver with the "-http" option. This will suppress the automatic sending of the pvserver version to the pvbrowser client. You might send the pvserver version as soon as you detect that a pvbrowser client has connected.

pvMain() for dual http and pvserver functionality

```
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;
    }
  }
```

```
}
```

Do not call generated_defineMask() in maskX.cpp if it is a http request

```
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 for dual http and pvserver functionality

```
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;
}
```

Chapter 7

Data Acquisition

Data Acquisition in pvbrowser is realized with separate daemons (processes running in background). These daemons 'talk' the protocol of the according fieldbus or PLC. A daemon consists of two threads where one thread reads data cyclically and writes the result to a shared memory and the other thread is waiting on a mailbox for commands for output data to the interface. The visualization can now read the shared memory and visualize it's content. If you want to output data the visualization sends a message to the mailbox.
An advantage of this architecture is that data acquisition and visualization are separated from each other. For example the visualization can be restarted without the need to read all inputs again because the inputs are still available in the shared memory and reading the data continues while you are stopping the visualization in order to make changes to it. Additionally it is no problem to run several daemons in parallel where each daemon has it's own shared memory and mailbox. Thus you are independent of protocol. You may run several daemons with another protocol on each.

ATTENTION: Under unix like operating systems zu must delete the old shared memory or mailbox if you changed their size.

```
# help for the commands
man ipcs
man ipcrm
# list ipc objects
ipcs
# remove shared memory with id=425986
ipcrm -m 425986
# remove mailbox with id=32769
ipcrm -q 32769
```

In pvbaddon you can find some daemons which are configured over a INI file. These daemons use the class rlDataAcquisitionProvider from rllib. On the side of the pvserver you use the class rlDataAcquisition for access to the shared memory and the mailbox. The variables are encoded as readable ASCII text with these classes.
Additionally it is possible to generate daemons from within pvdevelop where variables are encoded binary.
It must be decided if the overhead of ASCII encoding is acceptable or if it is better to use the binary encoding. The ASCII representation makes it easier for the developer where it is faster to access the binary encoded variables.
If you want to implement new protocols you can use an existing daemon as template and modify it to support the new protocol. The best template might be the modbus daemon.
You can use protocols that currently are not supported by pvbrowser. The only thing you need is a foreign library that implements this protocol. This can be foreign libraries written in C or C++ or libraries which you have created yourself by using other classes from rllib. If you do it like this we ask you to send us your implementation so that we can include it within our project.

7.0.1 Copy the daemon to the standard directory

The daemons from pvbaddon should be copied to a directory that is included in the $PATH environment variable of your operating system because then you can call them from any location by simple input of the name within the command line.

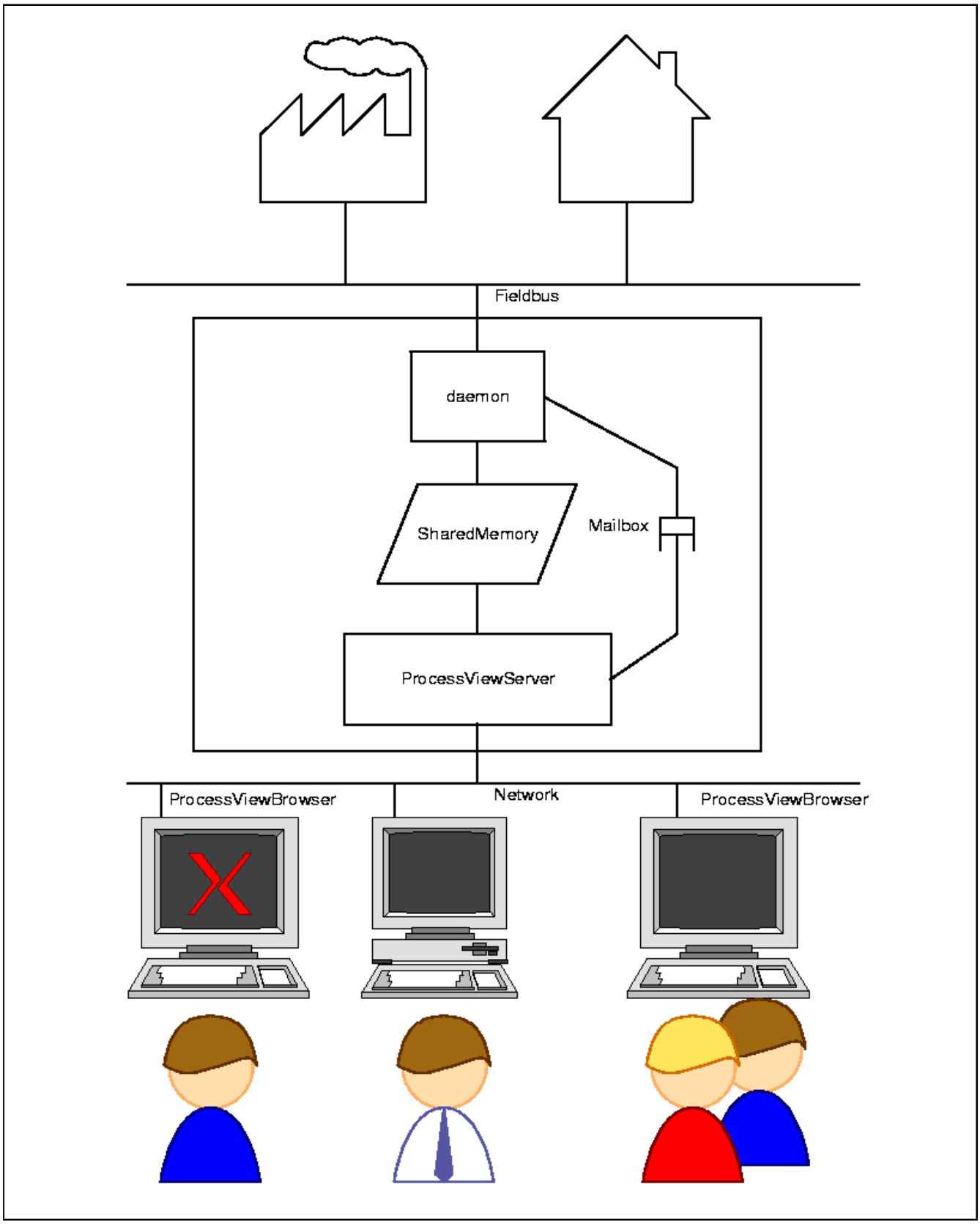

Figure 7.1: Principle of data acquisition in pvbrowser

Linux

```
cp your_daemon /usr/bin/
```

Windows

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

7.0.2 The INI file for your daemon

The INI file for the daemon could be located in any directory. Start the daemon with the following command.

Starting a daemon from a terminal/dos box

```
your_daemon /path/to/your_config.ini
```

For testing please use a terminal / dos box. If you want to start everything at boot time in the background it depends on if you use Linux or Windows. See section 'Starting a pvserver in the background'.

7.0.3 Configure the shared memory and mailbox

The examples in pvbaddon suggest a standard location.

Linux

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

Windows

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

Note that these directories must be created first by you. Note that the size of the shared memory must be equal in the daemon and in the pvserver.

7.0.4 Run the daemon and pvserver for testing

For testing open 2 terminals / dos boxes. Start the daemon in the first window and the pvserver in the second window. Every pvserver knows some command line options.

Show command line options of your pvserver

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

Starting your pvserver

```
yourpvserver -cd=/path/to/your/project
```

If the '-cd' (change directory) option is not given the pvserver will use the current directory.

7.1 Modbus

If you need to know more about the basics of Modbus please first read `http://en.wikipedia.org/wiki/Modbus` .
Modbus has the advantage that it's specification is publically available and 'reverse engineering' of the protocol is not necessary in order to implement it. Thus modbus has become very popular and there are a lot of hardware components that support this protocol. Thus modbus is the preferred protocol in many systems as also in pvbrowser.
Modbus is available in several versions. But there is always one master at the bus and up to 255 slaves. First modbus can use a serial connection with RS485. Here the two methods Modbus RTU and Modbus ASCII exist. Meantime modbus can also be used over TCP. Both can be combined by using a gateway that implements a modbus TCP slave on the network and is a modbus RTU master on the RS485 fieldbus.

The class rlModbus from rllib implements all these versions. By registering a rlSerial or a rlSocket object in rlModbus you can choose if to use the serial line or TCP. rlModbus is the basis for the implementation of the modbus daemon.
Within rlModbus all data addresses in Modbus messages are referenced to 0, with the first occurrence of a data item addressed as item number zero. Further, a function code field already specifies which register group it operates on (i.e. 0x, 1x, 3x, or 4x reference addresses). For example, holding register 40001 is addressed as register 0000 in the data address field of the message. The function code that operates on this register specifies a "holding register" operation and the "4xxxx" reference group is implied. Thus, holding register 40108 is actually addressed as register 006BH (107 decimal).

Modbus register map

```
0xxxx Read/Write Discrete Outputs or Coils.
1xxxx Read Discrete Inputs.
3xxxx Read Input Registers.
4xxxx Read/Write Output or Holding Registers.
```

7.1.1 Access using readable ASCII letters

In directory pvbaddon/daemons/modbus/client of pvbaddon you can find the modbus daemon that uses readable ASCII letters. In the INI file you specify what the daemon should read.

INI file for 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)
```

With USE_SOCKET you choose if to use a serial line or TCP. With USE_SOCKET=1 only the section [SOCKET] is relevant and with USE_SOCKET=0 the section [TTY]. With DEBUG some messages of the daemon can be switched on/off. In the section [RLLIB] the shared memory and the mailbox are specified. Please note that this example is for unix like operating systems. Using Windows [TTY][DEVICENAME], [RLLIB][SHARED_MEMORY] and [RLLIB][MAILBOX] must be specified in Windows syntax.

Example for Windows syntax

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

Please note that the directory for the shared memory and the mailbox must already exist.
In section [CYCLES] in the example 4 cycles are defined. In CYCLE1 10 contiguous inputStatus from Slave=1 and address=0 are read. In CYCLE2 8 contiguous coilStatus from Slave=1 and address=0 are read. In CYCLE3 2 contiguous holdingRegister from Slave=1 and address=0 are read. In CYCLE4 2 contiguous inputRegister from Slave=1 and address=0 are read.
After you have written the INI file for your system please start the daemon with DEBUG=1 within the command line prompt. With the DEBUG outputs you can verify that data is read correctly.
In directory pvbaddon/daemons/modbus/pvs in pvbaddon you find a pvserver, in which it shown how to access the shared memory and the mailbox.

Access to shared memory and mailbox from within 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 Access with binary encoded values

In pvdevelop there is a dialog for generating a modbus daemon which uses binary encoded values. In this dialog you can specify what should be read. With 'communication=serial' or 'communication=socket' you can choose which interface is used for the modbus communication. Please place a comment in front of the method you do not want to use. Again you can define several 'cycle' in which modbus is read cyclically and the values are written to the shared memory.

Please note that under Windows you must use the according Windows syntax. You have to take care that for the '\' in the path 2 '\\' must be specified because this results in C++ sourcecode in which this text is inserted. When you terminate the dialog the file modbusdaemon.cpp is generated in the current directory and is compiled. This results in a modbus daemon that reflects the configuration given in the dialog box. This daemon can now be used.

Please start this daemon within the command line prompt in order to verify if it works as desired. The cyclically read variables are written to the shared memory one after the other (compact).

The offset for each cycle is being defined in the generated file modbusdaemon.h . modbusdaemon.h is generated by pvdevelop and contains some definitions that you can use in your sourcecode. These are for example the name and size of the shared memory and the name of the mailbox. Additionally you find the offsets in the shared memory under which the values for the cycles are stored.

In directory pvbaddon/demos/modbusserial in pvbaddon you find an example.

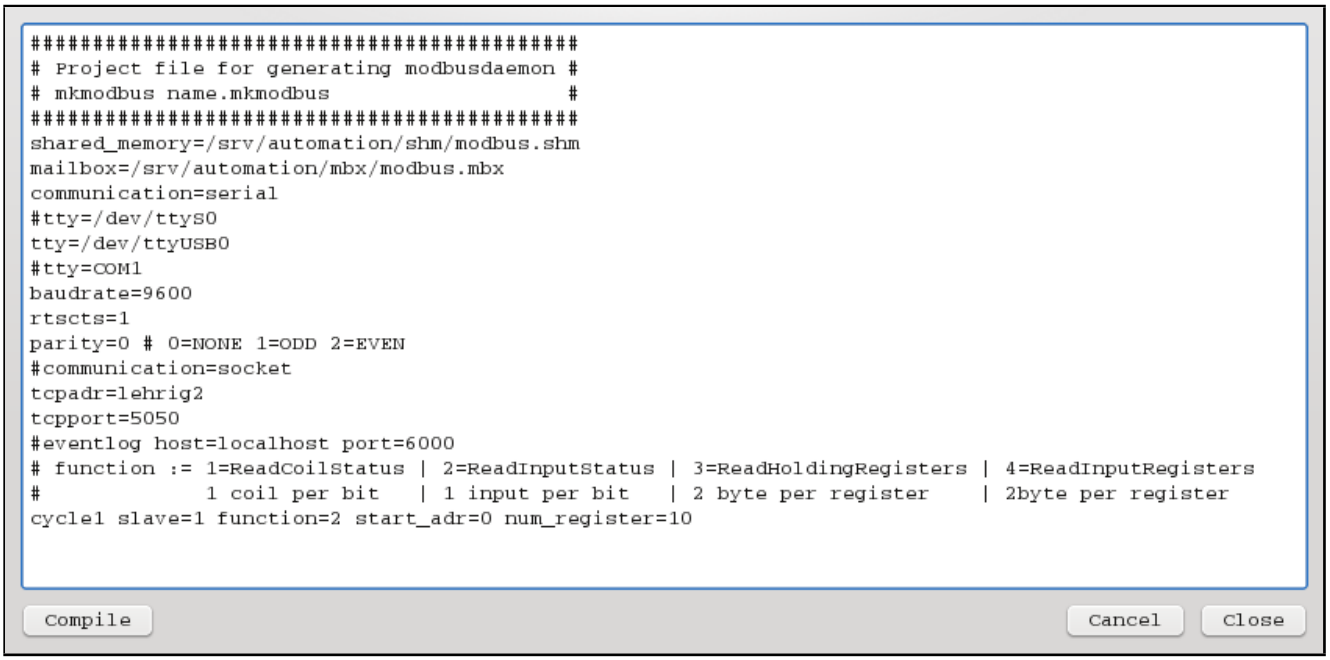

Figure 7.2: Dialog for generating a modbusdaemon from within pvdevelop

Access to shared memory and mailbox from within a pvserver and binary encoded values

```
#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

The daemons for Siemens S7 and S5 PLC are analog to the modbus daemon. Thus please read the section regarding modbus.

7.2.1 Access over TCP with readable encoded ASCII letters

In directory pvbaddon/daemons/siemenstcp/client in pvbaddon you find a Siemens daemon for TCP connections.
Here the class rlSiemensTCP from rllib is used.
The INI file is similar to the modbus INI file. You can specify as many PLC connections with one daemon as you want. HAVETO_SWAP should be set to 1 on a X86 CPU. If the processor uses a different byte order you can set this value to 0. On each SLAVE value there is a IP address or the name for the PLC. Then there is a comma and the PLC type. As optional parameter you can choose if to use the Fetch/Write protocol or the newer Siemens TCP protocol. In the INI file NUM_CYCLES=1 is set because for test purposes only the first cycle is used. If the second cycle should be done also you must set NUM_CYCLES=2 .

INI file for Siemens PLC over 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 Access over PPI using readable encoded ASCII letters

In directory pvbaddon/daemons/siemensppi/client in pvbaddon you find the Siemens daemon for PPI connections (serial line).
Here we use libnodave `http://libnodave.sourceforge.net/` A copy of libnodave is included in our package.
The INI file is similar to the modbus INI file.

INI file for Siemens PLC over 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 From pvdevelop generated daemons for Siemens TCP and PPI

Similar to modbus you can generate a daemon for Siemens PLC from within pvdevelop. The generated file is written as siemensdaemon.cpp into the current directory and is being compiled.

```
##################################################
# Project file for generating siemens daemon    #
# mksiemens name.mksiemens                      #
##################################################
shared_memory=/srv/automation/shm/siemens.shm
mailbox=/srv/automation/mbx/siemens.mbx
# type := S7_200 | S7_300 | S7_400 | S5
slave=0 adr=192.168.1.101 type=S7_200 fetch_write=1
slave=1 adr=192.168.1.102 type=S7_200 fetch_write=0
idletime=50 milliseconds
#eventlog host=localhost port=6000
# ORG := ORG_DB | ORG_M | ORG_E | ORG_A | ORG_PEPA | ORG_Z | ORG_T
# cycle := slave + org + dbnum + start + len
cycle1 slave=0 org=ORG_E  dbnum=0 start=0 len=8
cycle2 slave=0 org=ORG_A  dbnum=0 start=0 len=8
cycle3 slave=0 org=ORG_DB dbnum=0 start=0 len=32
```

Compile Cancel Close

Figure 7.3: Dialog for generating a Siemens TCP daemon from within pvdevelop

```
##############################################
# Project file for generating ppi daemon    #
# mkppi name.mkppi                          #
##############################################
shared_memory=/srv/automation/shm/ppi.shm
mailbox=/srv/automation/mbx/ppi.mbx
tty=/dev/ttyS0
baudrate=9600
# daveSpeed9k | daveSpeed19k | daveSpeed187k | daveSpeed500k daveSpeed1500k | dav
dave_speed=daveSpeed187k
idletime=50 milliseconds
#eventlog host=localhost port=6000
# area :=
# daveSD
# daveInputs
# daveOutputs
# daveFlags
# daveDB          //data blocks
# daveDI          //not tested
# daveLocal       //not tested
# daveV           //don't know what it is
# daveCounter     //not tested
# daveTimer       //not tested
# cycle := area, dbnum, start, len
cycle1 slave=2 area=daveDB dbnum=1 start=0  len=64
cycle2 slave=2 area=daveDB dbnum=1 start=64 len=64
```

Compile Cancel Close

Figure 7.4: Dialog for generating a Siemens PPI daemon from within pvdevelop

7.3 EIB Bus

In directory pvbaddon/daemons/eibnet/client in pvbaddon you find the EIBnet daemon that is used for communicating to the european installation bus over a TCP/EIB gateway with EIBnet.
Here the class rlEIBnetIP from rllib is used.
The INI file is similar to the modbus INI file. The IP address of the gateway and the address of your computer must be specified.
The EIB bus variables are not specified explicitly because the daemon will insert all data transmitted over the bus into the shared memory.

INI file for 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

In directory pvbaddon/daemons/ethernetip/client in pvbaddon you find the daemon which implements Ethernet/IP protocol that is used by Allen Bradley and Rockwell.
Here the open source project TuxEip is used. Unfortunately this project does not seem to be continued anymore.
The INI file is similar to the modbus INI file.

INI file for 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 and CAN

For Profibus and CAN we prefer the CIF cards from Hilscher `http://hilscher.com`.
These cards have a separate controller which handles the fieldbus protocol. The cards store the process variables in a dual ported memory onto which the controller and the PC have access to. With the configuration tool SyCon from Hilscher you define where within the dual ported memory a process variable is stored.
The new cards based on netX work according to the same principle but support practically every popular fieldbus protocol. The ARM processor which is integrated on these cards is loaded with the according firmware for the protocol.
In pvbrowser the class rlHilscherCIF is a wrapper for the Hilscher driver. Because these cards already store the process variables within the dual ported memory we do not use our shared memory and the mailbox. Instead we start a separate thread within our pvserver in which the card is read and it is possible to send data to the fieldbus.
In directory pvbaddon/demos/hilschercif of pvbaddon you find an example.
The globally defined variables sendData and receiveData can then be used by the visualization. With pbus.lock() and pbus.unlock() you can synchronize the mutual access to the variables by several clients.

Separate thread for Hilscher cards

```
#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 is the first platform independent version of OPC. In contrast to the classical OPC that is based on COM/DCOM from Microsoft OPC XML-DA works over http requests with XML and thus can work on every operating system.
In directory pvbaddon/daemons/opcxmlda/client of pvbaddon you find our daemon which implements a OPC XML-DA client.
Using this client you first read the object directory of your OPC XML-DA servers.

Reading a OPC XML-DA object directory

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

Example for the content of a 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 this itemlist you can comment out the variables you are not interested in.

Usage of 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
```

If the default command line arguments are sufficient a call to opcxmlda_client could look as follows.

Example of a call of opcxmlda_client

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

In your pvserver you have to use the class rlOpcXmlDa from rllib. An example for such a pvserver can be found in directory pvbaddon/daemons/opcxmlda/pvs of pvbaddon.

7.7 Usage of gateways

If a fieldbus protocol is not available in pvbrowser it could be possible to use an according gateway. Most probably you will use modbus on side of the PC.
Some examples:
Modbus LON Gateway
http://www.intellicom.se/lonworks_modbus_rtu.cfm
Profibus Modbus-TCP Gateway from 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 Template for further protocols

The modbus daemon in directory pvbaddon/daemons/modbus/client of pvbaddon can be used as template for your own data acquisition.
Even more simply you may use the modbusdaemon.cpp that is generated from within pvdevelop as a template for developing your own data acquisition.
If you do not want to use the shared memory and the mailbox you can start a separate thread within your pvserver for reading your process data. Please have a look at 'Profibus and CAN' how this is be done.

7.9 Template for Arduino integration via serial USB device

The Arduino microcontroller boards have a serial USB device that can be connected to a host computer. On Linux host computers this will be represented as /dev/ttyUSBx.
Within main.cpp of our pvserver we may start an additional thread that communicates with the Arduino over this serial interface (tty). There we might interpret the strings that result from print calls on the Arduino (send to the tty). Also we might use a rlFifo for communication with a slotFunction() that serves a client (different thread). Via this rlFifo the slotFunction() may send commands to the Arduino communication thread.

Additional thread that communicates with the Arduino board via USB

```
// 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 the additional thread within 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"
```

slotFunction() that sends something to the 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 ...
```

Chapter 8

Distributed Control System (DCS)

pvbrowser can be extended to the function of a DCS. We can already capture and output data of PLC systems, remote IO system via field buses or directly in our computer IO. These data pvbrowser stores in a 'shared memory' and can be displayed by the visualization and can be influenced by the visualization.
There is now no reason why a DCS system should not be implemented in addition to visualization in a separate process. Both visualization and DCS system can access the 'shared memory' at the same time.
In principle, it is possible to have any number of processes or threads simultaneously access the 'shared memory'. For practical reasons, but you should not use too many separate processes.
It appears useful, for example, to combine the data acquisition and the DCS together in a process.
The 'Shared Memory' you can access with read / write operations where a mutex will lock access from each other. It is also possible to get the base address of 'shared memory' and then take care about the synchronization yourself.
A sensible approach here would be a complex set of user data structure in the 'shared memory' that contains all the necessary information on outputs, inputs and calculated variables.

User-defined data structure

```
typedef struct
{ ... }
USER_DEFINED_DATA_STRUCTURE;
```

Since the individual values in the data structure are basic data types (int / float), they are transferred as atomic write operations to the 'shared memory'. So it is possible to dispense with locks of 'shared memory', so as to improve the parallelism of the overall solution under certain conditions. The overall solution is parallelized by multi-core processors. It is therefore not only useable as timeshare.
A DCS can now be implemented in a separate process and is composed of several parallel threads.

- *A thread with a loop for data collection*
- *A thread with a loop for logging in a database*
- *Multiple state machines, each running in its own thread*
- *Continuous Controller (P / PI / PID ...), each running in its own thread*

In our rllib all necessary classes are available to create such a process with little effort. The user can draw on templates (templates) that he can use as a starting point for his own solution. The programming effort is so greatly reduced, and the user only has to enter their own logic. The framework can be easily copied.
Such a template for data collection with Modbus + state machines can be found for example in 'pvbaddon/templates/statemachine'.

8.1 DCS template with data acquisition and state machines

The header file 'plcapp.h' includes some more headers and defines a custom data structure. 'main()' initializes the user-defined data structure and places it in the 'shared memory'.

user defined data structure in shared memory

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

Then the threads for the state machines are started . After initialization, 'main()' goes into an infinite loop, reads the inputs and writes the outputs cyclically. It is important to note the correct sampling rate. The analog inputs must be band-limited and the sample rate must be high enough to sample the analog inputs at least 2 times in the period of the cut-off frequency. The cycle time of the data acquisition should be at most half as large as the cycle time at which the state machines work.
Look here to see the corresponding source code:

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
```

A more detailed description of the templates can be found in the directory or in the reference classes in the rllib. Continuous controllers can be realized for example by means of class 'rlController'. This allows you to parameterize P/PI/PID controllers and start/stop them as parallel threads.
The state machines are working with pointers to functions. Therefore 'switch()' commands for the state transitions are not necessary.

8.2 Visualization using a state machine

It has been found to be convenient to design state diagrams with `http://graphviz.org` and generate SVG graphics from there. The object id's within the SVG graphics can be specified within the 'inputfile.dot'. This means that the SVG be transferred directly into the visualization, as you can see in the template. The id's contained in the SVG are used to address the graphical objects with our rlSvgAnimator class.

Graphviz file to generate a state graph as 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;
    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;
```

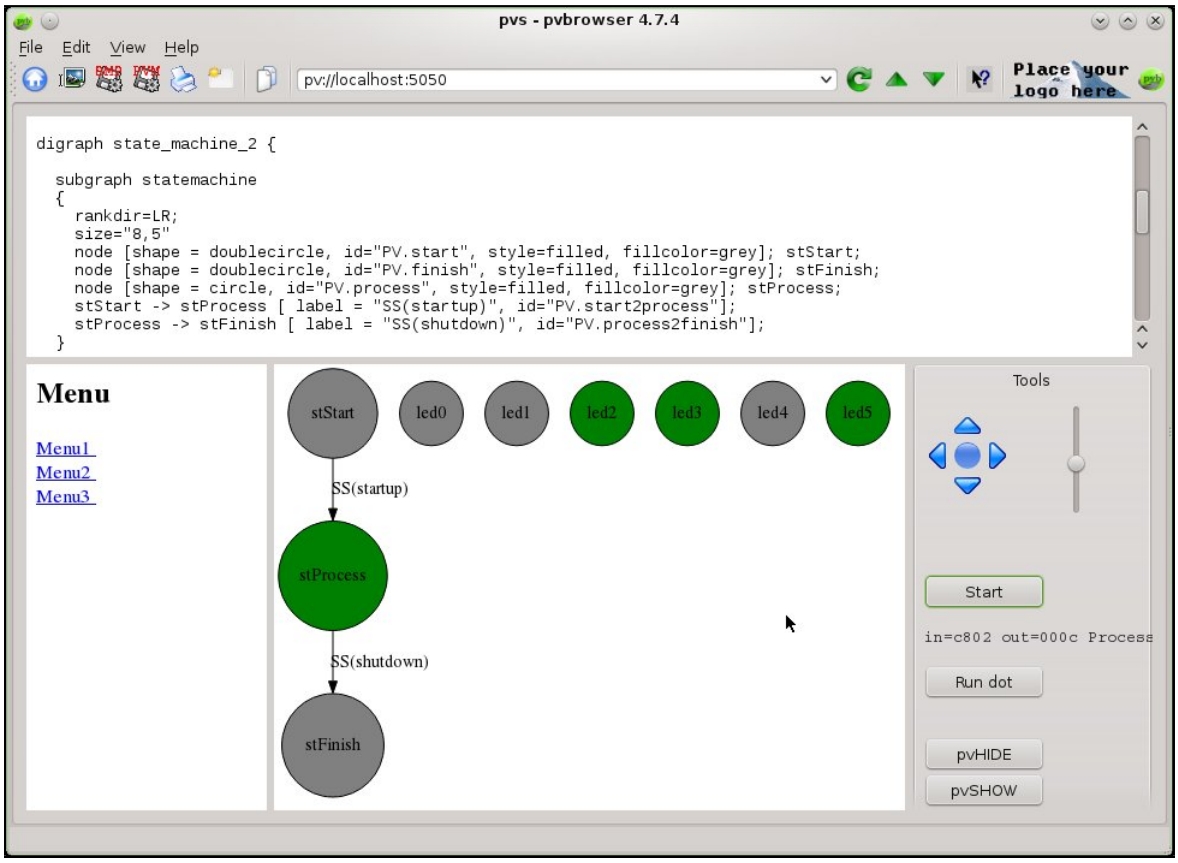

Figure 8.1: State machine visualization

```
    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;
}
```

Chapter 9

Starting a pvserver in the background

It should be possible to start a pvserver in background at boot time. The method for this is depending on the used operating system.
With a pvserver you can choose by the preprocessor symbol USE_INETD if to use a multi threaded server or if to start pvserver with the help of (x)inetd.
The multi threaded server consists of several threads. The main thread is waiting for new clients. If a client connects a new thread is started for serving the client. The main thread can wait for more clients.
(x)inetd is a super server that waits for clients and then starts the real server by forking a new process. There STDIN and STDOUT are used for communication. (x)inetd redirects STDIN and STDOUT of the server to the network connection.

9.1 Linux

For the multi threaded pvserver you can generate a 'startscript' from within pvdevelop using the menu 'Linux-WriteStartscript'.

The startscript for a server generated from pvdevelop

```
#!/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
```

You can adjust the variables DAEMON_PATH and DAEMON. The 'startscript' must be copied to /etc/init.d where all start scripts of servers are stored. Now you can do the following:

Usage of the startscript

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

In order to start YOUR_PVS in the right runlevel a link within 'rc5.d' must be created. Under openSUSE you can do this with YaST in the 'Runlevel Editor'. Choose YOUR_PVS and simply activate it.
If you want to use inetd or xinetd do the following. Install and activate (x)inted. In file /etc/services add the following lines.

Definition of a pvserver at port 5051

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

This defines a service pvsuper at port 5051. In /etc/xinted.d you need the following file.

/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
}
```

In order to activate the pvserver you must restart (x)inted.

restarting xinetd

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

9.2 Windows

For starting a pvserver as a 'Windows Service' in background you can use xyntservice `http://www.codeproject.com/KB/system/xyntservice.aspx` .
Because xynetservice seems to be removed from the internet, pvbrowser now provides the small tool 'pvservice' as a replacement for xynetservice. `http://pvbrowser.de/pvbrowser/index.php?lang=en&menu=6&left=3` .
The use of inetd is not possible under Windows because there is no standard inetd available on Windows.
If you want to start the pvserver from the 'Windows Autostart Directory' a 'DOS Box' will appear in which your pvserver is running. If you don't like the 'DOS Box' you can use the small tool pvb/winmingw/bin/start_pvbapp.exe .

9.3 OpenVMS

You can use loginout.exe for starting your pvserver in multi threaded mode. You can place this command in SYS$MANAGER:SYSTARTUP_VMS.COM .

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
$
```

You can use UCX SET SERVICE to start a pvserver in inetd mode.

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
```

The startup file for ucx set service looks as follows.

pvserver_startup.com

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

9.4 pcontrol

Within a typical automation system you may have several processes that build your automation. Most of these processes will run in background. There must exist a method for monitoring and controlling these processes. Also it should be possible to send event log messages to a central instance. The event log messages must be readable online and offline from historical archives.
pcontrol is a pvserver which is based on rllib and uses pvbrowser to fullfill these demands. The 'pvserver -pcontrol' starts the background processes and controls them. If you want to automatically start a complex automation system in background you can use pcontrol. pcontrol must be started in background as described in the previous section. All other processes can now be started and controlled by pcontrol.

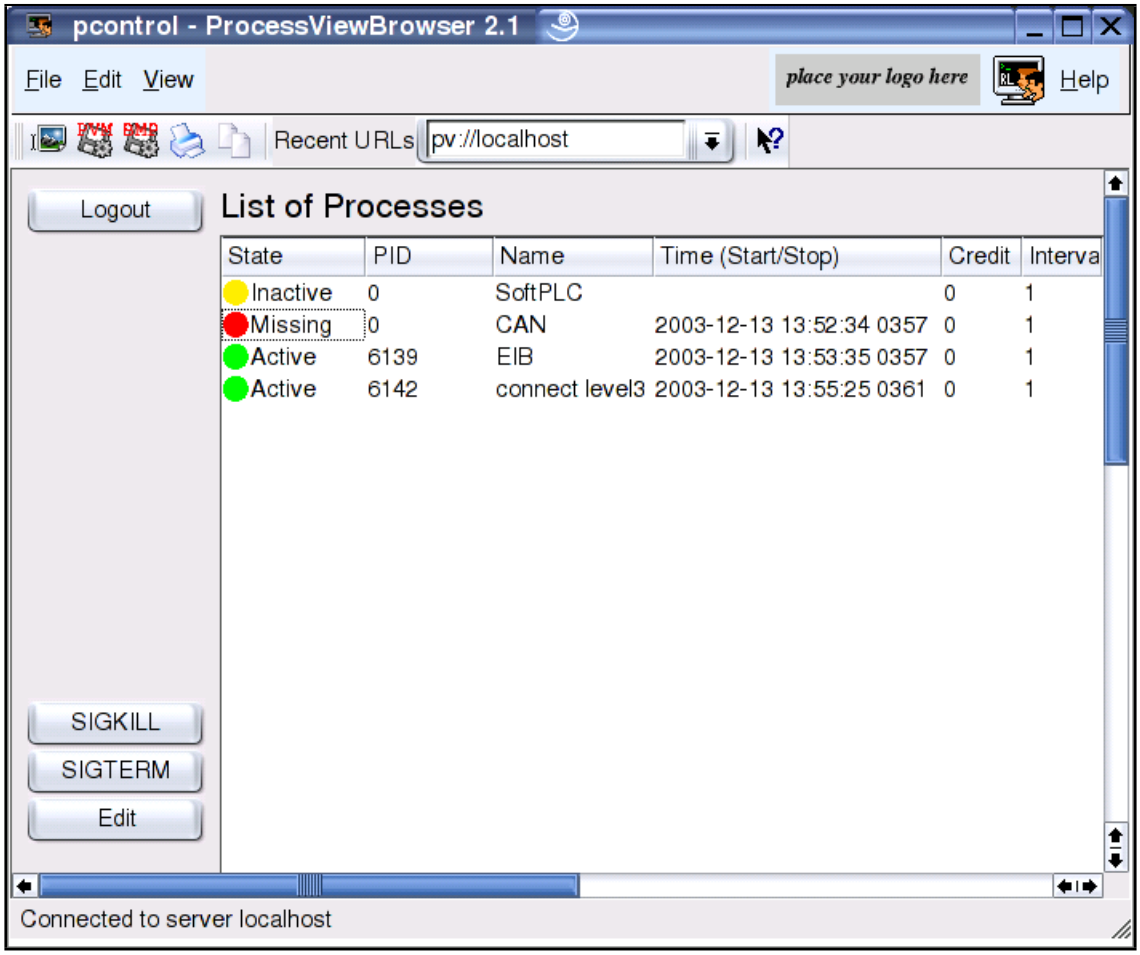

Figure 9.1: pcontrol controls background processes

With pcontrol you can receive event log messages from other processes (even over the network) In rllib there are some functions that enable such processes to send event log messages. First you define the server pcontrol with rlEventInit() . Then you can use rlEvent() to send event log messages similar to output text with printf() . There date/time, the name of the source file and the line in the source file where the output has been done will be included in the event log message.

Outputs of event log messages

```
#include "rlevent.h"
```

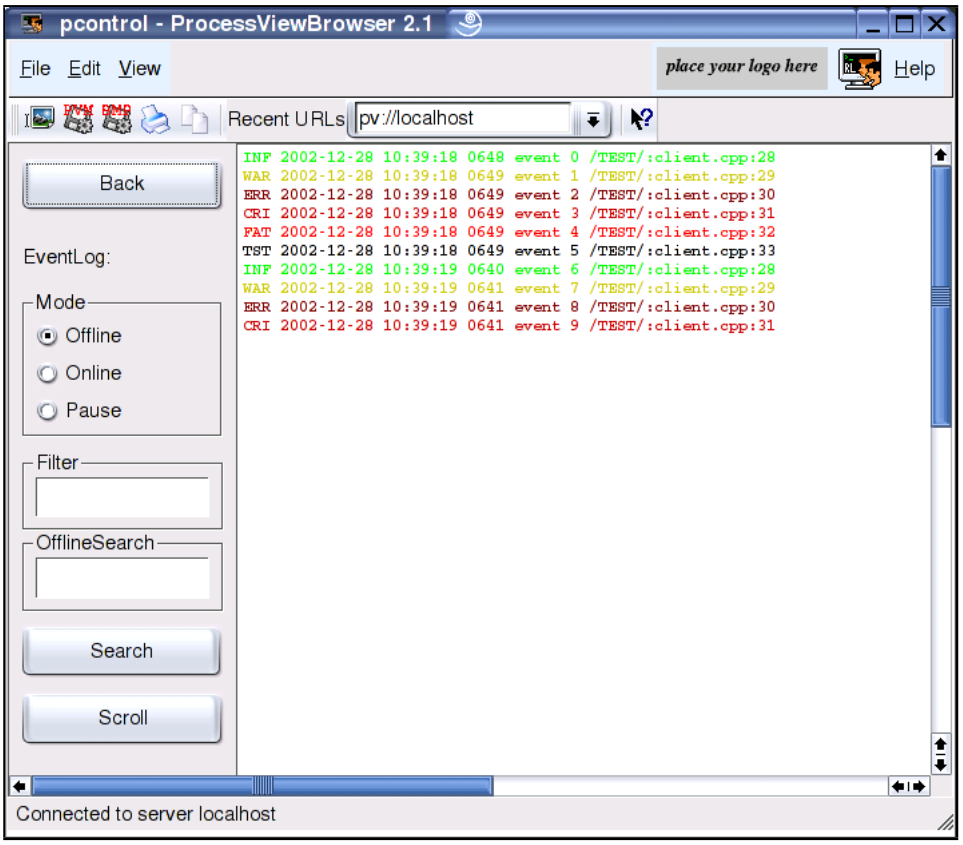

Figure 9.2: You can view event log messages in pvbrowser

```
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 Access control

The following functions allow you to define how many clients are allowed to connect to your pvserver from 1 IP address and how many clients are allowed to access your pvserver in total at one time. The value of 'max_clients' must be within the range from 1 to MAX_CLIENTS. You should insert these functions into main(). By default these values are set to MAX_CLIENTS = 100.

Set the maximum allowed number of clients

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

In order to get a secure connection to a pvserver over the internet it may be usefull to explicitly allow some IP ranges or disallow IP ranges. If you want to use these filters you have to store the files 'deny.ipv4' and 'allow.ipv4' within the directory your pvserver is running from.
If none of these files exist every IP Adress is allowed to connect.
If only the file 'allow.ipv4' exists only the explicitly allowed IP addresses can connect.
If only the file 'deny.ipv4' exists you can connect from every address that is not explicitly denied.
If both files exist access control hehaves as if only 'allow.ipv4' would be present.

Example: allow.ipv4

```
# A comment
# Here you can input a list of IP ranges that are allowed to connect.
# A valid address / address range has the following form aaa.bbb.ccc.ddd/number_of_significant_bits.
192.168.1.0/24 # allow connections on the private subnet 192.168.1.X
192.168.0.14/32 # allow connections from IP addresse 192.168.0.14
# input as many ranges as you want.
```

Example: deny.ipv4

```
# Here you can input a list of IP ranges that are not allowed to connect.
# A valid address / address range has the following form aaa.bbb.ccc.ddd/number_of_significant_bits.
100.101.0.0/16 # deny connections from 100.101.xxx.yyy
200.201.0.0/16 # deny connections from 200.201.xxx.yyy
```

A possible use case would be to only allow the private subnet to connect to your pvserver. If an external maintenance task should done the operator could explicitly allow the IP address of the customer engineer to connect over the internet.
The according functions for IPv6 use the files 'allow.ipv6' and 'deny.ipv6' and work as the IPv4 solution.

Example: allow.ipv6

```
# Here you can input a list of IP ranges that are allowed to connect.
# A valid address / address range has the following form ipv6_address/number_of_significant_bits.
::1/128
0001:0002:0003:0004:0005:0006:0007:0008/128
0000:0000:0000:0000:0000:0000:1000:0000/128
# input as many ranges as you want.
```

Chapter 10

Further suggestions

pvbrowser is a framework for HMI/SCADA written in C/C++ . Thus you are not limited to the options described above but can implement further features by your own or with the use of foreign libraries.

10.1 Using any database system

For using database systems we suggest to use the Qt library because this library comes with the development kit that we use anyway with pvbrowser.
The 'SQL Module' of Qt provides a comfortable API for SQL databases. By using plugins it supports almost any popular database system. In order to use the Qt library in your pvserver you must remove 'CONFIG -= qt' from the project file. Then qmake will bind the Qt library and you can use the Qt SQL classes.
An example for the usage of the Qt SQL module in a pvserver can be found in pvbaddon.tar.gz See directory: pvbaddon/templates/qtDatabase

10.2 Using a spreadsheet program at the client

If you want to use a spreadsheet program in order to enable the users to do their own evaluations you can create a CSV file on your pvserver and copy it with pvDownloadFile() to the client computer. The function pvClientCommand will now call the spreadsheet program with this CSV file.

pvClientCommand

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

The 'csv-viewer' is defined within the options in pvbrowser client. Here you can insert OpenOffice or Excel for example.

10.3 Generating reports with LaTeXby creating a PDF file

For documentation and quality control of the production you often need reports. A good format for reports is PDF. For example you could install LaTeXon your server, create document templates in LaTeXand insert the results from the production in there. After creating a PDF file with the use of LaTeXyou can copy it with pvDownloadFile() to the client computer.
The function pvClientCommand can be used to start your PDF viewer with this file.

pvClientCommand

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

The 'pdf-viewer' is defined within the options in pvbrowser client. Here you can insert Okular or Acrobat Reader for example.

10.4 Generating reports with HTML by creating a PDF file

Using the commandline program rlhtml2pdf you can convert HTML reports into PDF format. But rlhtml2pdf is based on the QTextDocument class and thus does not support all HTML and CSS features. With the following command you can get help for the rlhtml2pdf tool.

Help for rlhtml2pdf

```
rlhtml2pdf --help
```

10.5 Statistical evaluations

There are a lot of statistic programs that are run from command line and generate bitmap graphics as output. You can integrate this statistic programs the same way as external plot tools. With unix like operating systems on the serverside we suggest to use the class rlSpawn from rllib. This class starts an external program and connects it's STDIN and STDOUT over a pipe with the rlSpawn instance. You can now control the external application using rlSpawn from within your pvserver. The results of the statistics can then be included in the visualization or within the reports generated with LaTeX.

10.6 Ram disc for 'temp' directory of pvbrowser

The pvbrowser clients uses a temporary directory for storing files. This temporary directory is specified within the options of the pvbrowser client. You can set this directory into a ram disk for optimization. Furthermore this would have the advantage that this directory would be empty after restarting your computer.

Creating a ram disk with 512MB capacity under Linux in sub directory ram

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

Chapter 11

Conclusion

In this documentation we have described pvbrowser. With this framework you can quickly and relatively easy create visualizations and the according data acquisition. The time needed for creating a visualization can be compared to the effort of creating a webpage. Visualizations created with pvbrowser are always able to work over a network. Similar to a web browser pvbrowser can be used to jump from one visualization/pvserver to the other and thus view and control a complete plant with this distributed system. Because the communication is over TCP/IP pvbrowser can even be used over the internet.
It should be explicitly warned not to open critical parts of a plant for control over the internet. A visualization for internal purpose should use ports that are blocked against internet access by a firewall. Parts of the visualization that can be accessed over the internet should only allow viewing the state but do not allow inputs to the plant. If you need control over the internet you should use the pvssh:// protocol (Secure Shell) and use a strong password. It would be a good idea to terminate the connection if somebody inputs a wrong password. Thus inhibiting 'brute force attacks'.
Because pvbrowser can run on all popular operating systems there are nearly no limits. The experts may use the advantages of unix like operating systems to operate the server whereas the normal users can use their well known operating system. A visualization can also be started in fullscreen mode where the users do not even know which operating system is behind.
We hope that you will use pvbrowser successfully and would like to get response that will help to improve pvbrowser. This might be suggestions for new features, bug reports, contribution to documentation, sending patches or contributing even new drivers/daemons for further protocols.
We wish you success in using pvbrowser.

Yours pvbrowser community

`http://pvbrowser.org` July 19, 2016

Appendix

List of Figures