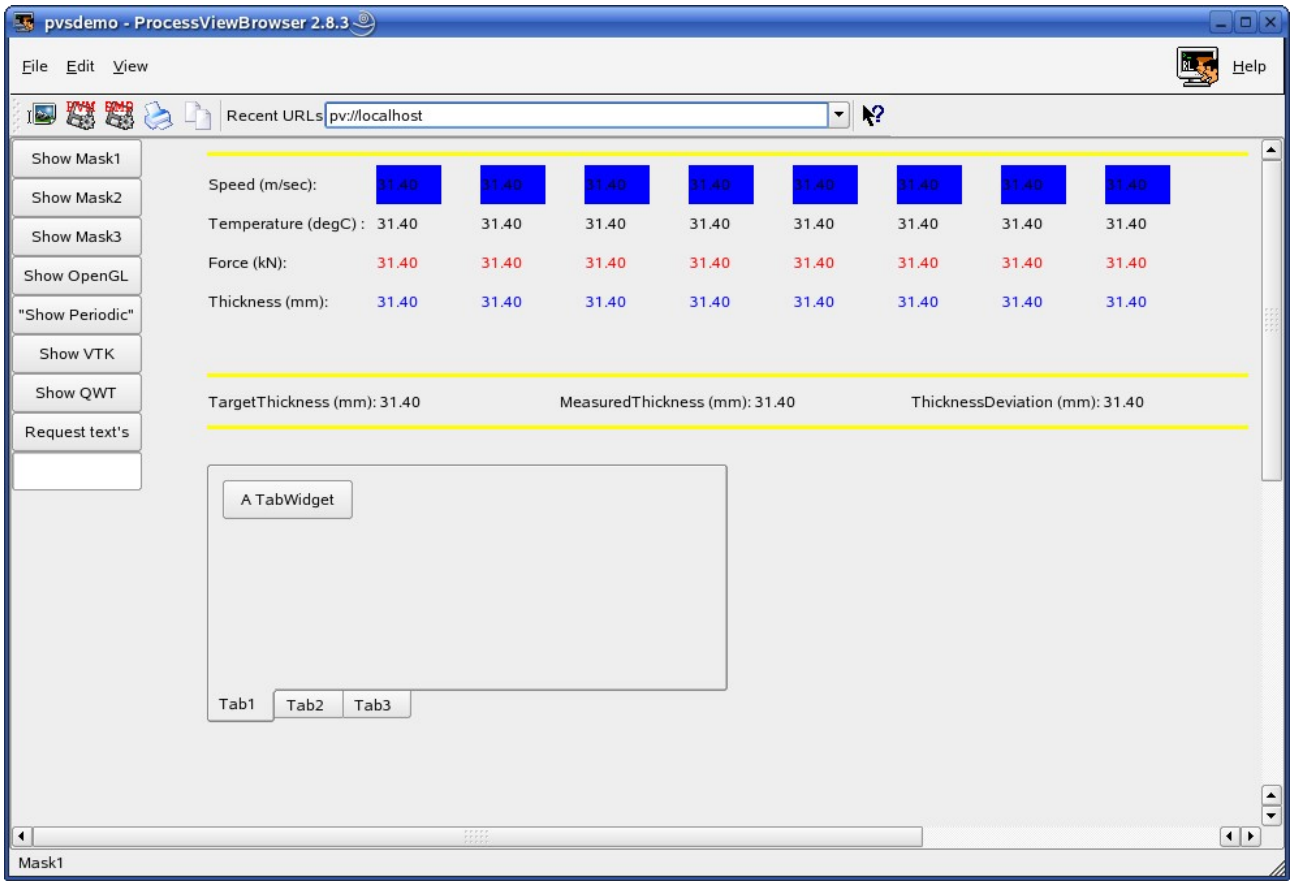
Process View Browser

Primer



Content

Preface

We present a Open Source process visualization, HMI and SCADA system. From the view of the user it works almost like a web browser. Because pvbrowser uses Qt it runs on many platforms. If you want to create a process visualization using pvbrowser you should be familiar with ANSI-C. Alternatively you can code in Python, Perl, PHP or Tcl.

Many people think, that process visualizations can be created graphically. We think, this is correct for the layout and design of the masks that are displayed on screen. But programming the logic of a visualization is much more powerful and flexible.

Please note that many web pages are programmed using PHP. Thus programming is also used for creating web pages.

If you want to create a visualization with pvbrowser you have to create a pvserver. This is done using pvbuilder. Programming is relatively easy, because only simple ANSI-C features are used. But you are not limited to ANSI-C. You can also use C++ and any foreign library written in C/C++ from within pvserver.

1 Introduction

The “ideal” process visualization would run in a standard web browser. In this case nothing had to be installed on the client computers, because web browsers are standard. Unfortunately web browsers have not been invented with process visualization in mind. In principle a web browser reads a HTML file from the webserver and displays it's content. Each time a HTML file is read the network connection is opened and closed. Dynamic changes in the content are difficult and inefficient.

A Java Applet might be a solution. Unfortunately complex Java Applets need a long time to load. Furthermore Java Applets are not famous for being fast.

We have build a new browser in C++, which is optimized for process visualization. Instead of HTML it dynamically handles Qt Widgets. pvbrowser can be run as standalone application or as kpart for konqueror.

Because pvbrowser is a browser it can display any visualization. There is no need to update client computers running pvbrowser, when you change your visualization. This is in contrast to many process visualization systems which use the “fat client” model. Furthermore pvbrowser supports many platforms at the same time.

When you choose the URL of your pvserver from within pvbrowser a TCP network connection is opened. The network connection will stay open until you terminate the session with your pvserver. After opening the network connection pvserver will start sending commands to pvbrowser. Each command is a line of text which is terminated by a newline. pvbrowser will interpret the text and call the according functions from the Qt widget library. Thus pvserver handles the whole widget tree within the main window of pvbrowser. When the user triggers an event (e.g. hit a button), a line of text will be send from pvbrowser to pvserver. In pvserver there is an event loop, in which these events can be handled.

For programming pvserver we provide a library that encapsulates the lines of text send to pvbrowser. The reference for this library can be found in “ProcessViewServer Programming” in the Manual, which is also available from the online help of pvbrowser and pvbuilder.

A substantial part of pvserver is not written manually. Instead pvbuilder and the file converter ui2pvc will gently generate the code for you. The functions provided by the library can be inserted by choosing from a menu in pvbuilder.

The whole layout and design of the masks you are using in your visualization is made graphically using “Qt Designer”. This is the GUI designer from http://www.trolltech.com/. The result of Qt Designer is a XML file. From this XML file your pvserver program will be generated automatically.

When you have further questions after reading this primer feel free to contact us at mailto:lehrig@t-online.de?subject=pvserver further questions .

Have fun using pvbrowser.

2 Installation

You can install from source or binary package. When installing from source refer to README in the main directory of the tar.gz file. Windows users may unpack the tar.gz file using WinZIP.

3 Running demo

There is a demo pvserver included in the installation packages. Change to the directory containing the demo and start the demo server.

Linux:

cd /usr/share/doc/packages/ProcessView/pvsdemo/

./pvsdemo -sleep=100

Windows:

Navigate to .\pvb\ExampleServer\Debug

Double click ExampleServer.exe

Within the console the demo pvserver prints the following:

Info: going to accept on port 5050

This means that the server is waiting for clients on it's default port 5050.

Currently the pvserver is started in foreground. Howto start pvserver in background will be explained later. The pvserver can be controlled with some options.

lehrig@nb3lehrig:/usr/share/doc/packages/ProcessView/pvsdemo> ./pvsdemo -h

ProcessViewServer 2.8 (C) by Rainer Lehrig 2000-2005 lehrig@t-online.de

usage: ProcessViewServer -port=5050 -sleep=500 -cd=/working/directory

default:

-port=5050

-sleep=500 milliseconds

-cd=/working/directory

4 Running pvbrowser

After starting the demo pvserver you may start pvbrowser.

Linux:

processview

Windows:

.\pvb\bin\processview_without_vtk.exe OR

.\pvb\bin\processview_with_vtk.exe

pvbrowser will connect to it's default pvserver at localhost. Within the window you see a simple demo visualization. The demo is mainly a periodic table of the available widgets. Please experiment around a little. Also look to File->Options. Here you can customize pvbrowser. Note that some changes under Options only take effect after restarting pvbrowser.

5 Using pvbuilder

pvbuilder is a simple IDE for developing pvservers. It relies on some external commands. These are:

Linux:

qmake, designer, g++, make

Before using pvbuilder make sure these commands are available. You may have to install the qt development package first and link qmake and designer to /usr/local/bin/

Windows:

fake_qmake, designer, Visual C++

Before using pvbuilder make sure Visual C++ is installed. The installation package only includes a very old non commercial version of designer. If you want the current designer and qmake you have to get Qt3 from trolltech or you may use http://qtwin.sourceforge.net/index.php/Main_Page . In the bin directory there are batch files you can edit in order to use these newer tools or even a different C++ compiler.

5.1 New visualization

Now start pvbuilder. Choose “File->New visualization”. Within the dialog box choose a empty directory for your project. Per default the new project is called “pvb”. You may change this and click OK.

Now Qt Designer is started and you can input your first screen mask.

5.2 Using “Qt Designer”

Within Qt Designer choose “Dialog” and click OK. Now you can insert the widgets you want to use. Some parameters of the widgets can be input in the “Properties” panel.

When you are ready with the design of your mask save it to disk. Qt Designer will write a XML file maskx.ui which stores everything you have designed.

Now terminate Qt Designer.

5.3 ui2pvc

pvbuilder will always call ui2pvc after you have edited a mask with Qt Designer. ui2pvc will convert the XML file produced by Qt Designer to sourcecode.

5.4 Programming

If you just have created a new visualization you will see the generated code within pvbuilder. Please browse the code and the project file. There is 1 project file, 1 main module, 1 headerfile and

currently 1 mask. Try to understand the structure of the pvserver. The code of the mask is split into an area you will have to edit and an area that will be generated from the Qt Designer XML file.

Before editing anything start your pvserver with Action->start server. Now start pvbrowser with Action->start browser. Now you should see your design within pvbrowser. You are now going to input your own logic into the mask code.

5.4.1 Output to widgets

The reference for programming can be found at “ProcessViewServer Programming->Modules” within the manual. All these functions have a “PARAM *p” as first parameter. In this structure pvserver holds some important information. The most important is p->s, which is the socket pvserver uses for communication with the client. Most of the provided functions will simply send a line of text terminated by newline to pvbrowser. pvbrowser will interpret this text and call the according Qt method.

Example:

pvPrintf(p, label1, ”Hello world. This is call number %d”, call_number++);

All widgets that you input in Qt Designer are listed in the enum in your generated mask. Assume you have input a text label “label1”. Then you can print text to it as shown above. Most functions described in the reference manual work that way.

The most frequently used output functions can be inserted by selecting the “Widget Name” toolbox in pvbuilder and clicking on the according widget name id.

5.4.2 Event handling

Within the message loop of your mask, you can insert your code. For example position the cursor to BUTTON_EVENT in the message loop. Now select the “Widget Name” toolbox, click at the button id and insert a BUTTON_EVENT.

```
case BUTTON_EVENT:
  //slotButtonEvent(p,i,&d);
  printf("BUTTON_EVENT id=%d\n",i);
  if(i == button1) return 2; // call mask 2
  break;
```

In the above example we terminate the current mask and return to pvMain() in order to show a different mask.

5.4.3 Graphics

Graphics can be done using different methods. Please view “Help->solutions” in the help system of pvbuilder. You can simply copy&paste these solutions into your code and modify it to your needs.

Using “Custom Widgets->QDraw” you can display XY-plots, simple line graphic and SVG graphics.

Using “Qwt->Plot” you can display more elaborate XY-plots.

You can also use external plot tools like GNUplot.

Bitmap pictures can be inserted using a QFrame in Qt Designer, set paletteBackgroundPixmap in the Property Editor pane and select a graphics file, set whatsThis=filename in the Property Editor pane.

5.4.4 Tables

Tables are a powerful means for displaying and editing tabular data. The data may come from physical interfaces or from database systems. The most simple method to output something to a table is.

pvTablePrintf(p, table1, x, y, "this is the cell text");

For more information view “Modules->construction->pvQTable” in the reference manual.

5.4.5 Assigning physical values

From within pvserver you can use any external library written in C/C++. Thus you may implement access to physical values. Such a library is rllib, written in C++. The necessary code for acessing modbus and siemens PLC's is already included in the generated code. In order to use it you only have to comment it out. Accessing modbus is shown below.

val = modbus.readBit(offset,number);

For more information view the online documentation.

6 Starting pvserver in background

Per default pvserver is a multithreaded server. For each client that connects to it a new thread is started. pvserver has to be started when your computer boots. Within the booklet this procedure is described for the different operating systems.

Alternatively you may start pvserver from xinetd. In this case you have to uncomment “DEFINES += USE_INETD” and use libpvsid.so instead of libpvsmt.so in the project file and recompile.

7 pcontrol

pcontrol is a pvserver available as separate download. It provides an event log system and a means to start processes in background and to control them. Using pcontrol you can comfortably print event log messages from any of your processes in the network to a central instance. The operators can browser the event logs from within pvbrowser. Also you may start all other processes your automation is consisting of from pcontrol. This may be your pvservers, daemons for fieldbus access, soft PLC's or whatever.

8 Conclusion

You should now be familiar with the basic structure of pvserver development. For further reading view the manual and the booklet available from online help. Within “solutions” you may find interesting code sequences you may copy&paste into your code.

pvbrowser operates on widgets (buttons, labels, tables ...). The function for constructing such a widget is automatically generated for you by pvbuilder and ui2pvc. The description of these constructors can be found under “manual->modules->construction”. Please view this topic because you will find all functions that apply to a special widget there.

We always appreciate feedback and wish you

much fun using pvbrowser.