pvbrowser manual

http://pvbrowser.org

31 ottobre 2013

Questa versione del manule di PvBrowser è stata tradotta e curata da Francesco Aria. Ringrazio mia moglie Stefania e mia figlia Francesca per la pazienza che hanno vauto.

Indice

Crediti

L'idea di pvbrowser fù sviluppata presso la SMS Siemag AG.
Alla SMS Siemag AG volevano un visualizzatore di processo che concedesse anche la disponibilità del codice sorgente , in modo di poter fare modifiche al programma , se necessario. Furono proposte soluzioni basate su Java e Qt. La soluzione basata su Java fù preferita dalla SMS Siemag AG ma lo sviluppo della soluzione basata su Qt fu avviata ugualmente come progetto open source con il nome di pvbrowser. Le caratteristiche principali di pvbrowser dovevano essere l'indipendenza dalla piattaforma e l'architettura rigorosamente client/server.
La variante Java non fù utilizzata per molto tempo ma in compenso pvbrowser fu presto utilizzato alla SMS Siemag AG. Dobbiamo ringraziare il personale della società per i test iniziali del software e per aver continuato ad utilizzarlo fino ad oggi. Senza il loro sostegno non avremmo potuto permetterci i costi di licenza per la versione commerciale di Qt.
Dopo la pubblicazione di pvbrowser la comunità di utenti e sviluppatori che contribuiscono allo sviluppo di pvbrowser è cresciuta. Senza questi utenti e sviluppatori non sarebbe stato possibile raggiungere il livello attuale dell'applicazione. Molte idee sono state sviluppate e molti errori sono stati eliminati.
Gli studenti e i loro professori che stanno facendo tesi di laurea in collaborazione con il nostro progetto sono apprezzati per il loro eccellente lavoro.
L'azienda Nokia/Trolltech devrebbe essere onorata di mettere Qt sotto licenza LGPL. Il personale di supporto di Nokia/Trolltech ci ha aiutato a risolvere i problemi più complicati. Inoltre siamo molto felici per i rapidi progressi di Qt .
pvbrowser utilizza alcuni componenti sviluppati da altri progetti open source. Ringraziamo tutti questi progetti da parte della comunità open source, senza la quale l'intera soluzione non sarebbe stata possibile.
Un nostro ringraziamento speciale va alle nostre famiglie che hanno avuto una grande pazienza con noi.

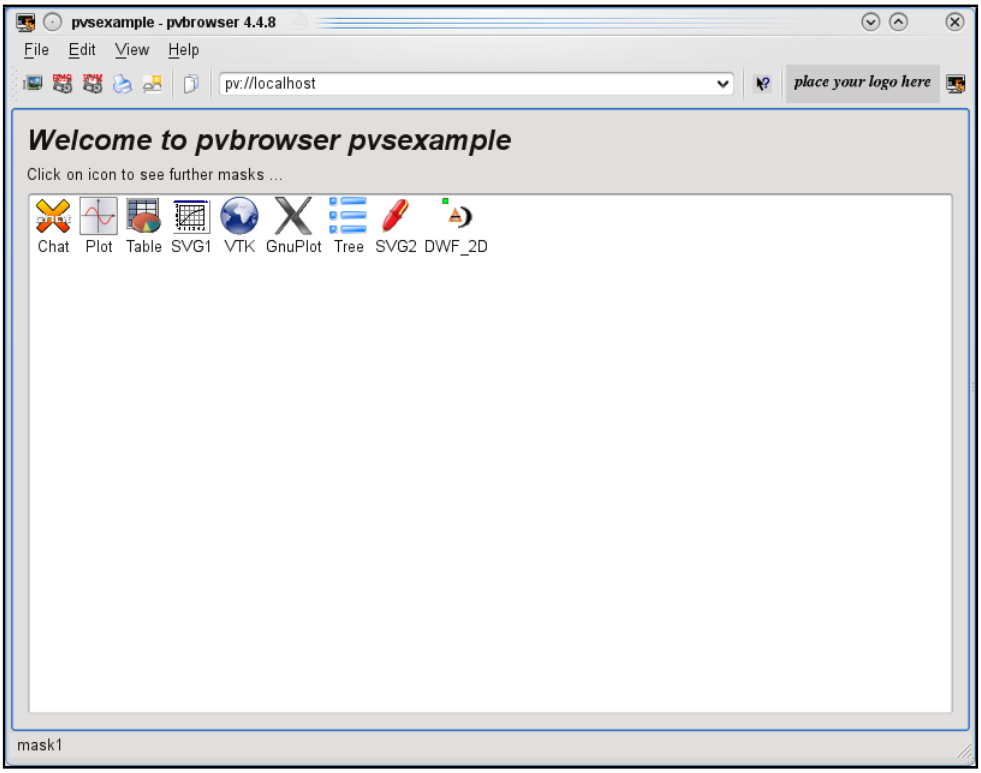

Figura 1: pvsexample maschera si avvio

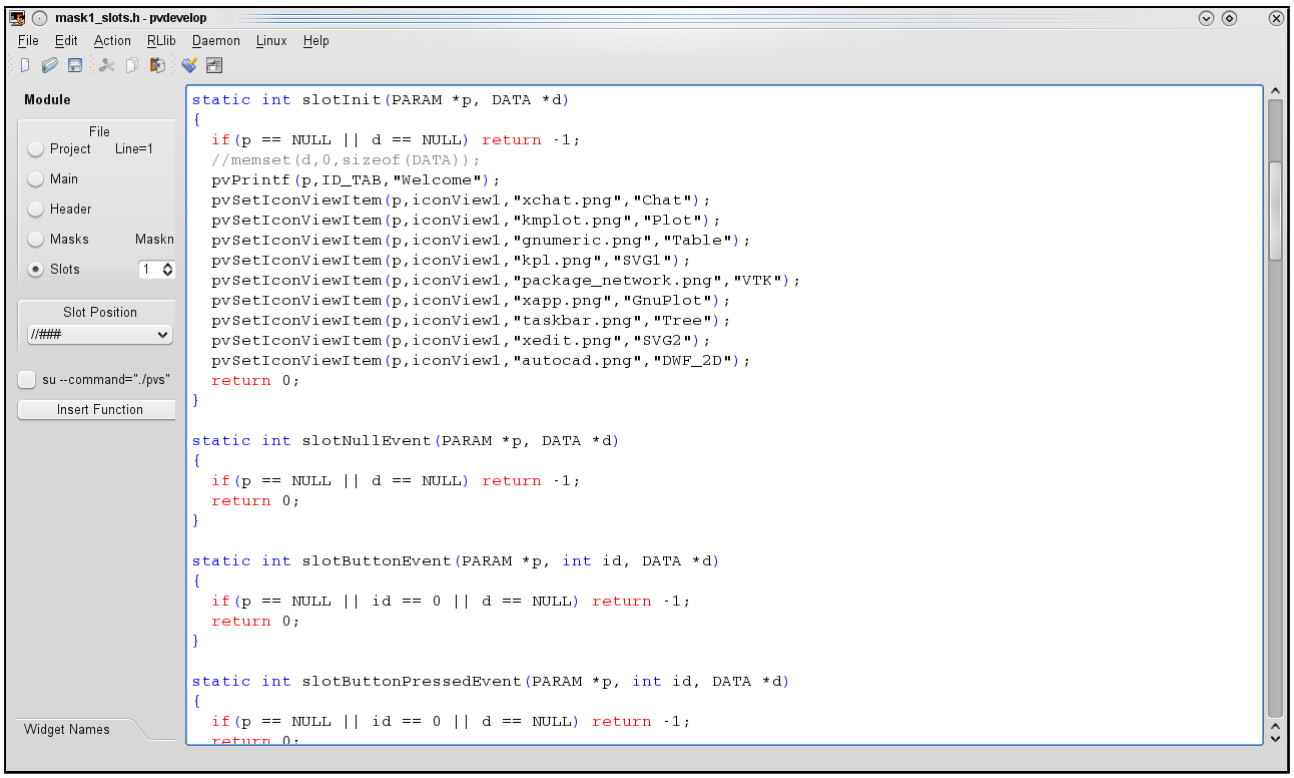

Figura 2: pvsexample maschera di avvio in pvdevelop

Prefazione

Cosa è uno SCADA ?

Da wikipedia: L'acronimo SCADA (dall'inglese Supervisory Control And Data Acquisition, cioè controllo di supervisione e acquisizione dati) indica un sistema informatico distribuito per il monitoraggio elettronico di sistemi fisici e processi. Il processo può essere industriale, infrastrutturale o basato sulle infrastrutture come descritto di seguito:

- I processi industriali sono quelli relativi alla produzione, la generazione di energia, fabbricazione, e la raffinatura , e possono funzionare in modo continuo, a lotto, ripetitivi, o in modo discreto.
- L'infrastruttura può essere pubblica o privata, e comprende il trattamento delle acque e la loro distribuzione, la raccolta delle acque reflue e trattamento, oleodotti e gasdotti, la distribuzione di energia elettrica , i sistemi di sirena per difesa civile, e i sistemi di comunicazione di grandi dimensioni.
- Gli impianti di processo si trovano sia in strutture pubbliche che in quelle private, compresi gli edifici, aeroporti, navi e stazioni spaziali. Questi impianti monitorano e controllano HVAC, gli accessi ed il consumo di energia.

File di progetto per un visualizzatore realizzato con pvbrowser. Utilizzando qmake viene generato un Makefile da questo 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       += /usr/lib/libpvsid.so
unix:INCLUDEPATH += /opt/pvb/pvserver
unix:LIBS       += /usr/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
```

Capitolo 1

Introduzione

pvbrowser è un software SCADA rilasciato sotto licenza GPL.
pvbrowser fornisce un framework per la visualizzazione di processo. Quelle riportate di seguito sono le caratteristiche di pvbrowser.

- Client/Server
- Componenti Qt
- Componenti personalizzati
- indipendente dalla piattaforma
- grafica SVG
- grfica xy
- grafica 3D
- Pagine WEB utilizzando WebKit
- supporto IDE
- progettazione grafica
- C/C++
- Lua, Python
- Multithreaded oppure Inetd
- supporto Unicode (Chinese, Arabic, Cyrillic, ...)
- Supporto di ssh-urls
- Connessioni a bus di campo
- Connessioni con PLC
- Controllo di processi di background
- Registrazione di eventi centralizzato
- È possibile codificare il proprio sistema di accesso e autorizzazione
- licenza GPL

Il client pvbrowser può essere paragonato ad un normale browser web. Tramite il pvbrowser non viene presentato nessun contenuto statico ma invece viene mostrato contenuto di tipo dinamico, come è necessario per un software SCADA. Con pvbrowser può essere realizzata qualsiasi maschera (pagina) di visualizzazione che può essere fruibile su tutto l'impianto oppure anche su Internet. Queste maschere (pagine) possono essere sfogliate utilizzando dei collegamenti ipertestuali.
La struttura delle finestre di pvbrowser è controllata dallo stesso pvbrowser , ma il contenuto delle stesse è completamente progettato dallo sviluppatore della maschera. La progettazione di queste maschere può essere fatta anche da computer remoti. Se una maschera (pagina) viene modificata non è necessario fare nessuna modifica sul computer client.
Le funzionalità ed il modo di operare di una finestra è determinato dalla realizzazione su misura della maschera. La finestra del client è dotata di un Help integrato. Un clic su ? accanto al campo dell'URL e poi un clic su di un'elemento nella barra degli strumenti farà aprire una guida contestuale. Il menu di aiuto in alto a destra è in realtà una pagina HTML a partire da index.html. Questa pagina deve essere scaricata dal server utilizzando il comando pvDownloadFile (). Se la pagina index.html non esiste non vi è alcun aiuto disponibile per la maschera (pagina) in funzione. Per l'help è possibile utilizzare tutte le funzionalità fornite da WebKit. Pvbrowser può essere personalizzato utilizzando l'editor in File Options. Si prega di notare che alcune variazioni vengono applicate solo dopo un riavvio del client. Sono disponobili le seguenti opzioni da riga di comando.

opzioni per pvbrowser da linea di comando

```
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
```

Il client pvbrowser si collegherà ad un pvserver via TCP/IP. Il pvserver inizierà l'invio di testo ASCII per il client che lo interpreterà ed il risultato porterà alla chiamata di un metodo Qt. Nella direzione opposta il client pvbrowser invierà messaggi di testo ASCII per il pvserver quando l'utente attiverà un evento come un clic su di un pulsante per esempio. Questo testo verrà interpretato all'interno di un ciclo di gestione degli eventi nel pvserver. Questo si tradurrà in una chiamata alla relativa 'slot function'. Il compito dello sviluppatore di una maschera di visualizzazione è quello di personalizzare queste 'slot function'. Lo scheletro del pvserver verrà generato e curato automaticamente da pvdevelop così lo sviluppatore potrà concentrarsi sulle 'slot function'. Il layout delle maschere verrà invece progettato graficamente.

Una tipica funzione in pvserver che invia del testo ASCII al client pvbrowser.

```
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;
}
```

Capitolo 2

Installazione

Pvbrowser funziona su piattaforma Linux / Unix / Windows e Mac OS-X. Un pvserver può anche funzionare su OpenVMS. `http://it.wikipedia.org/wiki/OpenVMS`

I sorgenti di pvbrowser sono inclusi nell'archivio `http://pvbrowser.de/pvbrowser/tar/pvb.tar.gz` . Scaricate questo archivio se volete compilarvi da soli pvbrowser. I pacchetti binari per Windows e OS-X sono disponibili sulla nostra homepage. I pacchetti binari per le più popolari distribuzioni Linux sono disponibili sul buildservice di openSUSE. C'è un link al buildservice sulla nostra homepage.

Inoltre vi è il package `http://pvbrowser.de/pvbrowser/tar/pvbaddon.tar.gz` , in cui troverete alcuni demo e modelli per pvbrowser. In particolare troverete alcuni esempi di acquisizione dati che utilizzano diversi protocolli.

Non descriveremo l'installazione dei pacchetti binari, perché questa operazione viene eseguita con la procedura standard di l'installazione relativa al proprio sistema operativo. Bisogna però tenere conto che sono necessari alcuni software aggiuntivi se volete sviluppare la vostra applicazione HMI o SCADA utilizzando l'IDE pvdevelop. Invece è possibile eseguire il client pvbrowser senza bisogno di installare questi pacchetti aggiuntivi. Solo le librerie Qt sono necessarie. Per OS-X installatele scaricandole dalla pagina della Nokia/Trolltech. Su Windows queste librerie sono già incluse nel pacchetto binario di installazione. Su Linux queste librerie dovrebbero essere già installate ma se non lo fossero dovrebbero venire installate automaticamente come dipendenze. Nei riquadro seguente vedrete come compilare e testare pvbrowser sul vostro sistema operativo.

Compilazione e test del software su Linux e 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
```

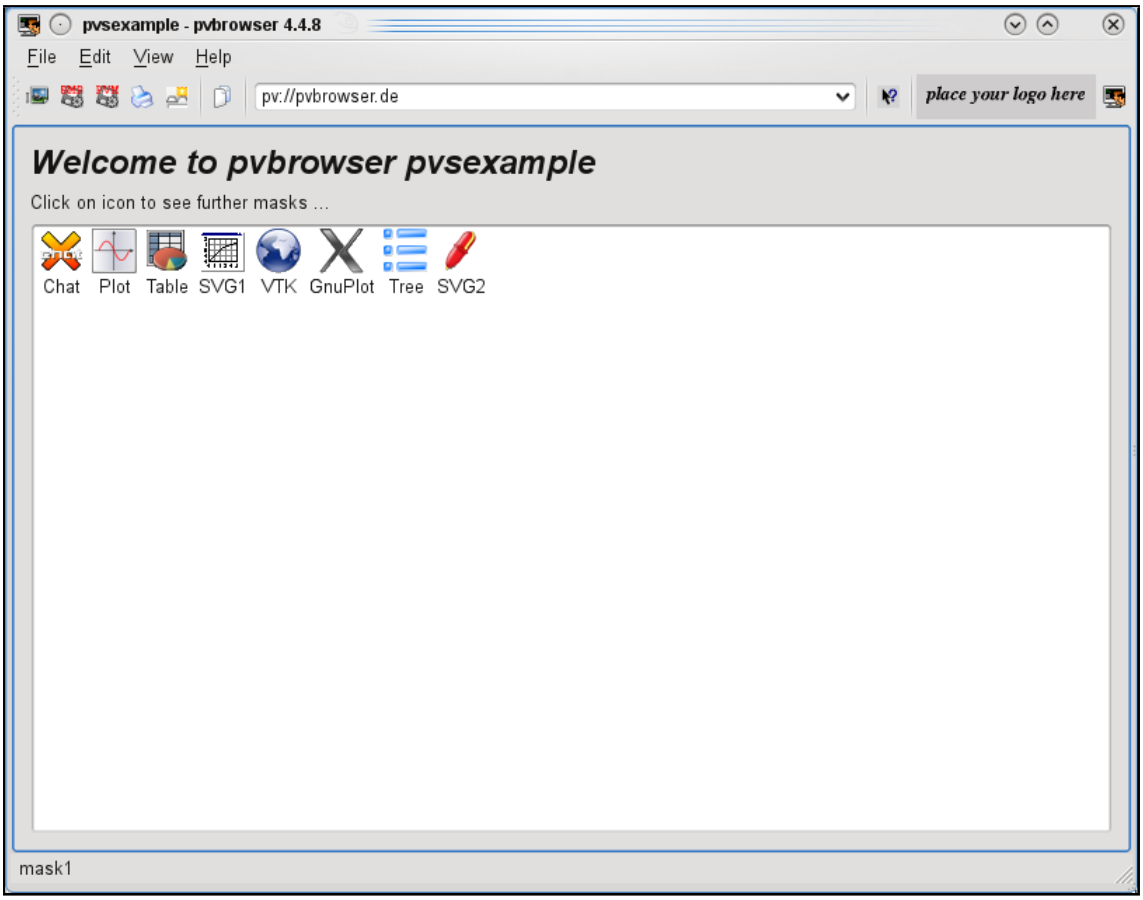

Figura 2.1: pvbrowser dopo una chiamata 'pvbrowser pv://pvbrowser.de'

Il file build.sh dovrà essere adattato in modo che possa trovare le librerie Qt sul vostro sistema. Editate build.sh e modificatelo secondo le istruzioni riportate al suo interno.

Compilazione e test su 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.
#
Scaricare e decomprimere http://pvbrowser.org/tar/pvb.tar.gz
Aprire una DOS-Box (shell comandi) e spostarsi nella directory pvb.
cd win-mingw
CreatePvbWithMinGW.bat
cd bin
pvbrowser pv://pvbrowser.de
```

Compilazione delle librerie per pvserver su OpenVMS

```
#
# the CXX C++ Compiler must be installed
#
Scaricare e decomprimere http://pvbrowser.org/tar/pvb.tar.gz
Spostarsi nella directory pvb.
@vms_build.com
```

Capitolo 3

pvbrowser client

Il client pvbrowser funziona come un browser web, ma oltre al protocollo http:// utilizza principalmente il suo protocollo pv:// ed il protocollo pvssh://. Il protocollo pv è orientato alla connessione e quando viene instaurata una connessione al server questa rimane in vita fino a quando la sessione non è terminata. Al contrario il protocollo http:// instaura una connessione poi legge la pagina web e dopo di che chiude immediamentamente la connessione. Quando si fa clic su di un link in una pagina web con il protollo http la connessione verrà nuovamente aperta. Ai fini della visualizzazione di processo un protocollo orientato alla connessione si adatta meglio perché si può fare un'autenticazione migliore e non si deve effettuare l'overhead del protocollo http://. Come con un browser web il client pvbrowser ha un campo di introduzione in cui è possibile inserire l'URL (indirizzo internet). Pvbrowser supporta i tre protocolli sopra riportati. Il protocollo http:// è supportato per mezzo di WebKit, che è integrato in Qt. È possibile collegare pagine che utilizzano questi protocolli per mezzo di link e navigare da una pagina all'altra , ovvero in una pagina WEB http possiamo introdurre un link ad un pagina pv o pvssh e viceversa.

Collegamenti tra maschere e siti Web

```
pvHyperlink(p,"pv://pvbrowser.org"); // Collegamento da un pvserver alla nostro server di test
pvHyperlink(p,"pvssh://user@pvbrowser.org"); // Collegamento utilizzando una shell sicura
pvHyperlink(p,"http://google.com"); // Collegamento da un pvserver ad un motore di ricerca Web

<a href="pv://pvbrowser.org">Cpllegamnto da un pvserver alla nostra homepage</a>
<a href="pvssh://user@pvbrowser.de">Collegamento da un pvserver usando una shell sicura</a>
```

La personalizzazione del client può essere eseguita modificando il file di configurazione che viene creato nella HOME directory (per windows directory dell'utente) . Il file può essere modificato direttamente dal pvbrowser scegliendo la voce Option nel menù File. Bisogna ricordarsi che alcune modifiche alla configurazione verranno applicate solo al riavvio del pvbrowser stesso.

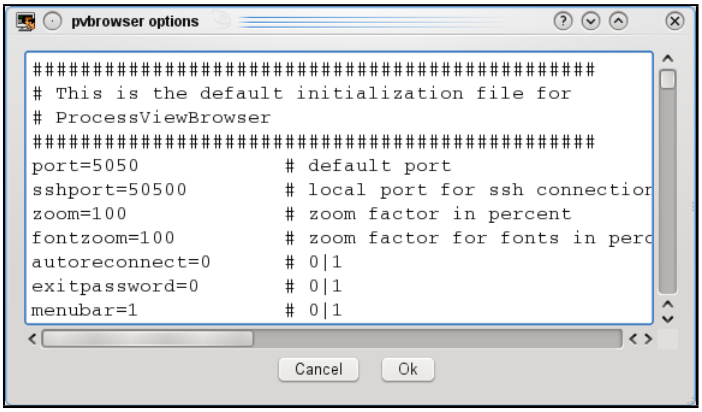

Figura 3.1: Finestra delle opzioni di pvbrowser

Configurazione del client pvbrowser

```
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 in pvbrowser

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

Esempi:
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
```

Capitolo 4

pvdevelop IDE

Pvdevelop è un IDE per lo sviluppo dei pvservers che permette l'uso dei seguenti linguaggi di programmazione C/C++ o Lua, Python. Quando si utilizza pvdevelop non è necessario preoccuparsi della creazione del file di progetto, del Makefile e dello scheletro di un pvserver. Il file di progetto, il Makefile e lo scheletro del pvserver vengono generati e aggiornati automaticamente. Si può facilmente navigare e modificare il codice sorgente e creare ed aggiornare graficamente le maschere di visualizzazione del vostro progetto.
In modalità editor è possibile visualizzare e modificare il file di progetto. Sul lato sinistro delle finestre è possibile vedere la toolbox. Usando la toolbox è possibile spostarsi tra i diversi file che costituiscono il vostro pvserver. Se si clicca su 'Insert Function' appare un albero di selezione da cui è possibile scegliere la funzione necessaria a svolgere un'operazione nel vostro pvserver. Si prega di notare che è necessario posizionare il cursore nella posizione in cui si desidera inserire la funzione. Se la casella degli strumenti selezionata è la 'Widget Names' verrà visualizzato un elenco dei widget che abbiamo inserito nel progetto. Se in questa lista viene scelto un'elemento si avrà la possibilità di selezionare ed inserire una delle funzioni utilizzate più frequentemente.
Nella modalità designer (progettazione) i widget possono essere progettati e posizionati graficamente. Per inserire un nuovo widget , nella modalità designer , bisognerà scegliere la voce insert new widget dal popupmenu che compare facendo un click destro sulla maschera in progettazione. Dopo l'introduzione di un widget è possibile posizionare e ridimensionare lo stesso con il mouse (lato in basso a destra del widget). Con la finestra di dialogo delle proprietà (fare un clic destro sul widget) è possibile impostare le proprietà del widget.
Nella modalità di progettazione il mouse è catturato dal designer ma facendo un clik destro è possibile scegliere dal menù contestuale cosa fare (grab mouse/release mouse). Se si sceglie 'release mouse' (rilasciare il mouse) è possibile testare la nostra maschera. Una volta rilasciato il mouse si può interagire con la maschera come se stesse funzionando nell'applicazione finita. Per continuare con la progettazione scegliere 'grab mouse'. All'interno del file projectname.pvproject è possibile definire le dimensioni massime delle maschere. Se si preferisce progettare il layout delle maschere con Qt Designer è possibile utilizzare la funzione di import/export per i file UI dal menu 'Action' nella modalità editor.
Lo sviluppo potrebbe anche essere fatto in un'altra IDE come Eclipse ad esempio. In modo particolare Qt Creator di Nokia/Trolltech è molto utile perché utilizza lo stesso formato per i file di progetto che usa pvdevelop. È possibile concentrarsi sulla codifica delle 'slot function' perchè il resto del codice sorgente di un pvserver è composto , generato e aggiornato da pvdevelop.
Siccome viene generato un Makefile per la compilazione , si può anche usare un qualsiasi editor ASCII per lo sviluppo e compilare dalla riga di comando usando make.

Compilazione da riga di comnado

```
user@yourbox:~/pvb/pvsexample> make
make: Nessuna operazione da eseguire per'primo'.
user@yourbox:~/pvb/pvsexample>
```

Con il menu di aiuto in pvdevelop è possibile aprire il manuale delle librerie usate per il controllo della finestra pvbrowser e per rllib utilizzata per la programmazione lato server.

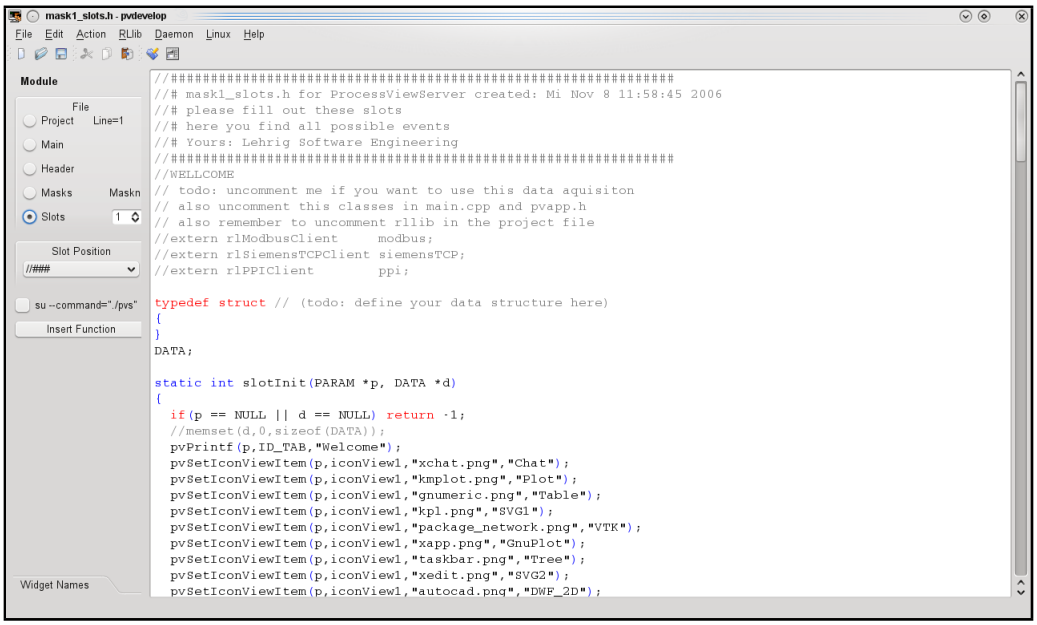

Figura 4.1: finestra principale di pvdevelop in modalità editor

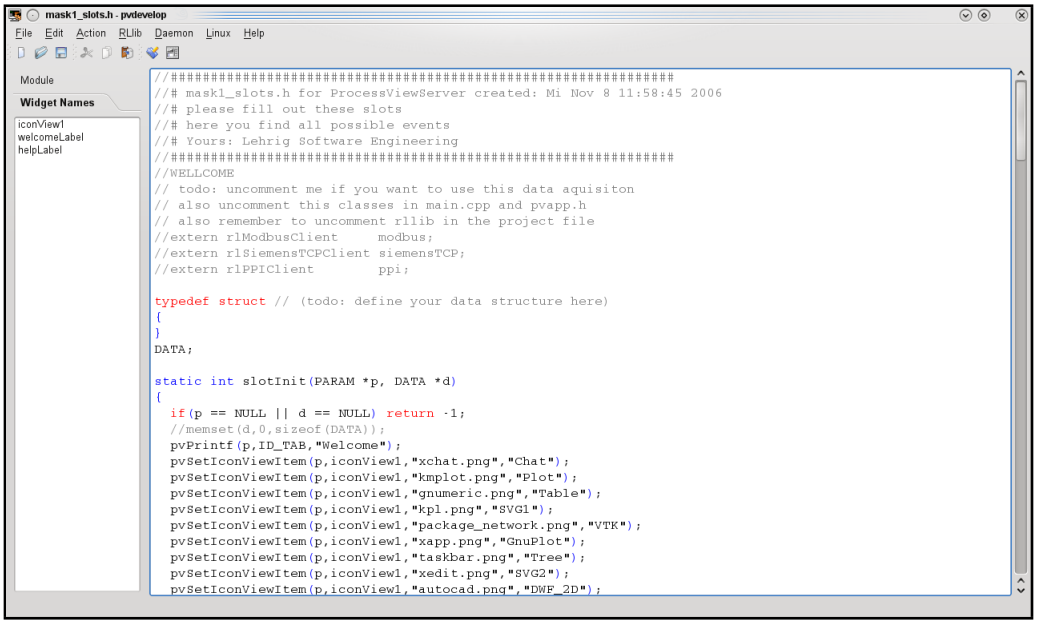

Figura 4.2: finestra principale di pvdevelop in modalità editor cona la toolbox selezionata

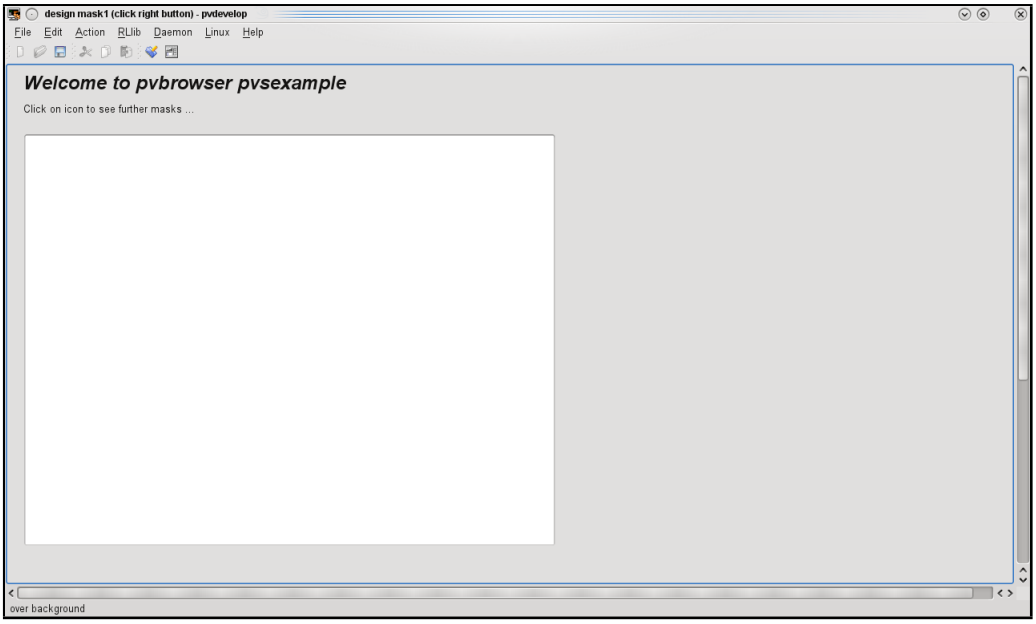

Figura 4.3: finestra principale di pvdevelop in modalità designer(progettazione)

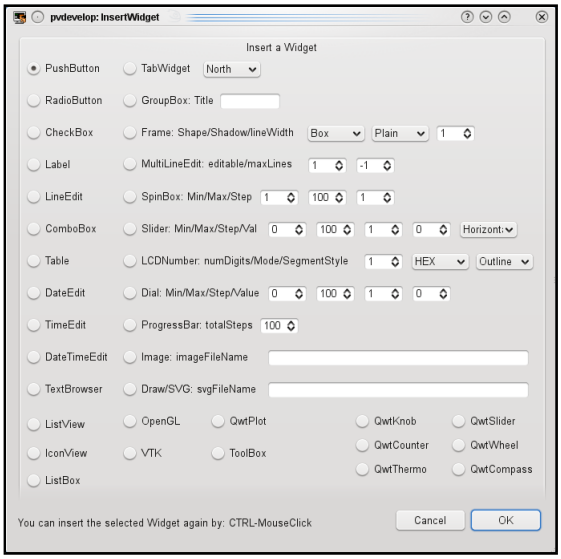

Figura 4.4: finestra di dialogo per l'inserimento dei widget

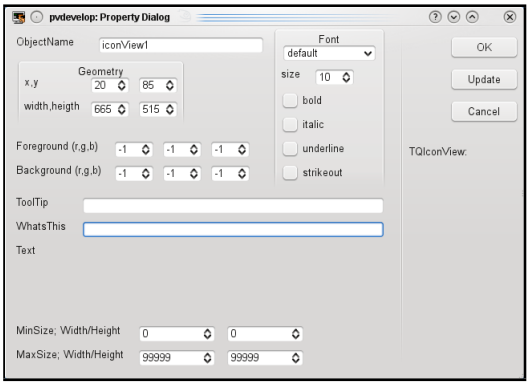

Figura 4.5: finestra di dialogo per le proprietà dei widget

Capitolo 5

Usare Qt Creator come IDE

Qt Creator è un tool ideale per programmare le vostre applicazioni pvservers perchè utilizza i file pro anche da qmake.
Per facilitare il vostro lavoro potete creare alcuni script di shell nella directory '~/bin' che vi aiuterenno automatizzando alcune operazioni.
Il primo script di shell crea un nuovo progetto e avvia Qt Creator.

~/bin/pvnew

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

Il secondo script di shell importa una maschera da Qt Designer 'maskN.ui' nel pvserver.

~/bin/pvimport

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

Il terzo script di shell aggiunge una maschera ad un progetto esistente.

~/bin/pvinsert

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

Ora create una nuova directory e lanciate in una shell il comando 'pvnew'. Verrà creato un nuovo pvserver e verrà avviato Qt Creator con su questo nuovo progetto.

Creare un nuovo pvserver e avviare 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 si dovrà accettare l'importazione delle impostazioni date. In 'Projects->Run Settings->Show Details' perfavore selezionate 'eseguire in un terminale'('execute in terminal') perchè così il pvserver scriverà i messaggi nella console che altrimenti non sarebbero visibili. Questi messaggi sono molto utili nella fase di debug. Ora in Qt Creator aggiungete un nuovo Qt Designer form. Perfavore selezionate il tipo 'Widget' e chiamate la form 'mask1.ui' e aggiungetela al progetto. Ora potete progettare la vostra maschera e memorizzarla. Ora , da una shell (termiale) , avviate il comando 'pvimport 1' per importare la form nel vostro pvserver. Ora potete eseguire il vostro pvserver. Quando avvierete il client pvbrowser sarete in grado di connettervi al nuovo pvserver. Su Windows i passi da eseguire sono molto simili ma bisognerà adattare gli script di shell in modo da trasformarli in batch files.
Commenti:
Per poter utilizzare i widget speciali di pvbrowser all'interno di Qt Designer bisognerà copiare questi plugins all'interno della directory dei plugin di Qt Designer.
Questo puo essere eseguito per mezzo dei seguenti comandi che sono leggermente differenti per i sistemi linux a 32 e 64 bit.

Copia dei plugins

```
cp /opt/pvb/designer/plugins/* /usr/lib/qt4/plugins/
cp /opt/pvb/designer/plugins/* /usr/lib64/qt4/plugins/
Windows:
copy %PVBDIR%/win-mingw/bin/plugins/designer/*.dll
C:\Qt\2010.05\qt\plugins\designer
```

In windows il path della directory dei plugin di QT Designer può variare a seconda della versione che avete installato oppure se avete modificato in fase di installazione il path. Comunque dovrebbe essere qualcosa del genere : C:010.05
Ovvero la parte finale del comando del copia , ovviamente se avete installato una versione diversa dell SDK QT o avete modoficato le opzioni di installazione dovrete corregere questo path.

Capitolo 6

Programmazione

Se si sceglie 'File->new pvserver' in pvdevelop vi verrà chiesto il nome e la directory di creazione per il vostro pvserver. Dopo di che il designer integrato diventerà visibile e utilizzato il popup menu disponibile con un clic destro del mouse sarà possibile definire il layout delle vostre maschere. Quando avremo finito dovremo salvare il layout e tornare alla modalità editor.
Lo scheletro del nuovo pvserver verrà generato. Si prega di dare un'occhiata al codice sorgente utilizando pvdevelop per comprendere la struttura del nostro pserver.
Dopo questi primi passi è possibile avviare la nostra applicazione.
'Action->start server' e 'Action->start browser' sono i comandi di base.
Il vostro compito è ora quello di definire la struttura dei dati DATA e di scrivere il codice delle mask∗_slots.h al fine di definire la logica del vostro pvserver.

6.1 Strutura di un pvserver

Un pvserver è composto dalle parti che vengono descritte di seguito. Vi è un file di progetto in cui sono definiti i file sorgenti e le librerie del pvserver.
C'è un file main nel quale si possono trovare due diverse versioni di main() una per inetd e una per il Multi-threaded e che sono selezionate da un simbolo #ifdef per il preprocessore. Di default è attiva la scelta per il server multi-threaded. Se si desidera avviare il pvserver dal Superserver xinetd è necessario decommentare USE_INETD all'interno del file di progetto e selezionare 'libpvsid' invece di 'libpvsmt' come libreria pvbrowser.
pvMain è la funzione principale per servire un client. In questa subroutine vi è un loop in cui sono richiamate le maschere disponibili. Il valore di ritorno di una maschera determina quale sarà la prossima maschera che verrà mostrata.
C'è un file intestazione (headers) che è incluso in tutti i sorgenti di un pvserver.
Ci sono tante maschere quante ne sono stato progettate. Il codice sorgente viene generato automaticamente da pvdevelop. Normalmente non è necessario preoccuparsi dello scheletro (struttura) dell'applicazione.
Esistono tanti mask_slots quante sono le maschere progettate. Questo file di intestazione è incluso all'interno della corrispondente maschera. In questo file viene definita la struttura locale dei dati e le funzioni di slot.

6.1.1 File di progetto per qmake

Dal file di progetto qmake in Qt SDK può generare un Makefile. All'interno di tale file di progetto sono specificati tutti i sorgenti e le librerie utilizzate dal vostro pvserver. Si prega di notare che il file di progetto seleziona automaticamente le librerie per il sistema operativo che si sta utilizzando. Se si desidera utilizzare librerie aggiuntive all'interno del vostro pvserver è possibile specificarle all'interno del file di progetto.

file di progetto

```
######################################################################
# 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       += /usr/lib/libpvsid.so
unix:INCLUDEPATH += /opt/pvb/pvserver
unix:LIBS       += /usr/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 Funzione main

La funzione main() è differente a seconda se si vuole utilizzare il MULTI THREADED o la la versione INETD. La versione INETD viene selezionate per mezzo del simbolo del preprocessore USE_INETD .

Se un nuovo client si collega al pvserver main() avvierà un nuovo thread che richiamerà pvMain () per servire il client. Nel caso di USE_INETD pvMain() sarà chiamato direttamente.
In pvMain () si può per prima cosa valutare l'URL che il client utilizza. Poi pvMain() esegue un ciclo senza fine in cui le diverse maschere che sono state progettate da voi verranno richiamate. Il valore restituito da una maschera determinerà quale maschera sarà chiamata successivamente. Se il client termina la connessione il thread con pvMain() sarà terminato automaticamente. E 'possibile impostare un puntatore a una funzione cleanup() all'interno della struttura PARAM se è necessario per liberare le risorse quando il pvserver è terminato. Normalmente questo operazione non dovrebbe essere necessaria.
La funzione getParams() mostrata nel codice sorgente è stata inserita dallo sviluppatore di pvsexample e non è stato generata automaticamente ma il resto dello scheletro è stato completamente generato da pvdevelop.

funzione main

```
//****************************************************************************
//                       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 Maschere

Dal ciclo senza fine all'interno di pvMain() vengono richiamate le singole maschere che compongono la vostra visualizzazione. Mostreremo ora la maschera iniziale di pvsexample.
In show_mask1() vi è un ciclo di eventi. Gli eventi sono analizzati e la corrispondente 'slot functions' in cui dovrete inserire il vostro codice sarà richiamata. Questo file non dovrà mai essere modificato manualmente perchè sarà un compito svolto automaticamente da pvdevelop.
Nella funzione genarted_defineMask() i comandi per i widget che avete progettato sono inviati al client pvbrowser. pvbrowser analizzerà i comandi e chiamerà i costruttori appropriati per questi widget.
Tutte le 'pv-function' utilizzanto la struttura PARAM come primo parametro. All'interno di un elemento di questa struttura si trova il socket utilizzato per comunicare con il client. Le 'pv-function' inviano messaggi di testo ASCII al client che interpreterà il testo e chiamarà un metodo dalla libreria Qt.
L'enum all'inizio del file elenca i nomi widget che avete progettato. Questi nomi sono utilizzati come id (identificatori) all'interno delle 'pv-function' e indirizzano al widget all'interno del client. Per questo ragione vi è un array con i puntatori ai widget all'interno del client. L'enum è un indice di questo array.
Le 'slot functions' sono incluse con un #include nella maschera.

codice sorgente della parte completamente autogenerata di una maschera

```
////////////////////////////////////////////////////////////////////////////
//
// 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

Le 'slot functions' devono essere codificate dallo sviluppatore della maschera. All'inizio del file vi è la definizione di una struttura DATA. All'interno di questa struttura lo sviluppatore può definire i suoi valori che sono di tipo privato. Questo può essere paragonato ad una classe C++ anche se quì è stata scritta in ANSI C.

Le funzioni slot che devono essere codificate dallo sviluppatore

```
//####################################################################
//# 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(0)
  {
    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

Il file di intestazione pvapp.h è incluso in ogni maschera . In questo file si possono definire alcune cose utili. La variabile trace può attivare/disattivare la funzione printf() di output del pvserver. Durante la fase di sviluppo questi messaggi di output sono utili, bisogna quindi lasciare attiva la funzione di trace. Se si usa la modalità inetd bisogna invece disabilitare questa funzione perché questi messaggi verrebbero inviati al client pvbrowser, invece di essere stampati a schermo nella shell, e questo potrebbe disturbarne il funzionamento.

file di intestazione pvapp.h che è incluso in tutte le maschere

```
//************************************************************************
//                      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 struttura PARAM

La struttura PARAM è utilizzata come primo parametro di tutte le 'pv-functions'. Si prega di dare un'occhiata a questo file e vedere quali informazioni possono essere trovate al suo interno.

La struttura PARAM descrive la connessione con il 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 */
}PARAM;
```

6.2 Programmazione degli slot

Nelle sezioni precedenti abbiamo già visto le 'slot function'. Lo scheletro dell'applicazione è già stato generato da pvdevelop. Ora è vostro compito codificare le 'slot functions' al fine di definire la logica della vostra visualizzazione.
La funzione slotInit() è responsabile dell'inizializzazione delle variabili all'interno della struttura DATA.
La funzione slotNullEvent() viene richiamata ciclicamente nell'intervallo definito in (PARAM *) p->millisecondi di attesa(sleep). È possibile inserire le operazioni che devono essere eseguite continuamente all'interno di questa funzione come l'aggiornamento di un valore numerico della maschera.
Le altre 'slot function' vengono richiamate quando l'utente attiva un evento nel pvbrowser. Ad esempio quando si clicca su di un pulsante.

Esempio di una 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);

  pvHtmlOrSvgDump(p,upperWidget,"dump.html");
  pvClientCommand(p,"html","dump.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);
  }
  return 0;
}
```

6.3 Funzioni Util

Le 'Util functions' sono usate per controllare la finestra del pvbrowser dal pvserver. Queste funzioni invieranno testo ASCII per il client pvbrowser che verrà interpretato ed il risultato sarà una chiamata ad un metodo di Qt. Nel file mask∗.cpp che è stato generato da pvdevelop i costruttori per i widget che avete progettato sono richiamati in 'generated_defineMask()'. Il nome delle 'util function' inizia con 'pv'. Il primo parametro delle 'Util function' è (PARAM ∗p) che descrive il collegamento al client pvbrowser. Il secondo parametro della maggior parte delle 'Util function' è un 'id' che fa riferimento ad un widget. L'id è uno dei nomi che sono elencati nella enum all'inizio del file mask∗.cpp e che fa riferimento ai widget che avete progettato.
Il manuale di riferimento delle 'Util function' è sulla nostra home page e anche nel Manuale di aiuto all'interno pvdevelop. Si prega di consultare soprattutto l'argomento 'Construction' perché vi si possono trovare tutte le funzioni che sono disponibili per ogni widget.

6.4 rllib

La libreria rllib permette la programmazione in modo indipendente dalla piattaforma di molte funzioni di sistema sul lato server. Vi si trovano anche numerose classi per il supporto della comunicazione con i PLC e protocolli fieldbus. Per poter utilizzare rllib all'interno del pvserver selezionare il menu 'RLlib->Uncomment rllib' in pvdevelop.

- *rl3964R* implementa il protocollo Siemens 3964R (Dust)
- *rlbussignaldatabase* classe per leggere/scrivere variabili di processo in un database MySQL.
- *rlController* Questa classe implementa il controllo ad anello chiuso secondo 'F. Doerrscheid/W. Latzel, Grundlagen der Regelungstechnik, B.G. Teubner Stuttgart, Regelalgorithmen mit der Trapezregel'
- *rlCorbaClient* usata per incapsulare comunicazioni corba.
- *rlCorbaServer* usata per incapsulare comunicazioni corba.
- *rlcutil* contiene alcune semplici funzioni ANSI C.
- *rlDataProvider* contenitore per le varibili di processo.
- *rlDataProviderClient* client per accedere alla variabili di processo che sono rese disponibili sulla rete tramite un server utilizzando *rlDataProviderThreads*.
- *rlDataProviderThreads* Fornisce contenitori *rlDataProvider* sulla rete. Per questo viene avviato un nuovo thread che gestisce un numero illimitato di *rlDataProviderClient*.
- *rlDataAcquisition* è utilizzato per l'acquisizione di dati secondo i 'principi di pvbrowser'. Da un'applicazione pvserver o addirittura da altre applicazioni è possibile accedere alla memoria condivisa e alla mailbox utilizzata per lo scambio dati con il demone che si collega al PLC o ai sistemi di bus di campo. Questa classe codifica le variabili di processo come testo ASCII leggibile. Si veda anche *rlDataAquisitionProvider*.
- *rlDataAcquisitionProvider* Questa classe viene utilizzata sullo stesso lato dove risiede il demone per la scrittura di variabili di processo nella memoria condivisa e in attesa di comandi di scrittura comandi nella mailbox. Si veda anche *rlDataAquisition*
- *rlEIBnetIp* implementa il protocollo EIBnetIp (European Installation Bus) su rete.
- *rlEventLogServer* è utilizzato per implementare un server che legge i messaggi del registro eventi da altri processi in rete. Tale pvserver è già disponibile in 'pcontrol' (vedi pvbaddon).
- *rlEventLogServerThreads* è utilizzato per implementare un server che legge i messaggi del registro eventi da altri processi in rete. Tale pvserver è già disponibile in 'pcontrol' (vedi pvbaddon).
- *rlFifo* buffer first in first out per comunicazioni tra differenti threads.
- *rlFileLoad* carica un file ASCII che può essere iterato in RAM.
- *rlHilscherCIF* Wrapper per i della Hilscher CIF cards (Profibus, CAN, ...).

- *rlHistoryLogger* è utilizzato per le registrazioni storiche su file ASCII e in RAM.
- *rlHistoryReader* è utilizzata per leggere le rigistrazioni storiche che sono state generate con *rlHistoryLogger*.
- *rlIniFile* implementa gli INI files come su Windows. Si possono leggere,scrivere e manipolare questi file. Inoltre è possibile utilizzare il metodo i18n per tradurre il testo in diverse lingue.
- *rlInterpreter* è utilizzato per l'implementazione di un interprete per la linea di comando.
- *rlIpAdr* Indirizzi IP address per l'uso con *rlUdpSocket*
- *rlMailbox* implementa una cassetta postale che è un meccanismo per la comunicazione tra processi sullo stesso computer in cui più processi possono scrivere e solo un processo è può leggere
- *rlModbus* implementa il protocollo modbus. Modbus RTU, Modbus ASCII su porta seriale e Modbus TCP sono supportati.
- *rlModbusclient* Classe da utilizzare in un pvserver per l'accesso alla memoria condivisa e alla cassetta postale con un modbusdaemon che è stato generato da pvdevelop. In questo caso le variabili di processo sono codificate binarie
- *rlMutex* implementa un mutex.
- *rlOpcXmlDa* implementa un client per OPC XML/DA protocollo (HTTP / SOAP / XML). In pvbaddon potete trovare un demone che si basa su questa classe.
- *rlPcontrol* Provides a class for starting and controlling processes.
- *rlPlcMem* Implementa le variabili per un soft PLC.
- *rlPlcState* Implementa arrays di variabili per un soft PLC.
- *rlPPIClient* Implementa il protocollo Siemens PPI utilizzando libnodave.
- *rlSerial* Classe per linee seriali (RS232/RS485)
- *rlSharedMemory* Implementa una memoria condivisa (RAM che viene condivisa tra più processi).
- *rlSiemensTCP* Implementa il protocollo per PLC Siemens per le serie S7 e S5.
- *rlSiemensTCPClient* Classe da utilizzare in un pvserver per l'accesso alla memoria condivisa e alla cassetta postale con un siemensdaemon che è stato generato da pvdevelop. In questo caso le variabili di processo sono codificate binarie.
- *rlSocket* Socket per comunicare su TCP utilizzando IPv4 e IPv6.
- *rlSpawn* Avvia un altro processo e collega è STDIN e STDOUT su una Pipe per mezzo di questa classe. Questa classe può essere usata per controllare le applicazioni a riga di comando da una interfaccia utente di tipo grafico. ATTENZIONE: Questa classe è disponibile solo sui sistemi operativi Unix.
- *rlSpreadsheetCell* Cella di una tabella.
- *rlSpreadsheetRow* Riga di una tabella.
- *rlSpreadsheetTable* Tabella. Le tabelle possono leggere e scrivere file CSV.
- *rlSpreadsheetWorkbook* Diversi tabelle. Cartella di lavoro in grado di leggere e scrivere una serie di File CSV..
- *rlString* Semplice classe di tipo string.
- *rlSvgAnimator* Classe per l'animazione grafica SVG all'interno di un pvserver.
- *rlSvgCat* Classe per la 'Normalizzazione' del codice XML per SVG. I singoli tag sono suddivisi per linee separate e scritto con giustificazione a sinistra su STDOUT.

- *rlSvgPosition* Utilizzato insieme a *rlSvgAnimator*. Esso rappresenta la posizione di oggetti grafici all'interno della grafica SVG.
- *rlThread* Wrapper per thread basati su pthread rispettive routine di thread di Windows.
- *rlTime* implementa time e date.
- *rlUdpSocket* Comunicazione di rete con protocollo UDP.
- *rlWebcam* Implementa un client per le webcam Motion JPEG che sono collegate su HTTP.
- *rlwthread* Funzioni C per incapsulare i threads.

Il manuale di riferimento di rllib lo si può trovare sul nostro sito e all'interno del manuale di aiuto in pvdevelop.

6.5 Lua

Lua `http://www.lua.org/` è un linguaggio di scripting da incorporare all'interno di programmi scritti in C/C++. Nel nostro pvserver viene creato un nuovo thread per ogni client che si collega. La gestione del client inizia nella funzione 'pvMain()'. Quando si utilizza Lua viene richiamato lo script Lua 'main.lua' con la funzione 'luaMain()'. La gestione completa del client può ora essere codificata utilizzando il linguaggio Lua. Le nostre librerie sono 'wrappate' (impacchettate) con Swig `http://swig.org/` per poterle utilizzare da Lua. Un vantaggio di Lua è la piccola dimensione della libreria che si utilizza per implementare il linguaggio. Questo consente di incorporare la libreria staticamente nel programma principale. L'utente finale non avrà la necessità di installare Lua perchè questo è già integrato. E' possibile sviluppare i vostri pvservers senza installare pacchetti aggiuntivi. Non avrete bisogno di un compilatre C/C++ e neanche della SDK Qt.
Un'ulteriore vantaggio di questa soluzione è che potete modificare il codice del vostro pserver mentre questo è in esecuzione. Ogni nuovo client che si collegherà vedrà il codice Lua senza la necessità di riavviare il server. Questo può essere paragonato all'utilizzo del linguaggio PHP all'interno di un web server.
Quando si crea un nuovo progetto con pvdevelop è necessario scegliere il linguaggio di programmazione che vogliamo utilizzare. Se si seleziona 'Lua' pvdevelop creerà l'intera struttura degli script Lua. La parte di script che è dedicata alla progettazione della maschera è completamente generata da pvdevelop. Le 'slotFunctions' vengono anche generate. Il programmatore delle visualizzazioni si può così concentrare sulla codifica delle 'slotFunctions'.
Nella directory pvbaddon 'pvbaddon/demos/lua/' si trovano degli esempi in linguaggio Lua. Qui vi mostriamo un pserver 'Hello World' in Lua che dimostra l'uso del protocollo Modbus e di un sistema di Database SQL.

6.5.1 main.lua

Il programma main in Lua

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

Il programma principale 'luaMain()' è richiamato dal nostro programma principale 'pvslua' scritto in C/C++ quando un nuovo client si collega con 'pvslua'. La gestione del client inizia in 'luaMain()'.
Come si può vedere dal codice richiamiamo le funzioni delle nostre librerie C/C++ da Lua. In 'main.lua' definiamo una classe globale per l'acquisizione dei dati dalla nostra memoria condivisa e mailbox ed una classe per un database MySQL. La classe database è basata sulle classe database Qt. In questo modo sono supportati una vasta gamma di database.

Database supportati

```
dbtype := Descrizione
"QDB2"    IBM DB2, v7.1 e superiore
"QIBASE"  Borland InterBase Driver
"QMYSQL"  MySQL Driver
"QOCI"    Oracle Call Interface Driver
"QODBC"   ODBC Driver (incluso Microsoft SQL Server)
"QPSQL"   PostgreSQL v6.x e v7.x
"QSQLITE" SQLite versione 3 e superiore
"QSQLITE2" SQLite versione 2
"QTDS"    Sybase Adaptive Server
```

Dopo questo vi è un ciclo in cui tutte le maschere della nostra visualizzazione vengono richiamate. Attualmente vi è solo una maschera.

6.5.2 maskN.lua

Codice per la visualizzazione di una maschera

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

Il codice per le maschere della visualizzazione viene creato da pvdevelop. All'interno del ciclo degli eventi vengono richiamate le 'slotFunctions'. Le 'slotFunctions' devono essere codificate dal programmatore della visualizzazione.

6.5.3 maskN_slots.lua

slotFunctions per una maschera della visualizzazione

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

Nelle 'slotFunctions' il programmatore codifica la logica di funzionamento del pvserver.
L'esempio mostra il collegamento al Modbus con la nostra memoria condivisa e con la mailbox. Inoltre vediamo una query al database. I commenti nel codice dovrebbero essere sufficienti per capirne il funzionamento.

6.5.4 modbus.ini

File INI per il demone modbus_client

```
# 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)
```

Il demone 'modbus_client' utilizza un file INI che definisce cosa deve essere letto dal Modbus,che nome hanno la memoria condivisa e la mailbox e quale interfaccia deve essere utilizzata.

Avvio del modbus_client

```
modbus_client modbus.ini
```

6.6 Python

Per Python c'è un binding per le 'Util function' e per rllib che viene creato con Swig `http://swig.org`. In questo modo gli sviluppatori Python possono usare queste librerie.
In pvdevelop è possibile scegliere di utilizzare Python quando si crea un nuovo pvserver. Pvdevelop genererà uno scheletro in C++ che chiamerà le 'slot function' scritte in Python. Lo sviluppatore può codificare in Python e usare le librerie di cui sopra.

slot functions in Python

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

class mask1:

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

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

  whatsThis = [
    '',
    '',
    '' ]

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

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

```
I = 0

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
    return 0

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

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

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

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

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

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

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

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

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

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

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

6.7 Widgets

Nelle sezioni seguenti viene mostrato come programmare i singoli widget. Le seguenti 'Util functions' sono disponibili per tutti i tipi di widget.

- *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);*

Le altre 'Util functions' fanno riferimento solo ai singoli widget. Questi li potrete trovare sotto 'Construction' all'interno dell'help su pvslib sotto ciascun widget.

6.7.1 PushButton

I PushButton possono avere icone addizionali.

Figura 6.1: PushButton

Forniscono gli eventi nei seguenti slot.

- *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

Alcuni RadioButton lavorano esclusivamente se condividono lo stesso widget genitore.

Figura 6.2: RadioButton

Forniscono gli eventi nei seguenti 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

I CheckBoxes possono essere contrassegnati come spuntati.

Figura 6.3: CheckBox

Forniscono gli eventi nei seguenti 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

Le Labels sono utilizzate per l'output di testo su di una sola linea.

Figura 6.4: Label

Forniscono gli eventi nei seguenti 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

Le LineEdits sono utilizzate per per inserire una linea di testo.

Figura 6.5: LineEdit

Forniscono gli eventi nei seguenti 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

Le MultiLineEdits sono utilizate per inserire diverse linee di testo. Le MultiLineEdit possono essere impostate in sola lettura.

Figura 6.6: MultiLineEdit

Forniscono gli eventi nei seguenti 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

Le ComboBoxes sono utilizzate per seleziona un'opzione tra molte. Le ComboBoxes possono essere editabili.

Figura 6.7: ComboBox

Forniscono gli eventi nei seguenti 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

Un LCDNumber può visualizzare valori come un display LCD.

Figura 6.8: LCDNumber

Fornisce gli eventi nei seguenti 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

Gli Slider sono utilizzati per l'inserimento di un valore in modo analogico.

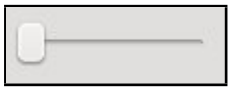

Figura 6.9: Slider

Forniscono gli eventi nei seguenti 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

Un Frame può essere utilizzato per circondare e raggruppare altri widget.

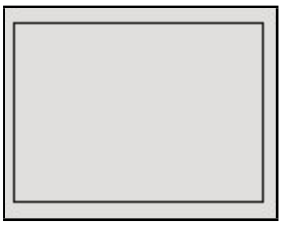

Figura 6.10: Frame

Fornisce gli eventi nei seguenti 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

Un GroupBox raggruppa diversi widgets e predispone un titolo.

Figura 6.11: GroupBox

Fornisce gli eventi nei seguenti 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

Un ToolBox fornisce diverse schede con le quali l'utente può passare tra le diverse aree.

Figura 6.12: ToolBox

Fornisce gli eventi nei seguenti 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

Un TabWidget è utilizzato per organizzare diverse sotto finestre che l'utente può selezionare.

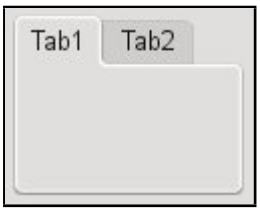

Figura 6.13: TabWidget

Fornisce gli eventi nei seguenti 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

Una ListBox fornisce una selezione di diverse voci.

Figura 6.14: ListBox

Fornisce gli eventi nei seguenti 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

Una Table è utilizzata per inserire o visualizzare dati tabulari.

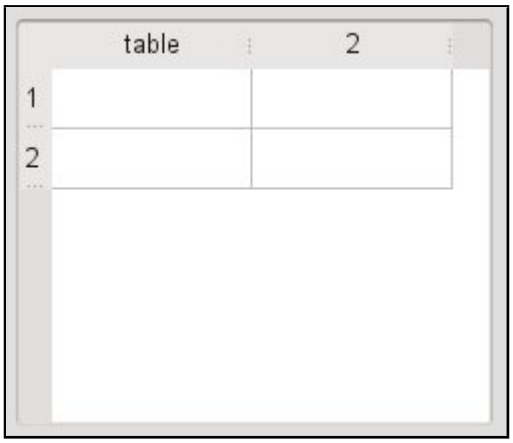

Figura 6.15: Table

Fornisce gli eventi nei seguenti 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

Lo SpinBox è utilizzato per l'input e l'output di valori che sono limitati da un valore minimo e massimo.

Figura 6.16: SpinBox

Fornisce gli eventi nei seguenti 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

Un Dial viene utilizzato per l'imput e l'output di valori.

Figura 6.17: Dial

Fornisce gli eventi nei seguenti 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

Una Line è utilizzata per l'organizzazione grafica.

Figura 6.18: Line

Fornisce gli eventi nei seguenti 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

Una ProgressBar può essere utilizzata per visualizzare lo stato di azioni di lunga durata.

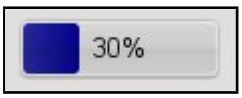

Figura 6.19: ProgressBar

Fornisce gli eventi nei seguenti 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

Una ListView implementa una vista ad albero.

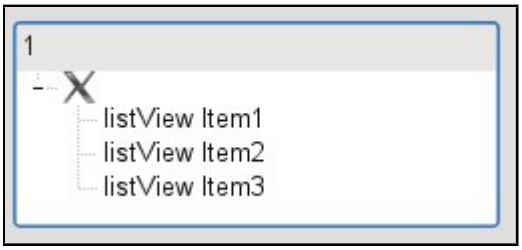

Figura 6.20: ListView

Fornisce gli eventi nei seguenti 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

Un IconView permette di effettuare la scelta di un'elemento utilizzando le icone.

Figura 6.21: IconView

Fornisce gli eventi nei seguenti 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 permette l'integrazione di qualsiasi file HTML o di una pagina web utilizzando un URL http per mezzo di WebKit. Un clic su un collegamento ipertestuale causa lo scatenarsi di un TextEvent con tale URL.

Figura 6.22: TextBrowser/WebKit

Fornisce gli eventi nei seguenti 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 è utilizzato per l'inserimento di date è ora.

Figura 6.23: DateTimeEdit

Fornisce gli eventi nei seguenti 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 viene utilizzato per inserire date.

Figura 6.24: DateEdit

Fornisce gli eventi nei seguenti 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 è utilizzato per inserire ore.

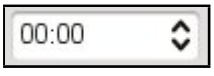

Figura 6.25: TimeEdit

Fornisce gli eventi nei seguenti 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

Un QwtThermo può essere utilizzato per visualizzare valori analogici.

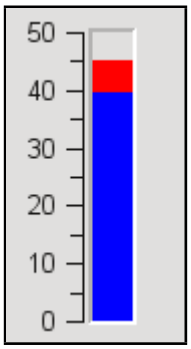

Figura 6.26: QwtThermo

Fornisce gli eventi nei seguenti 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)*

I widget qwt restituiscono valori float. Al fine di utilizzare i valori float è necessario estendere il ciclo degli eventi della maschera.

```
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

Un QwtKnob è usato per valori di input e output su una manopola rotante.

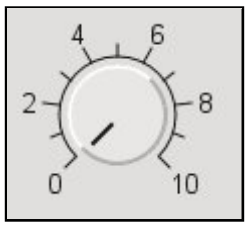

Figura 6.27: QwtKnob

Fornisce gli eventi nei seguenti 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

Un QwtCounter è usato per l'input e l'output limitato da un valore minimo e massimo.

Figura 6.28: QwtCounter

Fornisce gli eventi nei seguenti 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

Un QwtWheel è utilizzato per introdurre valori.

Figura 6.29: QwtWheel

Fornisce gli eventi nei seguenti 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

Un QwtSlider è utilizzato per l'input e l'output di valori limitati da un valore minimo e massimo.

Figura 6.30: QwtSlider

Fornisce gli eventi nei seguenti 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

Un QwtDial è utilizzato per visualizzare valori analogici.

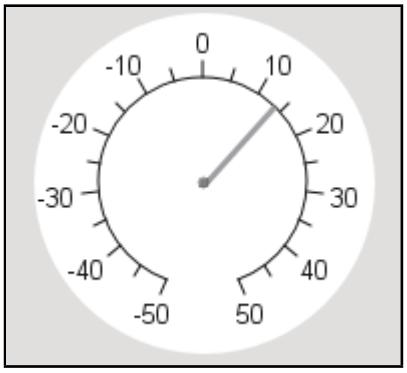

Figura 6.31: QwtDial

Fornisce gli eventi nei seguenti 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

Un QwtAnalogClock è utilizzato per visualizzare l'ora in modo analogico.

Figura 6.32: QwtAnalogClock

Fornisce gli eventi nei seguenti 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

Un QwtCompass è utilizzato per l'input e l'output di valori con una bussola.

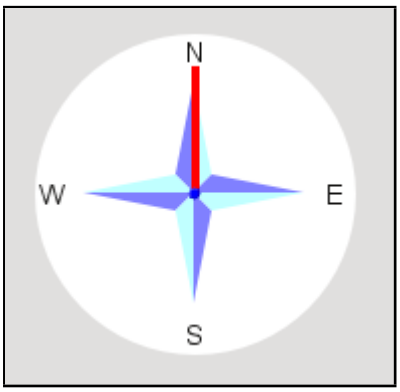

Figura 6.33: QwtCompass

Fornisce gli eventi nei seguenti 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)*

6.8 Grafica

Pvbrowser fornisce diversi metodi per generare la grafica.
In primo luogo è possibile generare output di semplice grafica bitmap nei formati comuni supportati da Qt. E' possibile generare grafica XY con la quale , per esempio , possono essere visualizzati grafici di curve. La grafica di tipo SVG può essere animata dinamicamente ed è possibile ricevere gli eventi quando l'utente effettua dei clic sugli oggetti grafici. Uno strumento esterno di disegno come gnuplot può essere facilmente integrato in quando è in grado di produrre come output grafica bitmap. E 'anche possibile utilizzare la grafica 3D all'interno pvbrowser. Questo può essere fatto con OpenGL. Così , per esempio , è possibile includere file DWF di Autocad nel pvbrowser e animarli. E' anche possibile utilizzare il Visualization Toolkit VTK `http://vtk.org/` in pvbrowser. Gli script Tcl vengono inviati al client pvbrowser che controlla la grafica VTK. Si prega di notare che normalmente VTK non è compilato in pvbrowser. Se si desidera utilizzare VTK è necessario installarlo , modificare il file pvb/pvbrowser/pvbrowser.pro e rimuovere il commento prima di CONFIG += USE_VTK Infine si deve ricompilare pvbrowser.

6.8.1 Grafica Bitmap

La grafica bitmap può essere utilizzato con i seguenti formati JPG, PNG, GIF, TIFF e BMP.

Figura 6.34: Bitmap graphic

Inserire un'immagine in un widget immagine

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

Fornisce gli eventi nei seguenti 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 Grafica xy

La grafica xy può essere creata utilizzando alternativamente i widget Draw o QwtPlot.
Forniscono gli eventi nei seguenti 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)*

Esempi di grafica xy utilizzando i widget Draw e QwtPlot

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

All'interno della funzione 'drawGraphic1' viene disegnato un diagramma con il widget Draw. Le funzioni con 'qpw' all'inizio sono richiamate per il widget QwtPlot.
Il *widget Draw* fornisce alcune 'g-functions' (che iniziano con la lettera 'g') con le quali può essere disegnata ogni forma. In particolare ci sono alcune funzioni per il disegno di assi e curve per i diagrammi.
Il *widget QwtPlot* può disegnare diagrammi più elaborati. E' anche possibile creare i diagrammi in scala logaritmica.
Si prega di rivedere gli argomenti sotto *Graphics* e *QwtPlotWidget* nella guida di riferimento della libreria pvslib.

6.8.3 Strumenti esterni di plottaggio

Strumenti esterni di plottaggio possono essere richiamati dal vostro pvserver. Se lo strumento di plotting è in grado di creare come output immagini bitmap queste possono essere visualizzate in modo semplice nel pvbrowser. Di seguito faremo vedere un'esempio con Gnuplot.

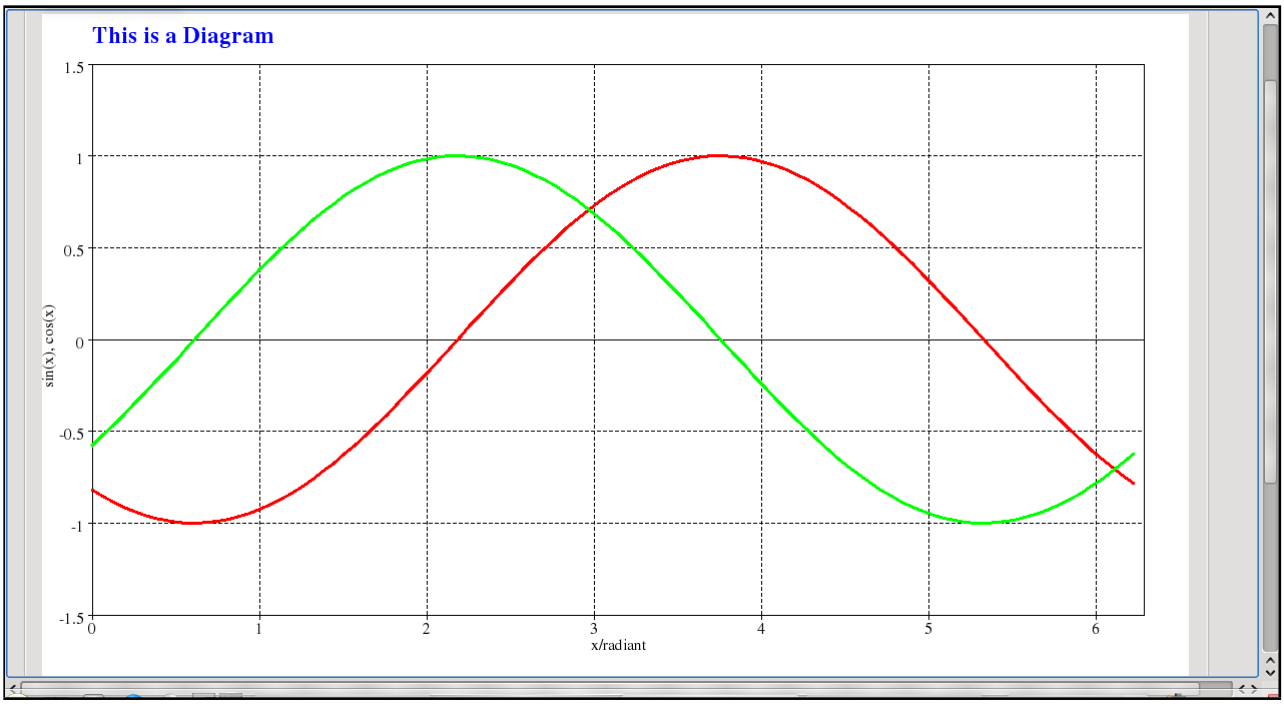

Figura 6.35: Grafica xy usando i widget Draw

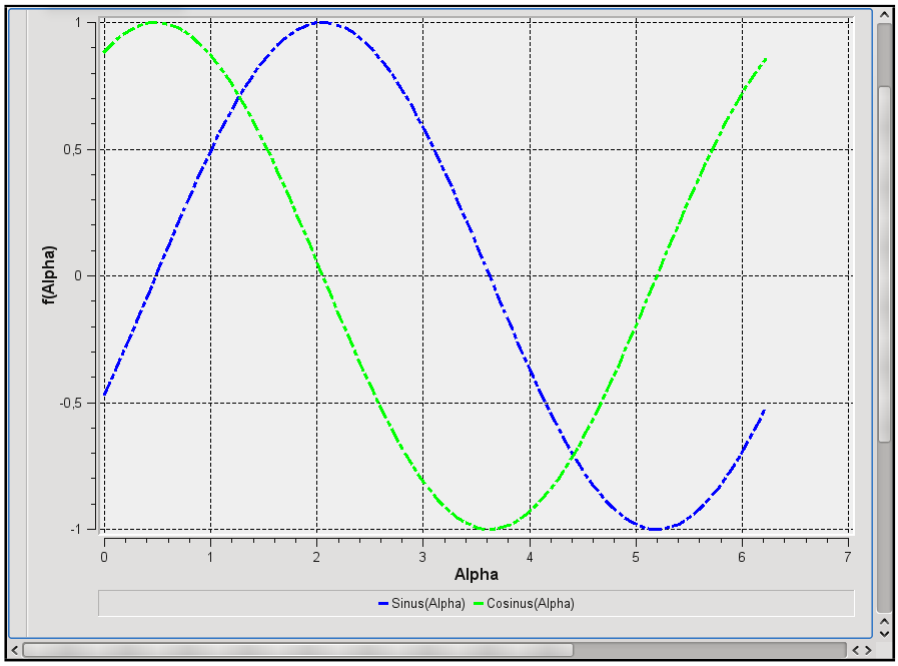

Figura 6.36: Grafica xy utilizzando i widget QwtPlot

Esempio con Gnuplot come strumento di plottaggio estreno

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

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

  // write gnuplot file
  sprintf(buf,"%sgnuplot.dem",p->file_prefix); // p->file_prefix makes filenames unique for multiple
      clients
  fout = fopen(buf,"w");
  if(fout == NULL) return 1;
  fprintf(fout,"# gnuplot test.dem\n");
  fprintf(fout,"set output \"%sgnuplot.png\"\n",p->file_prefix);
  fprintf(fout,"set terminal png\n");
  fprintf(fout,"set xlabel \"x\"\n");
  fprintf(fout,"set ylabel \"y\"\n");
  fprintf(fout,"set key top\n");
  fprintf(fout,"set border 4095\n");
  fprintf(fout,"set xrange [-15:15]\n");
  fprintf(fout,"set yrange [-15:15]\n");
  fprintf(fout,"set zrange [-0.25:1]\n");
  fprintf(fout,"set samples 25\n");
  fprintf(fout,"set isosamples 20\n");
  fprintf(fout,"set title \"Radial sinc function.\"\n");
  fprintf(fout,"splot sin(sqrt(x**2+y**2))/sqrt(x**2+y**2)\n");
  fclose(fout);

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

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

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

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

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

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

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

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

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

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

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

```
}

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

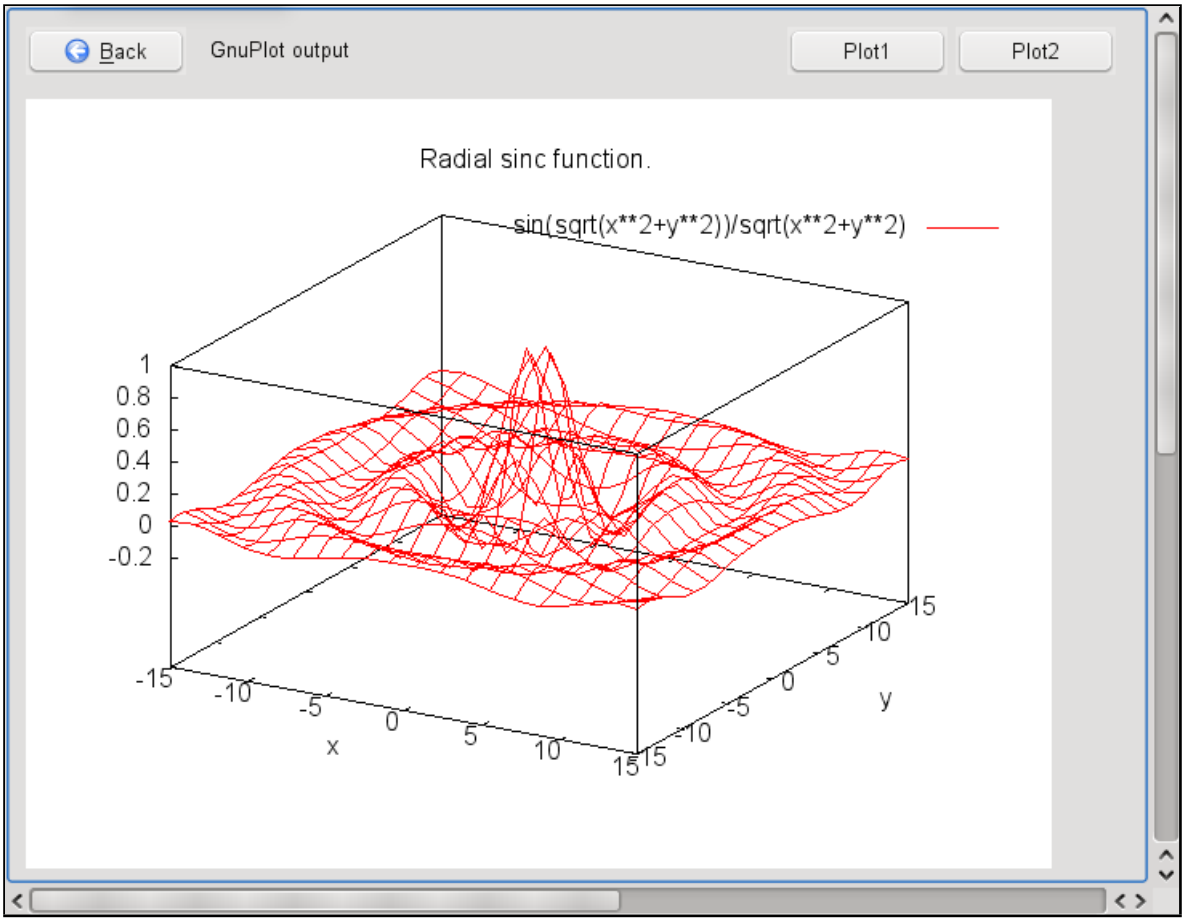

Figura 6.37: Output di Gnuplot in pvbrowser

6.8.4 Grafica SVG

La Scalable Vector Graphic (SVG) è una specifica XML ed è il formato per file di grafica vettoriale bidimensionale che può essere statica o animata. Si tratta di uno standard aperto, che è gestito dal Gruppo SVGWorking W3C.
La grafica SVG può essere creata con speciali programmi di disegno come Inkscape. In alternativa si possono utilizzare strumenti per la conversione di formati CAD come DWF in SVG.

Codice di esempio SVG

```
<?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>
```

Questi elementi grafici possono essere utilizzati in pvbrowser. Basta inserire un widget Draw/SVG e un rlSvgAnimator da rllib per disegnare e animare la SVG. Utilizzando questa classe la grafica SVG viene inviata dal server al client pvbrowser. Una volta che la grafica è nel client pvbrowser rlSvgAnimator è utilizzato per modificare/animare la grafica. Il server riceverà gli eventi quando si 'clicca' (sceglie) gli oggetti all'interno della grafica SVG. Inoltre alcuni eventi del mouse saranno inviati.

Definizione di un rlSvgAnimator

Parte pubblica di 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:
```

Caricare e disegnare grafica SVG

Il codice seguente mostra come caricare un grafico SVG nel client pvbrowser e visualizzarlo.
Per prima cosa un'istanza della classe rlSvgAnimator viene definita in DATA. In 'slotInit' il socket utilizzato per la comunicazione con il cliente pvbrowser e l'id dell'oggetto è registrato nel svgAnimator. Usando il metodo

'read' il file in formato SVG è inviato al client pvbrowser. Con le chiamate a 'pvSetZoomX' e 'pvSetZoomY' pvbrowser è incaricato di rispettare il rapporto di aspetto della grafica quando questa viene ingrandita o ridotta. La funzione di supporto 'drawSVG1' ridisegna la grafica SVG dopo che alcuni passi di animazione sono stati eseguiti.

Caricamento e visualizzazione di un file SVG

```
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);

  pvHtmlOrSvgDump(p,upperWidget,"dump.html");
```

```
  pvClientCommand(p,"html","dump.html");
  return 0;
}
```

Animazione di grafica SVG

Dopo che la grafica SVG è stata caricata e si è in grado di ridisegnarla si può animare/modificare la grafica utilizzando rlSvgAnimator.
Il nome dell'oggetto è utilizzato per indicare gli oggetti grafici all'interno dell'SVG (id = 'objectname'). Usando lo strumento di disegno in formato SVG è possibile impostare l'id come voluto.

Modifica del testo di un oggetto SVG

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

Nascondere o visualizzare un oggetto

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

Impostazione delle proprietà oggetto

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

Ogni oggetto può essere scalato, ruotato e spostato usando la trasformazione della matrice SVG. Usando rlSvgAnimator è possibile modificare le matrici di ciascun oggetto.

Cambiare la matrice di trasformazione di un oggetto

```
// sx    := fattore di scala
// alpha := agolo di rotazione in gradi radianti
// x0,y0 := posizione
// cx,cy := posizione attorno alla quale ruotare
d->svgAnimator.setMatrix(const char *objectname, float sx, float alpha, float x0, float y0, float cx,
     float cy);
// oppure usando
d->svgAnimator.setMatrix(const char *objectname, rlSvgPosition &pos);
```

Attenzione:
Durante la prima chiamata di setMatrix() l'oggetto 'ObjectName' verrà raggruppato. Durante questa operazione tutti i figli di 'ObjectName' che sono stati nascosti verranno nuovamente mostrati.

Definizione di 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);
};
```

Dopo aver definito un 'istanza di rlSvgPosition la matrice è impostata su identity matrix (no movement, no rotation, zoom factor 1.0). Modificando rlSvgPosition è possibile ingrandire, ruotare e spostare l'oggetto. Dopo aver modificato rlSvgPosition è necessario chiamare 'setMatrix' da rlSvgAnimator per applicare la trasformazione per l'oggetto.

Il metodo 'SvgSearchAndReplace' di rlSvgAnimator può sostituire parte delle proprietà. Questo è utile per esempio se si desidera modificare le proprietà di stile. Il parametro 'before' può includere caratteri jolly come ∗(qualsiasi carattere) ?(1 carattere qualsiasi).

Modificare le proprietà

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

Inoltre vi sono metodi per applicare le modifiche ricorsivamente a tutte le parti al di sotto di un oggetto grafico.

Settare o modificare le proprietà di un oggetto in modo ricorsivo

```
  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);
```

Dopo aver modificato tutti gli oggetti come desiderato l'oggetto SVG dovrà essere ridisegnato utilizzando 'drawSVG1'.
Il metodo 'read' che invia la grafica SVG al client pvbrowser può avere un file ini di parametri opzionali. Il file ini dovrà essere riempito con le proprietà dell'oggetto grafico SVG. L'ID dell'oggetto grafico dovrà corrisponde alle sezioni all'interno del file ini. Suggerimento: È possibile utilizzare le proprie proprietà come tuo:tag = 'qualsiasi valore' all'interno del SVG. Queste proprietà saranno ignorate dal renderizzatore SVG. È possibile utilizzare questa proprietà per i propri scopi.

Parametro opzionale nel file INI quando si chiama read.

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

Gestione degli eventi da oggetti grafici SVG

La grafica SVG viene implementata con il widget Draw. Il widget Draw renderà disponibili i seguenti eventi quando contiene un grafico SVG.

- *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)*

Soprattutto il TextEvent è interessante.
Se si desidera che venga generato un'evento quando l'utente fa clic su un oggetto grafico all'interno del grafico SVG l'id dell'oggetto deve iniziare con 'pv'. o 'PV'. .
Esempio: id='pv.testclick'
Allora un 'slotTextEvent' sarà attivato. Se il nome comincia con 'PV.' id='PV.textclick' pvbrowser potrà inoltre dare un feedback visuale all'utente cambiando la forma del mouse con Qt::PointingHandCursor quando l'utente muoverà il mouse sopra l'oggetto. Questo potrebbe essere utilizzato per implementare 'Buttons' con feedback visivo.
Lo slotTextEvent non è usato solo per l'evento click ma ci sono più funzioni che possono innescare slotTextEvent. Per questo mettiamo a disposizione alcune funzioni di supporto che possono interpretare il testo.

slotTextEvent quando si usa grafica SVG

```
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;
}
```

La funzione di supporto 'svgObjectName' restituirà il nome dell'oggetto SVG. La funzione di supporto 'textEventType' rileverà un TextEvent normale o eventi speciali per l'utilizzo con SVG. Si saprà se l'oggetto è stato cliccato e quale pulsante del mouse è stato usato.
Il 'case SVG_BOUNDS_ON_ELEMENT:' è una risposta alla seguente richiesta.

Richiesta di rettangolo che circonda un oggetto in formato SVG

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

Il 'case SVG_MATRIX_FOR_ELEMENT:' è una risposta alla seguente richiesta.

Richiesta di matrice per un oggetto in formato SVG

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

Zoom e panoramica per tutta la grafica SVG

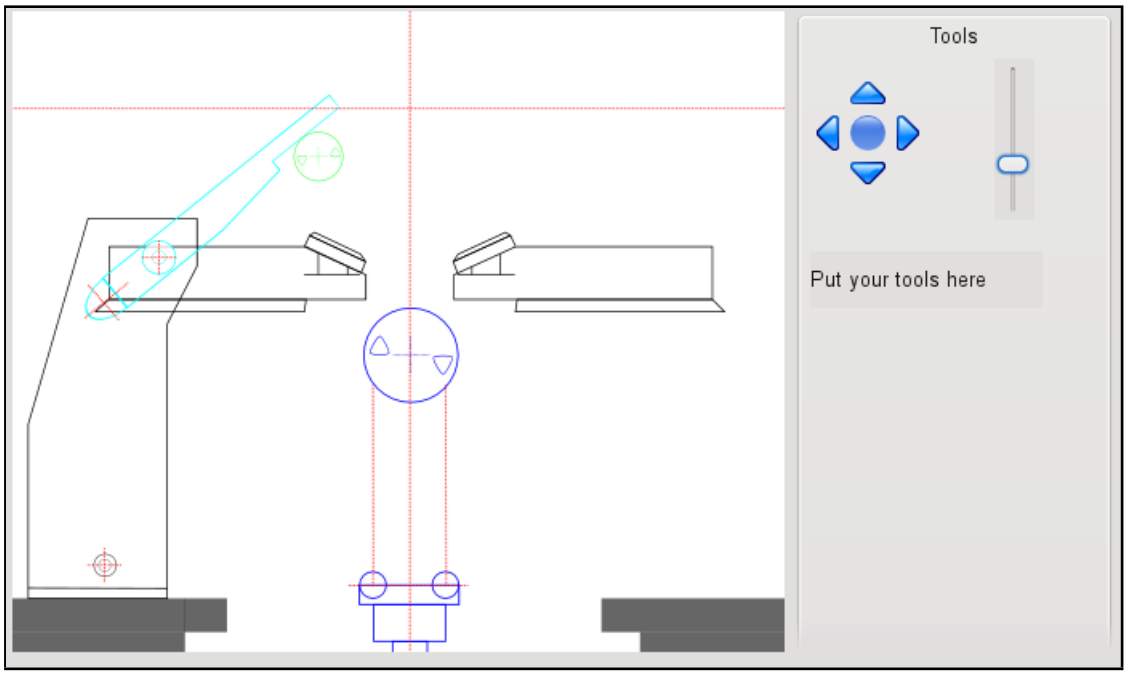

Figura 6.38: Zoom e panoramica per tutta la grafica SVG

Nel direttorio pvbaddon/template/weblayout/ dentro pvbaddon è possibile trovare un modello dove lo zoom e panning per la grafica SVG è completamente implementato.
Innanzitutto guardare come il nome (ID) della intera grafica SVG viene chiamato all'interno del codice XML SVG. Si consiglia di chiamarlo 'main', perché questo è il nome predefinito che rlSvgAnimator assume.
Le seguenti slot functions sono responsabili dello zoom e panning.

Zoom dell'intero grafico SVG in percentuale

```
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;
}
```

Regolazione delle dimensioni della finestra attuale

```
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;
}
```

Pan del grafico con il 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;
}
```

Pan del grafico SVG con i pulsanti

```
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);
  }
  return 0;
}
```

Esempi di grafica SVG

I primi 2 esempi sono in pvsexample che viene fornito con pvbrowser.

6.8.5 OpenGL

In pvbrowser sono disponibili widgets OpenGl. Il pvserver invia i comandi OpenGl al client pvbrowser dove questi vengono eseguiti. Un degli utilizzi di OpenGl potrebbe essere quello di incorporare dei disegni CAD in pvbrowser. Per esempio Autocad utilizza il formato DWF come formato di interscambio con le altre applicazioni. Per questo formato esiste un convertitore che trasforma i file DWF in OpenGL e così puo essere utilizzato in pvbroweser. Il disegno generato viene inviato al client pvbrowser con 'pvSendOpenGL' il quale lo visualizzerà. Gli oggetti inclusi nel disegno sono contenuti nelle liste display (listarray). Con queste liste display la grafica può essere animata.
I widgets OpenGL forniscono gli eventi nei seguenti slot.

- *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)*

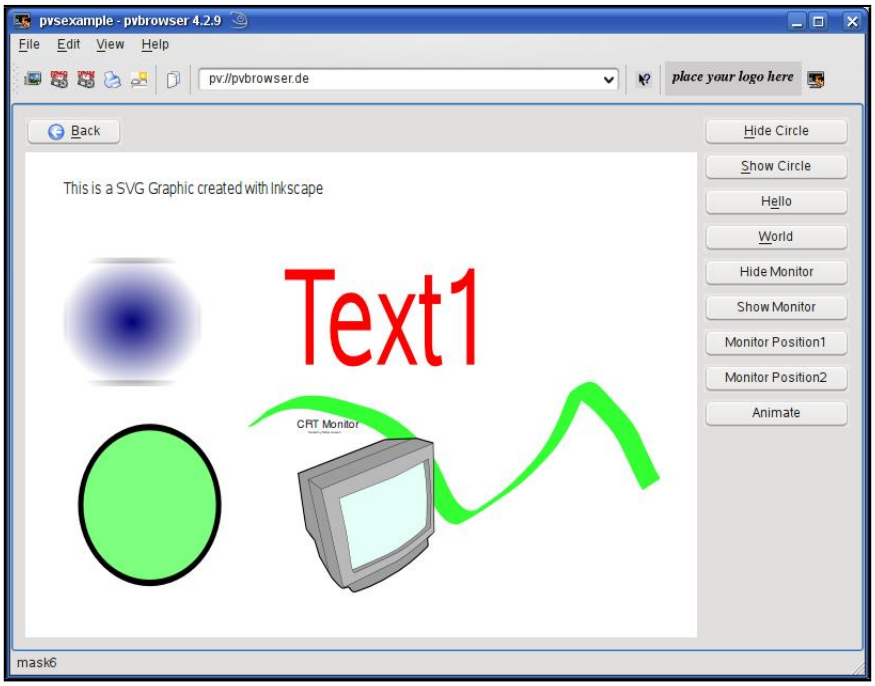

Figura 6.39: Semplice SVG

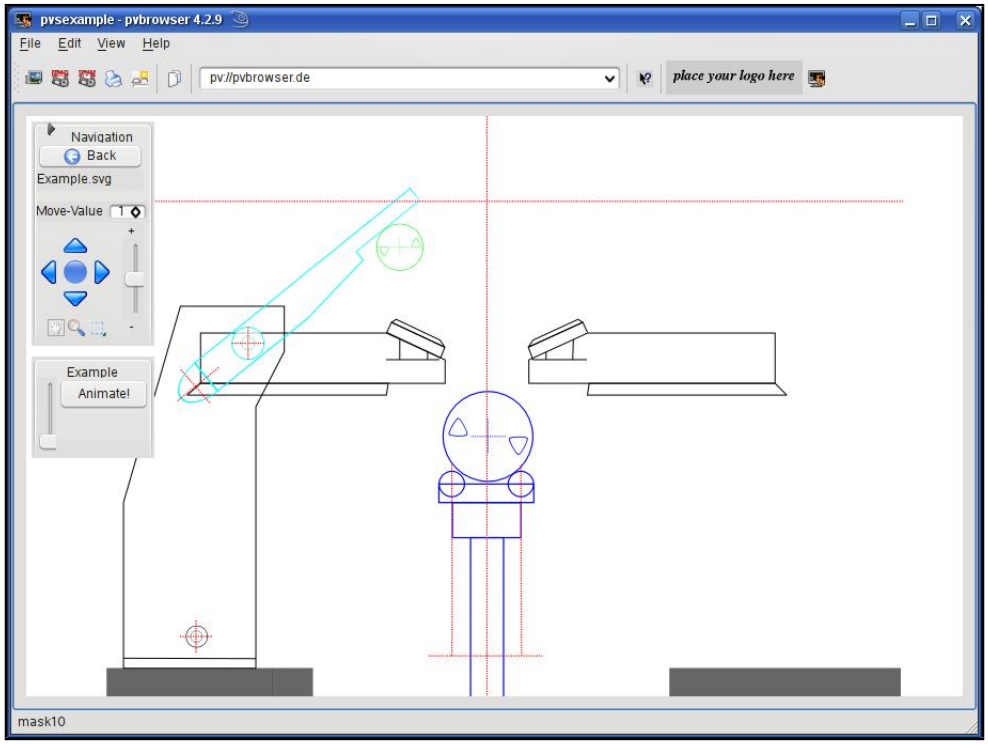

Figura 6.40: Un disegno meccanico

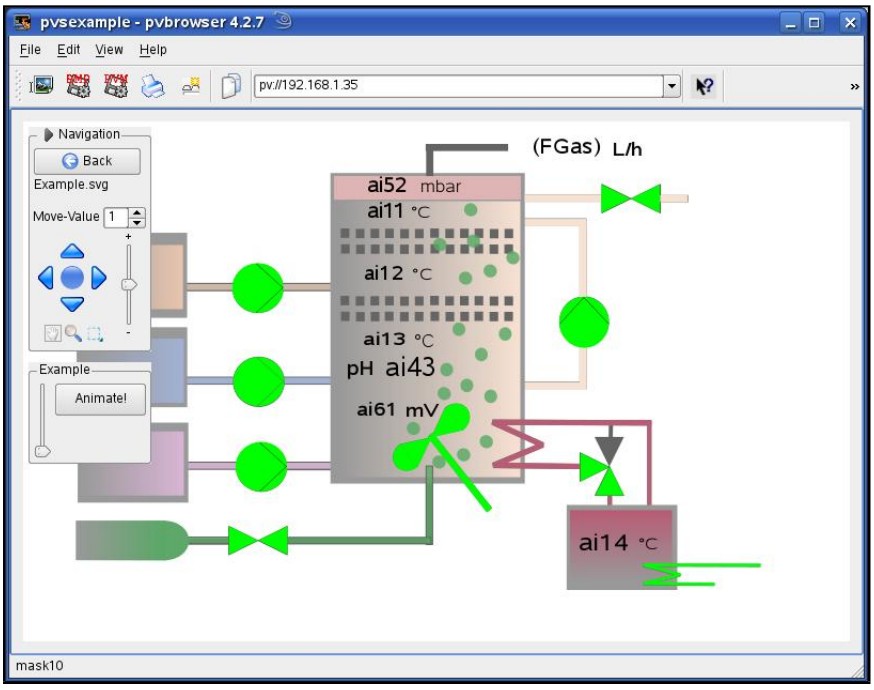

Figura 6.41: Un SVG di in processo

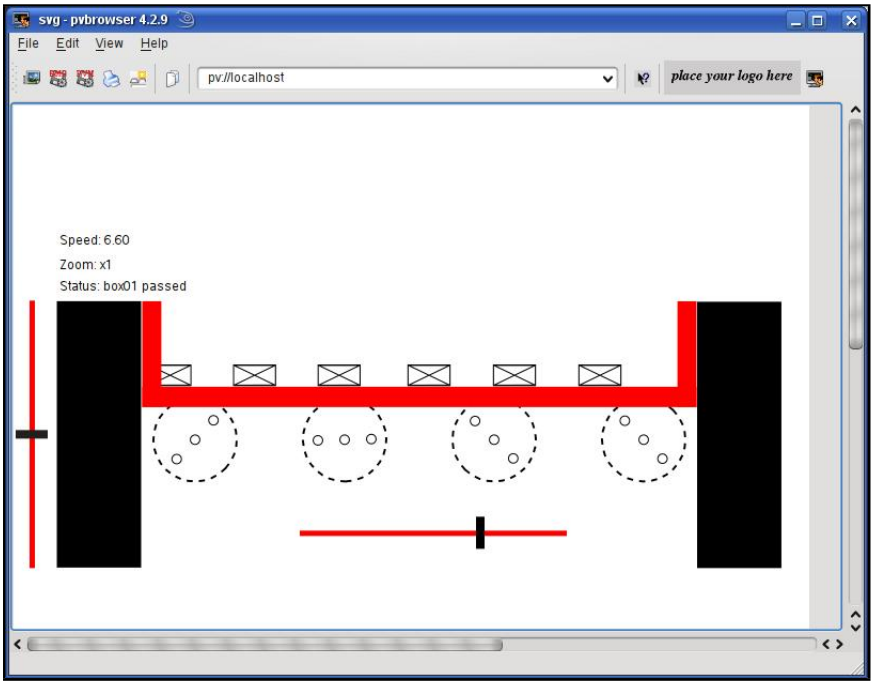

Figura 6.42: Un SVG che mostra i pacchetti in movimento su di un nastro trasportatore

Un file DWF Autocad viene esportato in formato OpenGL e viene visualizzato e animato in pvbrowser

```
//#################################################################
//# 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;
  // 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
  pvSetValue(p,sliderPos,50);

  // load OpenGL graphic
  pvGlBegin(p,OpenGL1);
  d->gl.proportional.read("gl/proportional.glfont"); // load proportional font
  d->gl.fixed.read("gl/fixed.glfont");          // load fixed font
  d->gl.proportional.setZoom(0.9f);
```

```
  d->gl.fixed.setZoom(0.9f);
  d->gl.num_listarray = pvSendOpenGL(p,"gl/scene.gl.cpp",d->gl.listarray,100,&d->gl.proportional,&d->
      gl.fixed);
  pvGlEnd(p);

  paintGL(p,d);

  // Download Graphics
  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");

  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);
  }
  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 è un potente visualizzatore 3D che è stato scritto in C++ ed è basato sulle OpenGL. In VTK si possono utilizzare script (sequenze di istruzioni per operazioni automatizzate) utilizzando il Tcl.
Nal client pvbrowser c'è un widget VTK se il pvbrowser è stato compilato con il supporto a VTK. Un pvserver può ora inviare scripts Tcl al client pvbrowser come se lo inviasse ad un widget che può interpretali. In questo modo VTK può essere incorporato in una visualizzazione senza generare un carico eccessivo sulla CPU del pvserver. Il rendering della scena 3D viene comletamente eseguito dal client. Il pvserver invierà solamente alcuni comandi Tcl al client.

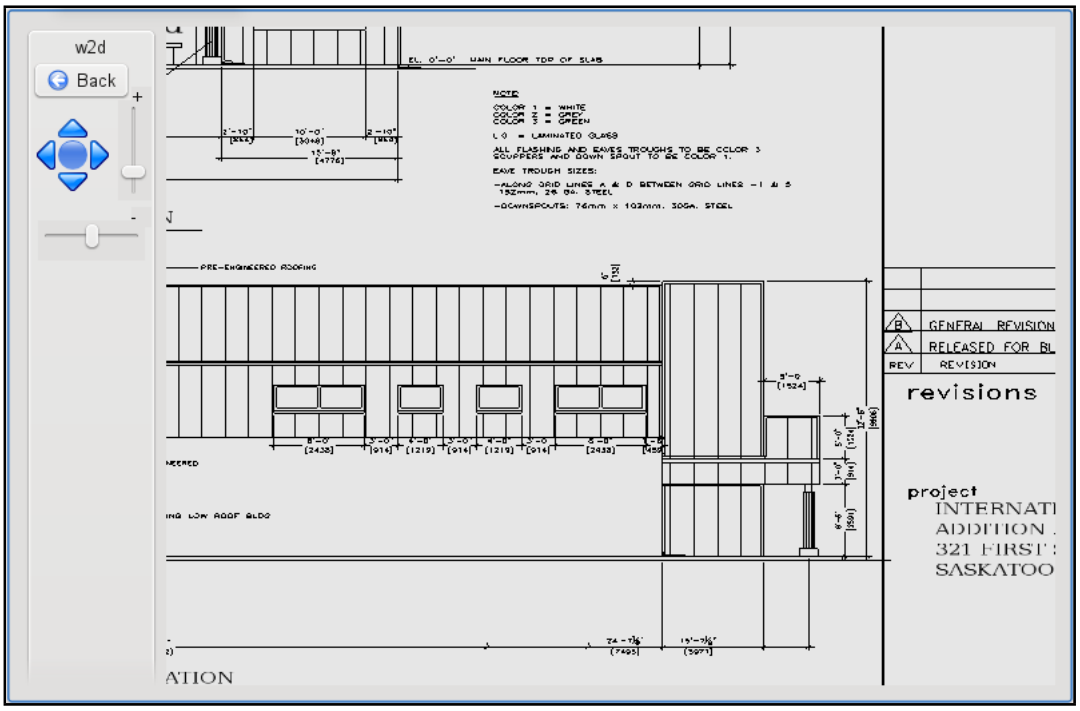

Figura 6.43: Disegno Autocad in pvbrowser

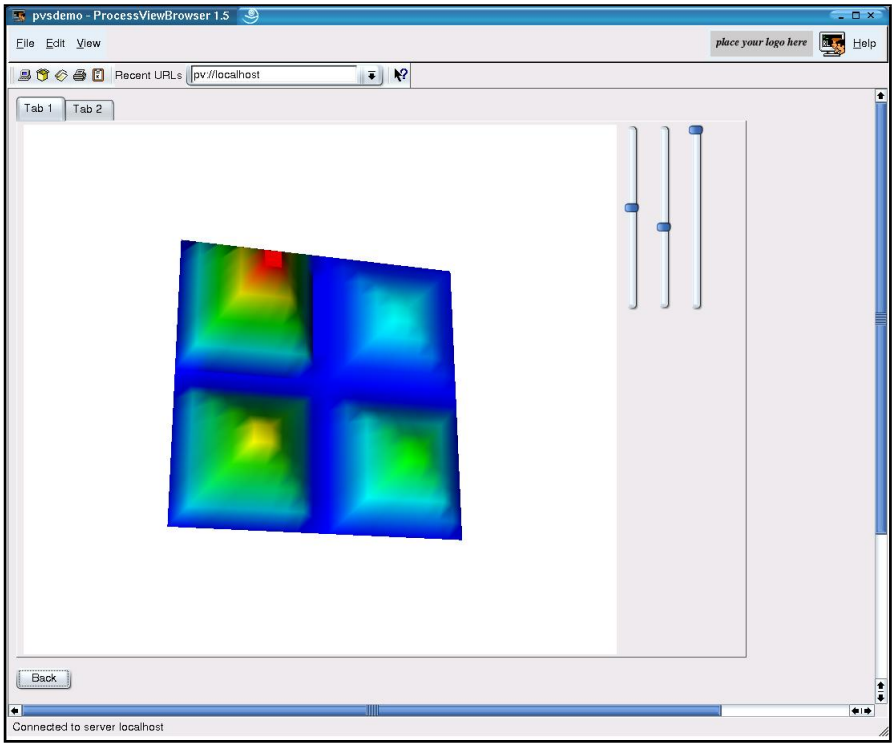

Figura 6.44: Display of a 2D datasset data1.vtk

Esempio per visualizzazione di un dataset 2D 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]
    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 potrebbe per esempio essere un profilo di misura. Il pvserver potrebbe generare il data1.vtk e aggiornare la schermata con le nuove misure che ha rilevato.

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
```

La funzione slot per 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;
}
```

6.9 Dialoghi

In pvbrowser sono disponibili alcuni semplici dialoghi (dialog box). Ma possono anche essere creati dialoghi molto complessi.

6.9.1 MessageBox

Figura 6.45: A MessageBox

I message boxes possono essere programmati con la funzione 'pvMessageBox'. id_return determina sotto quale id il risultato del MessageBox deve essere inviato. E' possibile assegnare valori negativi per non generare conflitti con i widgets che sono già stati progettati. Il valore del pulsante che è stato cliccato viene consegnato in 'slotSliderEvent'. Se si vogliono utilizzare meno di tre pulsanti i rimanenti valori sono settati a 0 oppure 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

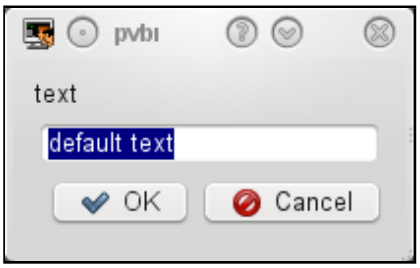

Figura 6.46: Un Input Dialog

Gli Input Dialogs possono essere programmati con la funzione 'pvInputDialog'. id_return determina sotto quale id il risultato dell'InputDialog deve essere inviato. E' possibile assegnare valori negativi per non generare conflitti con i widgets che sono già stati progettati. Il testo che è stato inserito dall'utente viene consegnato in 'slotTextEvent'.

pvInputDialog

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

6.9.3 FileDialog

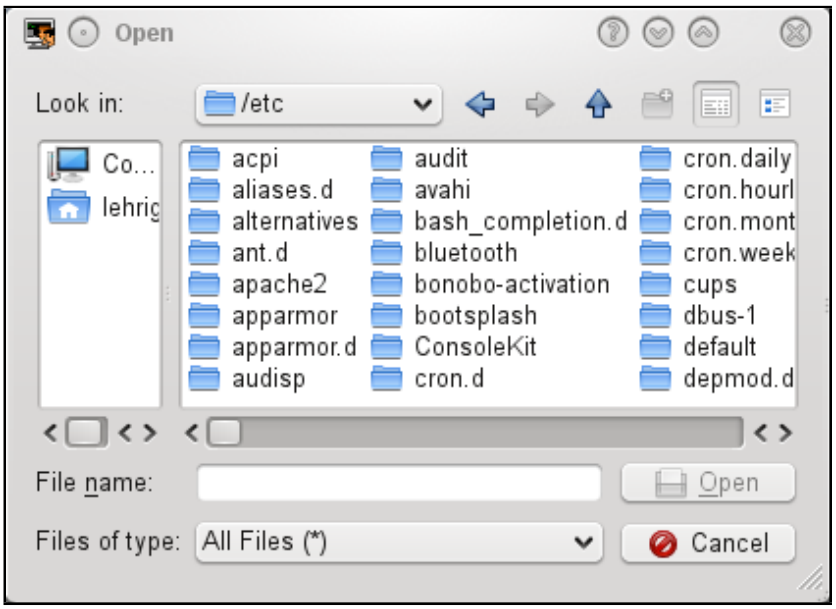

Figura 6.47: Un File Dialog

I File Dialogs possono essere programmati con la funzione 'pvFileDialog'. id_return determina sotto quale id il risultato del FileDialog deve essere inviato. E' possibile assegnare valori negativi per non generare conflitti con i widgets che sono già stati progettati. Il risultato verrà consegnato in 'slotTextEvent'.

File Dialog

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

6.9.4 Dialoghi Modali

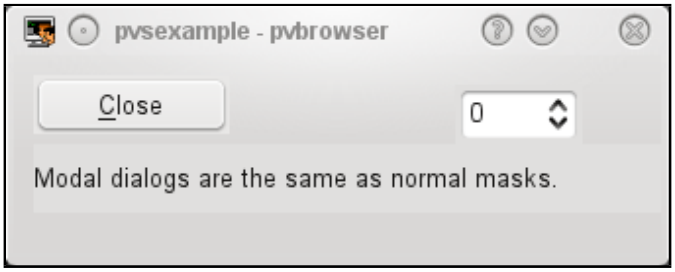

Figura 6.48: Un dialogo modale

I dialoghi modali possono essere progettati come normali maschere. Nella maschera chiamante bisognerà utilizzare la funzione 'pvRunModalDialog'. La dimensione della finestra di dialogo è impostata con larghezza (width) e altezza(height). La funzione 'showMask' è la funzione per visualizzare il dialogo modale. 'userData' è l'indirizzo della struttura dati che può essere utilizzata per ritornare valori dal dialogo modale. Se si desidera aggiornare la maschera base dal dialogo modale deve essere utilizzato lo 'slotNullEvent' per il parametro 'showData'. Oppure può essere impostato 'readData' e 'showData' a NULL. Il parametro 'd' alla fine e necessario per 'showData'.

Dialogo modale

```
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);
```

Chiamare un dialogo da una maschera

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

Nello 'slotNullEvent' il dialogo modale può aggiornare la finestra di base.

aggiornare una finestra base dall'interno di un dialogo modale

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

terminare un dialogo modale e inviare valori alla maschera chiamante

```
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

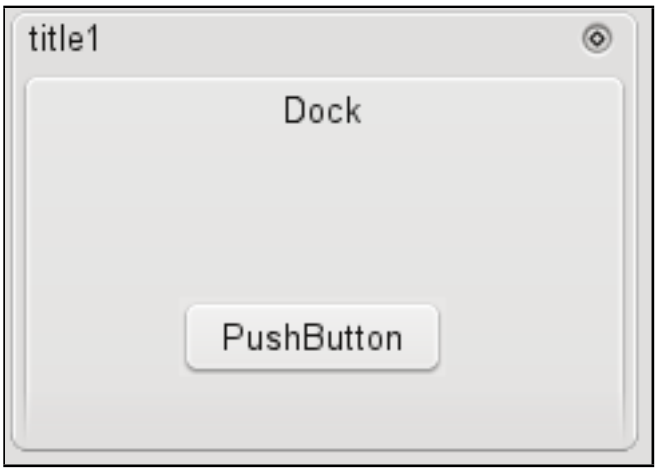

Figura 6.49: Dock Widget

I Dock Widget in linea di principio lavorano come finestre di dialogo non modali ma possono essere ancorate ai bordi della finestra di pvbrowser. Il contenuto di un Dock Widget può essere progettato all'interno della maschera. Si può inserire questo disegno sul bordo più a destra della maschera. Quindi si imposta l'oggetto principale (esempio: GroupBox) come root_id nel DockWidget. A questo punto l'oggetto scompare dalla maschera e lo si ritrova nel Dock Widget.

Aggiungere un 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 Menu a comparsa

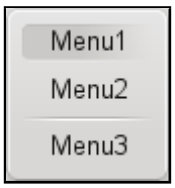

Figura 6.50: Popup Menu

Un menu a comparsa può essere creato con il seguente codice. I menu popup sono programmati con la funzione 'pvPopupMenu'. id_return determina sotto quale id il risultato della PopupMenu deve essere inviato. E' possibile assegnare valori negativi per non generare conflitti con i widgets che sono già stati progettati. L'input di testo da parte dell'utente viene consegnato in 'slotTextEvent'. Due virgole nel testo servono a creare un separatore all'interno del menu.

Creare un Menu a comparsa

```
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 Traduzioni

pvbrowser utilizza la codifica UTF-8. In questo modo si possono anche utilizzare le lingue non latine. E' possibile utilizzare il cirillico o lettere cinesi all'interno della vostra visualizzazione (maschera). Cioè è possibile utilizzare tutte le lettere che possono essere rappresentate con la codifica UTF-8. Il testo del menu di pvbrowser può essere adattato per nuovi linguaggi modificando il file INI '. Pvbrowser.ini'.
Durante l'esecuzione di pvbrowser, è possibile passare da una lingua all'altra, se si ha avuto cura di predisporre il vostro pvserver. In processviewserver.h e' definita la macro ' define pvtr(txt) txt'. La macro semplicemente restituisce l'originale " txt". Se si crea un pvserver si dovrebbe usare 'pvtr (" Qualsiasi testo")' per tutti i testi. Quando si desidera tradurre il pvserver in più lingue si puo' utilizzare un file INI (classe rlIniFile). Si include rlinifile.h in pvapp.h. rlinifile.h ridefinirà la macro 'pvtr (txt)' e cercherà di tradurre "qualsiasi testo" utilizzando il file INI. In main.cpp si inserisce 'rlSetTranslator' all'inizio del main().

Impostazione della lingua predefinita per 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;
}
```

Con questo codice il file INI 'translation.ini' viene caricato e la sezione di default è impostata su" TEDESCO". Ogni chiamata a 'pvtr(" Qualsiasi testo")' tenterà ora di tradurre il testo in tedesco.

Mentre un client è connesso è possibile passare da una lingua all'altra. Ad esempio, l'utente può premere un pulsante e il pvserver potrebbe richiamere 'pvSelectLanguage(p," inglese"),' impostando la lingua corrente per l'utente ad inglese.

Impostazione della lingua per un singolo 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;
}
```

Dopo che si è cambiata la lingua con 'pvSelectLanguage()' l'esempio fa un 'return 1' per ritornare a 'pvMain()' e mostrare di nuovo la maschera ma ora con la nuova lingua. La lingua corrisponde ad una sezione del file INI. È possibile definire un numero illimitato di lingue all'interno di un file INI. Quando si codifica il proprio pvserver si dovrebbe semplicemente farlo nella lingua di default e racchiudere ogni testo con 'pvtr("Qualsiasi testo nella lingua di default")'. Poi si dovrebbe fare la traduzione.
C'è un particolare argomento da passare al comando pvdevelop che vi aiuterà ad estrarre il testo che dovrete tradurre.

Estrarre il testo da tradurre

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

This will search the sources of your pvserver for 'pvtr(' and dump the text to dump.ini. Questo cercherà 'pvtr(' nei sorgenti del vostro pvserver e scarichera' il testo in dump.ini.

Example dump.ini

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

Da questo si modifica il file INI traducendo nella lingua di destinazione

Traduzione in inglese

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

Vi ricordiamo che è anche possibile utilizzare il testo come una stringa in formato 'printf ()'. Se il testo da tradurre contiene un carattere '=' questo carattere deve essere racchiuso con '\', per poterlo distingue dal carattere '=' he si frappone tra il nome e il valore nel file INI.
pvdevelop inserisce la macro 'pvtr(txt)' quando si genera una maschera. Se non vi è alcuna traduzione viene restituito il testo originale. In pvbaddon 'pvbaddon/templates/MyEventHandler' si trova un esempio per la traduzione.
La struttura PARAM contiene una variabile 'int language' che può essere utilizzata per ricordare la lingua scelta. È possibile valutare questa variabile per esempio quando si vuole utilizzare le unità di misura in una lingua specifica.

Impostazione della lingua in pvserver

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

6.11 Conversione delle unità di misura

Nella struttura PARAM c'è il membro 'convert_units'. Si può impostare p->convert_units a 0 o 1. (standard: 0).
Per la conversione delle unità si utilizza la funzione 'float unit(PARAM *p, float val, int conversion);' . Se (p->convert_units == 0) il valore originale viene restituito.

Conversione delle unità

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

Quelle riportate di seguito sono le conversioni disponibili. Si può impostare p->convert_units a seconda della lingua selezionata.

Unità di conversione

```
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 Gestione del Layout

La gestione del layout può essere definita in pvdevelop. Scegliere il menù inerente (tasto destro del mouse) all'interno della progettazione grafica di pvdevelop. Quando si progetta la maschera è possibile impostare i valori minimo e massimo per la dimensione dei widget e organizzare i widget in modo che possano essere modificati solo entro certi limiti. Se i valori minimi e massimi hanno lo stesso valore allora le dimensioni del widget sono fisse. Questo è un'esempio di codice per la gestione del layout.

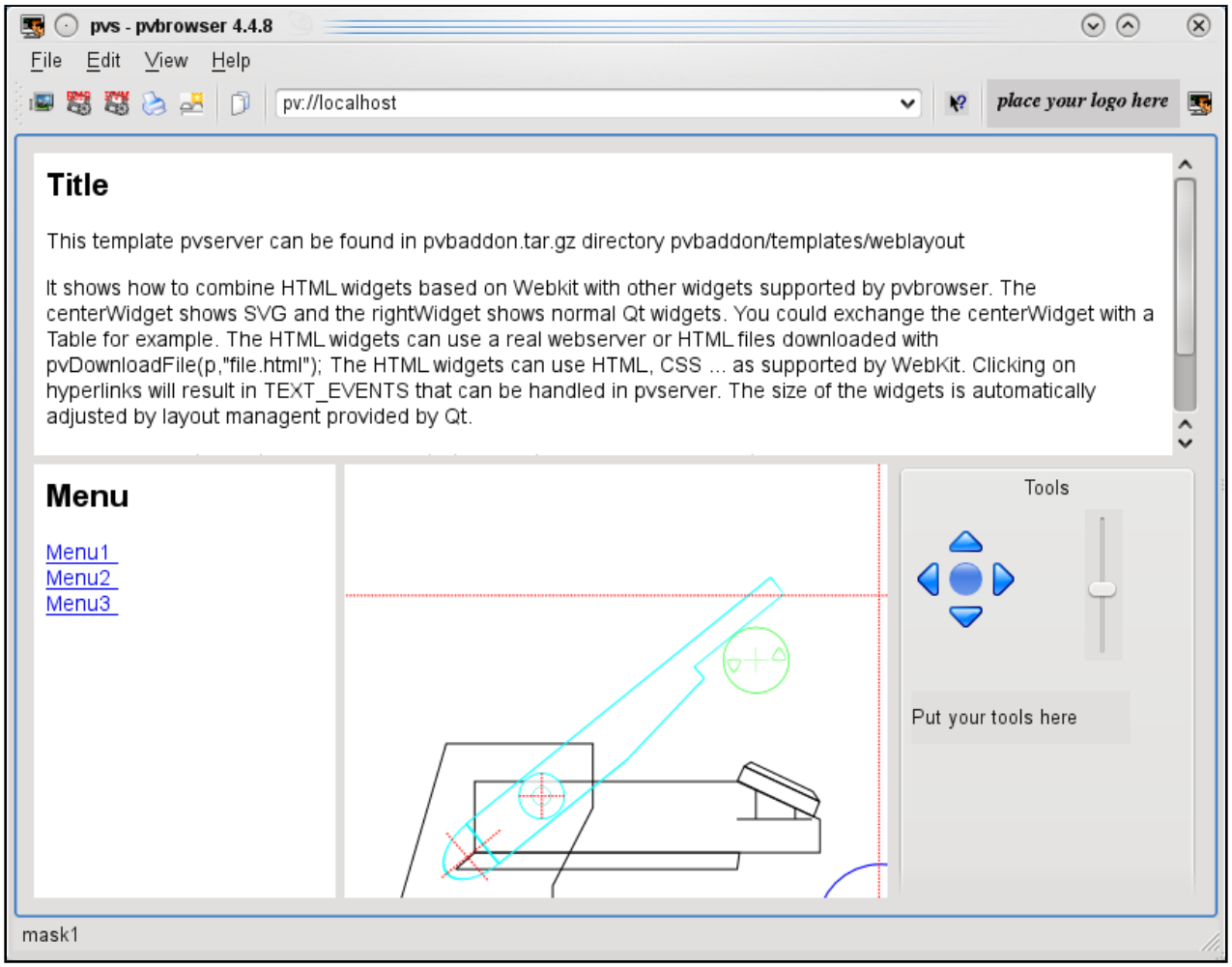

Figura 6.51: Layout per il codice d'esempio

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 Impostare l'ordine di tabulazione

L'ordine di tabulazione può essere impostato in pvdevelop. Sceglendo il menù inerente (tasto destro) nella progettazione grafica di pvdevelop. Quindi fare clic sugli oggetti nella sequenza desiderata. Gli oggetti scelti vengono nascosti per aiutare la scelta.

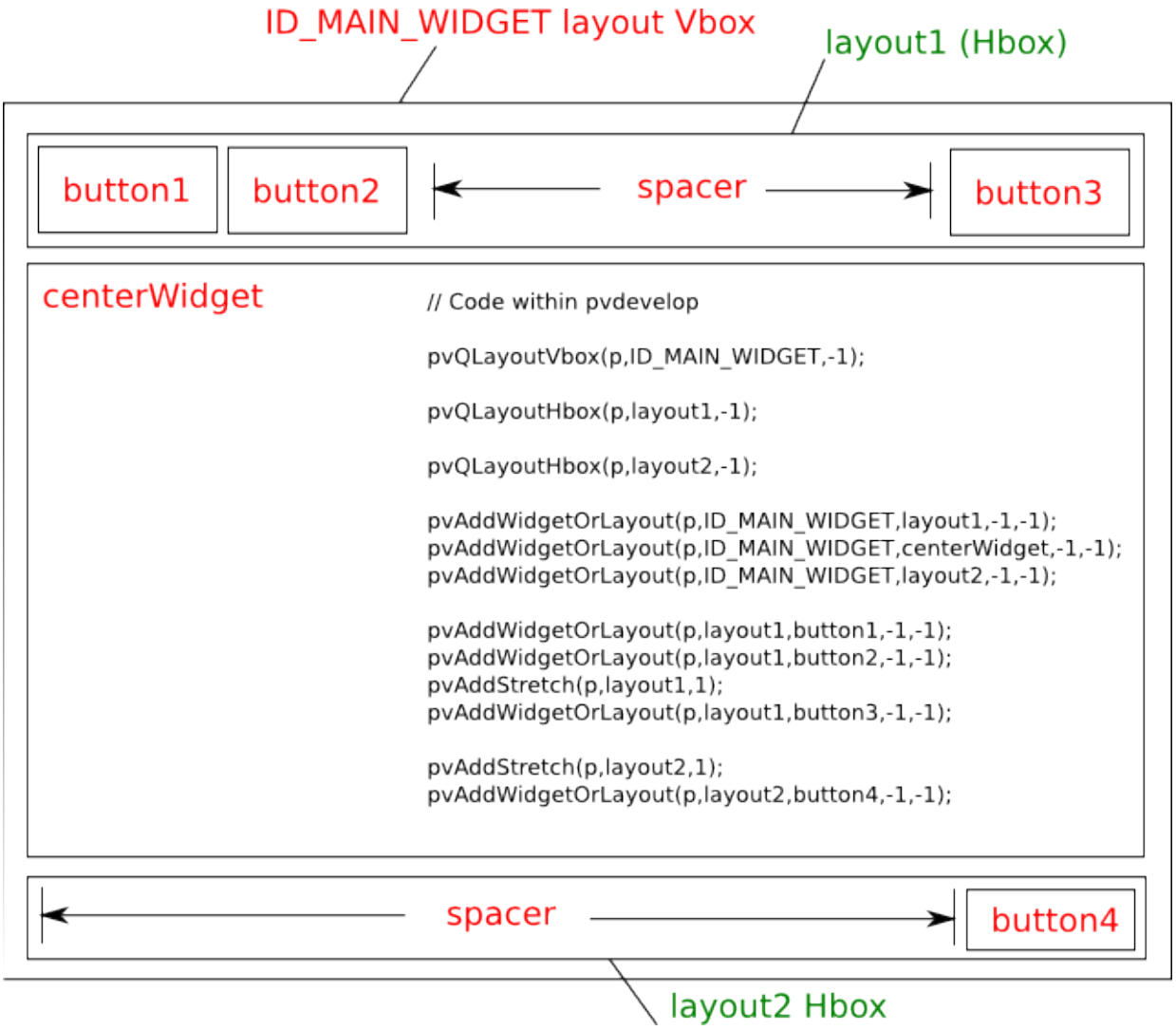

Figura 6.52: Layout Example

6.14 Webcam

Le Webcams con flusso Motion JPEG su http possono essere utilizzate grazie alla classe rlWebcam.
Se non si conosce sotto quale URL la WebCam rende disponibile il flusso video M-JPEG si può capirlo con tcpdump.

Utilizzare 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....
```

Integrazione di una Webcam 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;
}
```

6.15 Cookies

I cookie sono brevi informazioni che vengono memorizzate sul computer client se il client pvbrowser è configurato per accettare i cookie.

Esempio di 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
```

Capitolo 7

Acquisizione dati

L'acquisizione Dati in pvbrowser è realizzata per mezzo di demoni separati dal pvserver(i demoni sono processi in esecuzione in background). Questi demoni 'Parlano' il protocollo a seconda del bus di campo o del PLC con cui devono interagire. Un demone è costituito da due thread. Un thread legge i dati ciclicamente e scrive il risultato in una memoria condivisa mentre l'altro thread è in attesa dell'arrivo dei comandi di output sulla casella di posta per inviare i dati in uscita al bus di campo o al PLC. La visualizzazione può così leggere la memoria condivisa e visualizzarne il suo contenuto. Se si desidera inviare dati in uscita la visualizzazione invia un messaggio alla cassetta postale.

Un vantaggio di questa architettura è che l'acquisizione dei dati e la visualizzazione sono separati gli uni dagli altri. Ad esempio la visualizzazione può essere riavviata senza la necessità di rileggere tutti gli ingressi perché questi sono ancora disponibili nella memoria condivisa e la lettura dei dati continua anche mentre la visualizzazione è ferma al fine di poter apportare modifiche ad essa. Inoltre non è un problema eseguire diversi demoni in parallelo dove ogni demone ha la propria memoria condivisa e una cassetta postale. Così si è indipendenti dal protocollo ed è possibile eseguire diversi demoni in contemporanea che utilizzano un diverso protocollo su ciascuno.

ATTENZIONE: Su sistemi operativi di tipo unix è necessario cancellare i vecchi file di memoria condivisa o le caselle postali se viene cambiata la loro dimensione.

```
# 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 si possono trovare alcuni demoni che sono configurati per mezzo di un file INI. Questi demoni utilizzano la classe rlDataAcquisitionProvider di rllib. Sul lato del pvserver si utilizza la classe rlDataAcquisition per l'accesso alla memoria condivisa e alla cassetta postale. Le variabili sono codificate come testo ASCII leggibile con queste classi.

Inoltre è possibile generare demoni utilizzando pvdevelop. Con questi demoni generati le variabili sono codificate binarie.

La scelta tra i due deve essere effettuata analizzando se il carico della codifica ASCII è accettabile o se è meglio utilizzare la codifica binaria. La rappresentazione ASCII rende più semplice la loro gestione per lo sviluppatore mentre invece è più veloce accedere alle variabili codificate in binario.

Se si desidera implementare nuovi protocolli è possibile utilizzare un demone esistente come modello e modificarlo per supportare il nuovo protocollo. Il miglior modello potrebbe essere il demone Modbus.

È possibile utilizzare protocolli che attualmente non sono supportati da pvbrowser. L'unica cosa di cui avete bisogno è una libreria esterna che implementi questo protocollo. Queste librerie possono essere scritte in C o C++ oppure librerie che avete creato utilizzando altre classi di rllib. Ci farebbe piacere se ci inviaste le vostre implemetazioni in modo da poterle includere all'interno del nostro progetto.

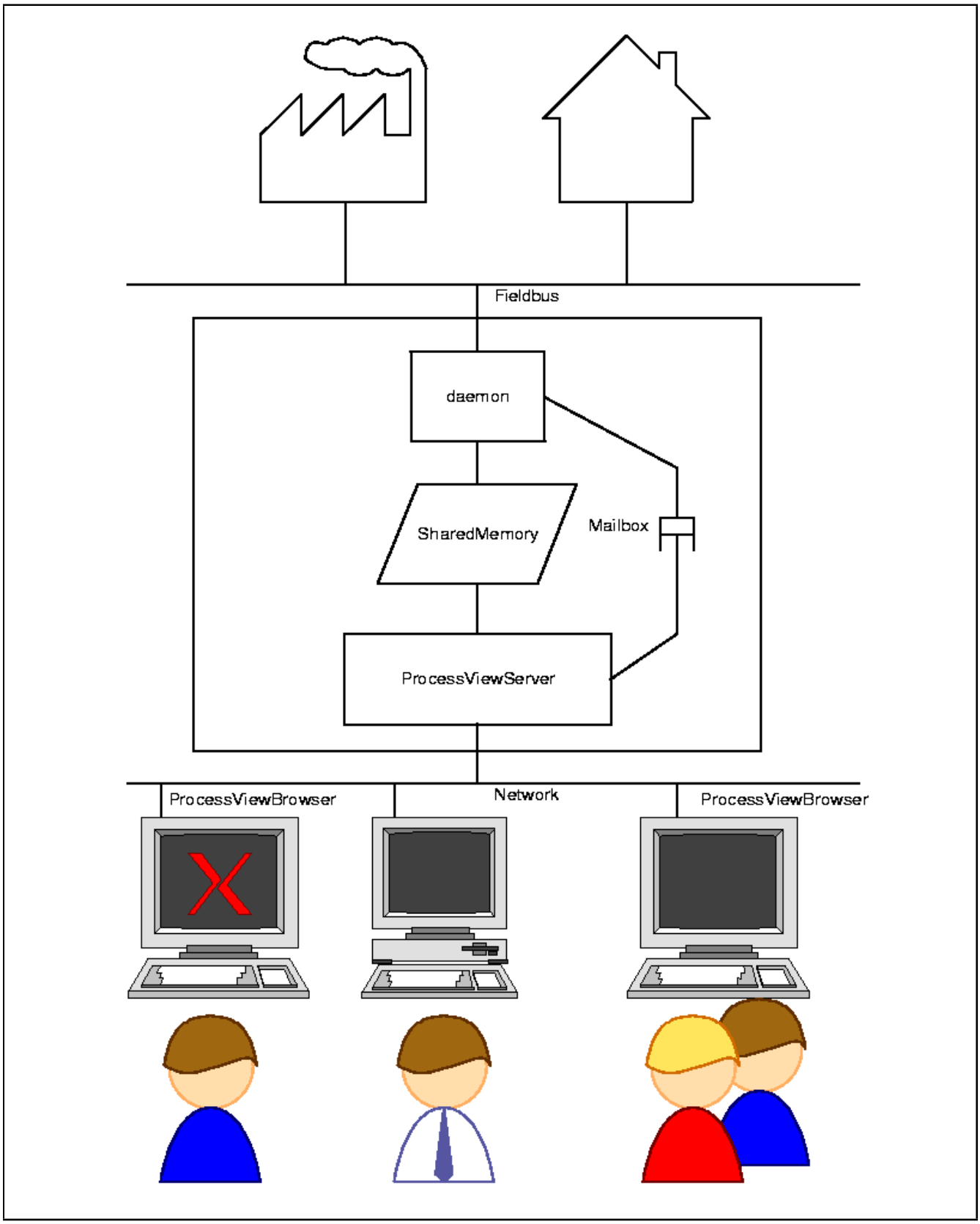

Figura 7.1: Principio dell'acquisizione dati in pvbrowser

7.0.1 Copiare il demone nella directory standard

I demoni di pvbaddon devono essere copiati in una directory che sia inclusa nella variabile d'ambiente $PATH del vostro sistema operativo perché così lo si potrà avviare da qualsiasi posizione, semplicemente inserendo il nome sulla linea di comando di una shell.

Linux

```
cp tuo_demone /usr/bin/
```

Windows

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

7.0.2 Il file INI per il vostro demone

Il file INI per il demone può essere posizionato in qualsiasi directory. Avviare il demone con il seguente comando.

Avviare il demone da un terminale/dos

```
tuo_demone /path/del/tuo_config.ini
```

Per i tese utilizzate un terminale dos. Se volete avviare tutto al momento del boot ed in background il modo di agire dipenderà se come sistema operativo state utilizzando Linux o Windows. Leggete la sezione 'Avviare un pvserver in background'.

7.0.3 Configurare la memoria condivisa e la mailbox

Gli esempi in pvbaddon suggeriscono una posizione standard.

Linux

```
/srv/automation/shm (memoria condivisa)
/srv/automation/mbx (mailboxes)
```

Windows

```
c:\automation\shm (memoria condivisa)
c:\automation\mbx (mailboxes)
```

Ricordatevi che dovrete creare voi stessi queste directory. Ricordatevi che la dimensione della memoria condivisa deve essere la stessa sia nel demone che nel pvserver.

7.0.4 Avviare il demone ed il pvserver per i test

Per i test aprite due terminali dos. Avviate il demone nella prima finetra ed il pvserver nella seconda finestra. Ogni pvserver riconosce alcune opzioni da linea di comando.

Visualizzare le opzioni di linea di comando del vostro pvserver

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

Avviare il vostro pvserver

```
vostropvserver -cd=/path/del/vostro/progetto
```

Sel l'opzione '-cd' (cambia directory) non viene fornita il pvserver dovrebbe utilizzare la directory corrente.

7.1 Modbus

Se avete bisogno di maggiori informazioni sulle basi del Modbus vi preghiamo di leggere prima questo `http://en.wikipedia.org/wiki/Modbus` .
Il protocollo Modbus ha il vantaggio che è una specifica pubblicamente disponibile e quindi il 'Reverse engineering' del protocollo non è necessaria per poterlo implementare. Il modbus è diventato così molto popolare e ci sono molti componenti hardware che supportano questo protocollo. Il Modbus è il protocollo preferito in molti sistemi come anche in pvbrowser.
Il Modbus è disponibile in diverse versioni ma sul bus c'è sempre un solo master e fino ad un massimo di 255 slave. Questo protocollo può essere utilizzato su una connessione seriale con RS485. In questo caso possono essere utilizzati sia il Modbus RTU che il Modbus ASCII. E' anche possibile utilizzare il Modbus su TCP. Entrambi possono essere combinati utilizzando un gateway che implementa una slave Modbus TCP sulla rete ed è un RTU Modbus master sul bus di campo RS485.
La classe rlModbus di rllib implementa tutte queste versioni. Registrando un rlSerial o un oggetto in rlSocket rlModbus si può scegliere se utilizzare la linea seriale o TCP. rlModbus è la base per l'implementazione del demone Modbus.
Riguardo a rlModbus tutti gli indirizzi dei dati nei messaggi Modbus sono referenziati a 0, con la prima occorrenza di un dato indirizzato come entita' numero zero. Inoltre, un campo codice funzione specifica già su quale gruppo di registro si opera (cioè 0x, 1x, 3x, o 4x indirizzi di riferimento). Per esempio, l'holding register 40001 è indirizzato come registro 0000 nel campo dati indirizzo del messaggio. Il codice funzione che opera su tale registro specifica un'operazione su holding register ed il riferimanto al gruppo 4xxxx è implicito. Così, l'holding register 40108 in realtà è indirizzato come registro 006BH (107 decimale).

Mappa dei registri Modbus

```
0xxxx Leggi/Scrivi Uscite discrete o Coils.
1xxxx Leggi Ingressi discreti.
3xxxx Leggi Registri ingresso.
4xxxx Leggi/Scrivi Uscite o Holding Registers.
```

7.1.1 Accesso utilizzando caratteri ASCII leggibili

Nella directory pvbaddon/daemons/modbus/client di pvbaddon si trova un demone modbus che utilizza caratteri ASCII leggiili. Nel file INI si specifica cosa il demone dovrebbe leggere.

File INI per demone modbus

```
# 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

[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)
```

Con USE_SOCKET viene scelto se usare una linea seriale oppure TCP. Con USE_SOCKET=1 solo la sezione [SOCKET] è rilevante e con USE_SOCKET=0 la sezione [TTY]. Con DEBUG alcuni messaggi del demone vengono attivati/disattivati. Nella sezione [RLLIB] vengono specificate la memoria condivisa (shared memory) e la casella postale. Si prega di notare che questo esempio è per i sistemi operativi di tipo unix. Utilizzando Windows [TTY][DEVICENAME], [RLLIB][SHARED_MEMORY] e [RLLIB][MAILBOX] devono essere specificati con la sintassi di Windows.

Esempio per la sintassi Windows

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

Si prega di notare che le directory per la memoria condivisa e la casella postale devono già esistere.
Nella sezione [CYCLES] dell'esempio sono definiti 4 cicli. Nel CYCLE1 10 inputStatus contigui dallo Slave=1 e address=0 sono letti. Nel CYCLE2 8 coilStatus contigui dallo Slave=1 e address=0 sono letti. Nel CYCLE3 2 holdingRegister contigui dallo Slave=1 e address=0 sono letti. Nel CYCLE4 2 inputRegister contigui dallo Slave=1 e address=0 sono letti.
Dopo aver scritto il file INI per il vostro sistema dovrete avviare il demone con DEBUG=1 dal prompt della linea di comando (shell). Per mezzo dell'opzione di DEBUG l'output del demone ci aiuterà a verificare se i dati vengono letti correttamente.
Nella directory pvbaddon/daemons/modbus/pvs in pvbaddon si trova un pvserver, nel quale viene mostrato come accedere alla memoria condivisa (shared memory) ed alla casella postale.

Accedere alla memoria condivisa e casella postale da un 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 Accesso a dati codificati binari

In pvdevelop vi è una finestra di dialogo per la generazione di un demone modbus che usa valori codificati binari. In questa finestra è possibile specificare che cosa deve essere letto. Con 'communication=serial' o 'communication=socket' si può scegliere quale interfaccia viene utilizzata per la comunicazione Modbus. Si prega di inserire un commento di fronte al metodo che non si desidera utilizzare. Anche in questo caso è possibile definire diversi 'cycle' nei quali il modbus legge ciclicamente ed i valori vengono scritti nella memoria condivisa.

Si prega di notare che sotto Windows è necessario utilizzare la sintassi di Windows. Si deve fare attenzione che '\' nel path deve essere scritto due volte '\\' in quanto il testo inserito viene tradotto in codice sorgente C++. Quando si chiude la finestra di dialogo il file modbusdaemon.cpp viene generato nella directory corrente e viene compilato. Il risultato è un demone modbus che riflette la configurazione inserita nella finestra di dialogo. Questo demone può ora essere utilizzato. Si prega di avviare questo demone dalla riga di comando per verificare se funziona come desiderato. Le variabili lette ciclicamente vengono scritte nella memoria condivisa una dopo l'altra (compatte). L'offset per ogni ciclo viene definito nel file modbusdaemon.h generato. Il file modbusdaemon.h contiene alcune definizioni che è possibile utilizzare nel vostro codice sorgente. Queste definizioni sono per esempio il nome e la dimensione della memoria condivisa e il nome della cassetta postale. Inoltre vi si trovano gli offsets della memoria condivisa secondo i quali i valori letti vengono memorizzati. Nella directory pvbaddon/demos/modbusserial in pvbaddon si può trovare un esempio.

Accesso alla memoria condivisa e alla cassetta di posta da un pvserver per valori codificati in binario

```
#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
```

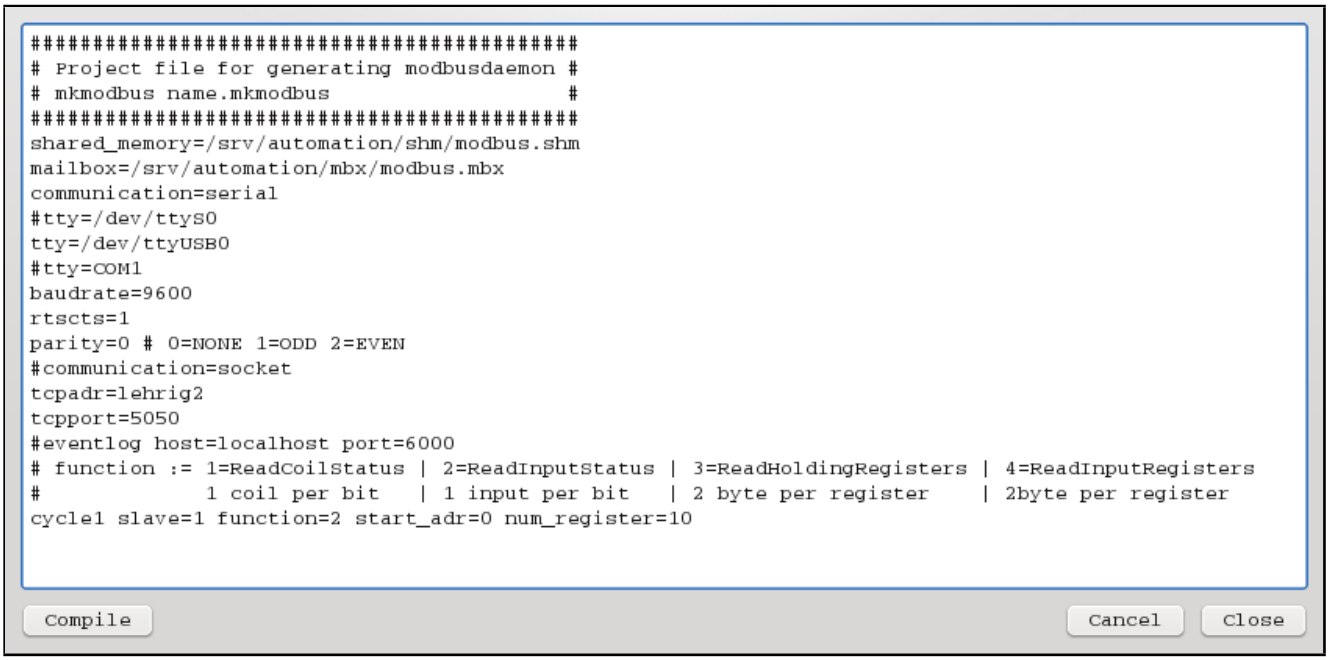

Figura 7.2: Dialogo per la generazione di un modbusdaemon all'interno di pvdevelop

7.2 Siemens

I demoni per Siemens S7 e S5 sono analoghi al demone Modbus. Pertanto si prega di leggere la sezione relativa al Modbus.

7.2.1 Accesso via TCP con caratteri codificati ASCII

Nella directory pvbaddon/daemons/siemenstcp/client in pvbaddon si trova un demone Siemens per le connessioni TCP.
Qui viene utilizzata la classe rlSiemensTCP di rllib.
Il file INI è simile al file INI del modbus. È possibile specificare tante connessioni PLC quante si desidera con un solo demone. Haveto_SWAP dovrebbe essere impostato a 1 su una CPU x86. Se il processore utilizza un diverso ordine di byte è possibile impostare questo valore a 0. Per ogni valore SLAVE vi è un indirizzo IP o il nome per il PLC poi c'è una virgola e il tipo di PLC. Come parametro opzionale è possibile scegliere se utilizzare il protocollo Fetch/Write o il più recente protocollo TCP Siemens. Nel file INI NUM_CYCLES=1 è impostato in quanto per scopi di test solo il primo ciclo viene utilizzato. Se anche il secondo ciclo dove essere eseguito è necessario impostare NUM_CYCLES = 2.

File INI per PLC Siemens su 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 Accesso su PPI utilizzando caratteri leggibili ASCII

Nella directory pvbaddon/daemons/siemensppi/client in pvbaddon si trova il demone Siemens per le connessioni PPI (linea seriale).
In questo caso utilizziamo libnodave `http://libnodave.sourceforge.net/` Una copia di libnodave è inclusa nel nostro package.
Il file INI è simile al file INI modbus.

File INI per PLC Siemens su 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 Demoni per Siemens TCP e PPI generati da pvdevelop

Come per il modbus è possibile generare un demone per PLC Siemens da pvdevelop. Il file generato e compilato è siemensdaemon.cpp ed è nella directory corrente.

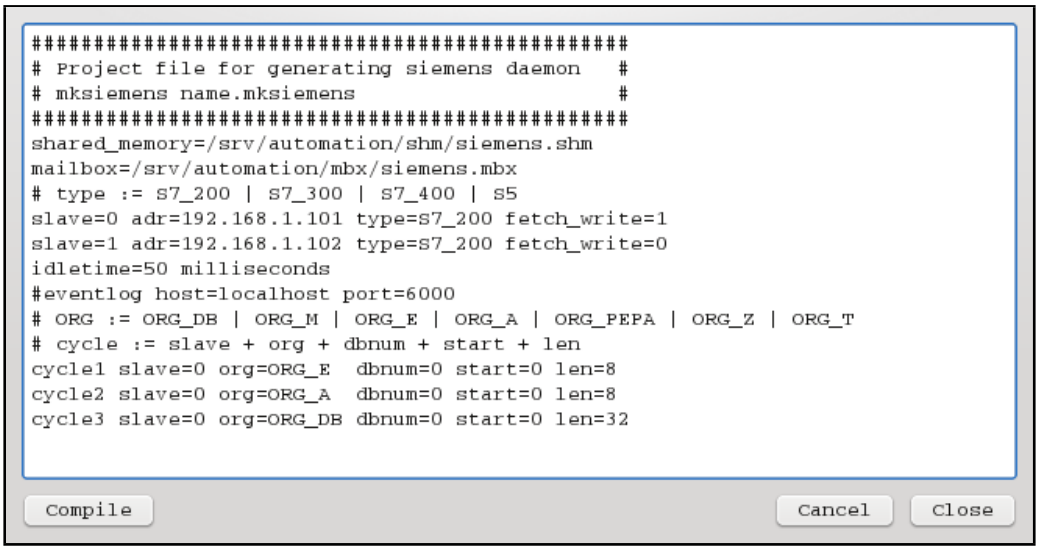

Figura 7.3: Dialogo per generare un demone TCP per Siemens da pvdevelop

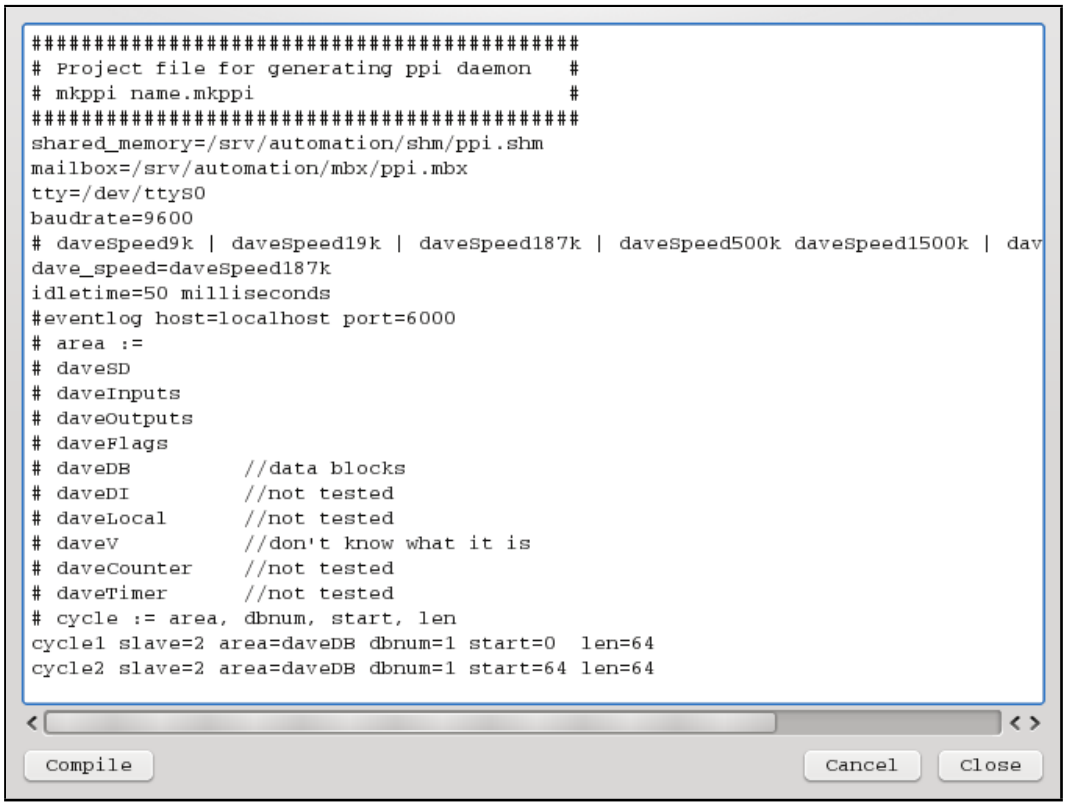

Figura 7.4: Dialogo per generare un demone TCP per Siemens da pvdevelop

7.3 EIB Bus

Nella directory pvbaddon/daemons/eibnet/client in pvbaddon si trova un demone EIBnet che è utilizzato per comunicare con il european installation bus su gateway TCP/EIB con EIBnet.
Quì viene utilizzata la class rlEIBnetIP di rllib. Il file INI è simile al file INI modbus. L'indirizzo IP del gatewy e l'indirizzo del vostro computer deve essere specificato.

Le variabili su bus EIB non vengono specificate in modo esplicito, perché il demone inserirà tutti i dati trasmessi attraverso il bus nella memoria condivisa.

File INI per 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

[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

Nella directory pvbaddon/daemons/ethernetip/client in pvbaddon si trova il demone che implementa il protocollo Ethernet/IP che è utilizzato da Allen Bradley e Rockwell.
Qui viene utilizzato il progetto open source TuxEip. Sfortunatamente sembra che questo lo sviluppo si questo progetto sia fermo.
Il file INI è simile al file INI del modbus.

File INI per 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 e CAN

Per il protocollo Profibus e il CAN preferiamo le schede CIF cards della Hilscher `http://hilscher.com`.
Queste schede hanno un controller separato che gestisce il protocollo fieldbus. Le schede memorizzano le variabili di processo in una memoria dual ported alla quale hanno accesso sia il PC che il controller. Con il tool di configurazione SyCon della Hilscher si definisce dove, all'interno della memoria dual ported, una variabile di processo è immagazzinata.
Le nuove schede basate su netX lavorano sullo stesso principio ma supportano praticamente ogni protocollo fieldbus popolare. Il processore ARM integrato in queste schede viene caricato con il firmware per il protocollo.
In pvbrowser la classe rlHilscherCIF è un wrapper per il driver Hilscher. Siccome queste schede già memorizzano le variabili di processo nella memoria dual ported non usiamo la nostra memoria condivisa e la cassetta postale. Invece avviamo un thread separato all'interno del nostro pvserver in cui la scheda viene letta ed è possibile inviare i dati al fieldbus (bus di campo).
Nella directory pvbaddon/demos/hilschercif di pvbaddon si trova un'esempio.
Le variabili globali sendData e receiveData possono quindi essere utilizzate nella visualizzazione. Con pbus.lock () e pbus.unlock () è possibile sincronizzare il mutuo accesso alle variabili da diversi clients.

Thread separate per le schede Hilscher

```
#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 è la prima versione di piattaforma indipendente OPC. In contrasto con l'OPC classico che si basa su COM/DCOM di Microsoft OPC XML-DA opera su richieste http con XML e quindi può lavorare su ogni sistema operativo.
Nella directory pvbaddon/daemons/opcxmlda/client di pvbaddon si trova il nostro demone che implementa un client OPC XML-DA.
Utilizzando questo client prima leggete la directory oggetto del vostro server OPC XML-DA.

Lettura di una object directory OPC XML-DA

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

Esempio del contenuto di una opcxmlda.itemlist

```
#./opcxmlda_client http://172.16.96.128/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 questa itemlist si possono commentare le variabili che non ci interessano.

Utilizzo di 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]   l' url del server OPC XML-DA.
[METHOD]   il metodo da chiamare. [METHOD] := GetStatus | Browse | Run
[URL] e [METHOD] sono obbligatori e devono essere i primi 2 parametri.

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

Esempio per la creazione di una opcxmlda.itemlist:
./opcxmlda_client http://server/opcxmlda/xmldaserver Browse > opcxmlda.itemlist
```

Se gli argomenti di default della riga di comando sono sufficienti una chiamata a opcxmlda_client potrebbe essere come segue.

Esempio di una chiamata a opcxmlda_client

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

Nel vostro pvserver è necessario utilizzare la classe rlOpcXmlDa da rllib. Un esempio di tale pvserver può essere trovato nella directory pvbaddon/daemons/opcxmlda/PVS di pvbaddon.

7.7 Utilizzo di un gateways

Se un protocollo per bus di campo non fosse disponibile in pvbrowser potrebbe essere possibile utilizzare un'opportuno gateway. Molto probabilmente si utilizzerà il protocollo Modbus sul lato PC.
Alcuni esempi :
Modbus LON Gateway
`http://www.intellicom.se/lonworks_modbus_rtu.cfm`
Profibus Modbus-TCP Gateway da 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 per ulteriori protocolli

Il demone modbus nella directory pvbaddon/daemons/modbus/client di pvbaddon può essere utilizzato come modello per la acquisizione dati.
Ancora più semplicemente si può utilizzare il modbusdaemon.cpp che viene generato da pvdevelop come modello per sviluppare la vostra acquisizione dei dati.

Se non si desidera utilizzare la memoria condivisa e la cassetta postale è possibile avviare un thread separato all'interno del pvserver per la lettura dei dati di processo. Si prega di dare un'occhiata al 'Profibus e CAN' per vedere come è possibile fare questo.

Capitolo 8

Avviare un pvserver in background

Dovrebbe essere possibile avviare un pvserver in background al momento del boot. Il metodo per fare questo dipende dal sistema operativo utilizzato.
Con un pvserver si può scegliere per mezzo del simbolo USE_INETD del preprocessore se usare un server multi-threaded o se per avviare il pvserver con l'aiuto di (x)inetd.
Il server multi-threaded è composto da più thread. Il thread principale resta in attesa di nuovi client. Se un client si connette un nuovo thread viene avviato per servire il client. Il thread principale può restare in attesa di più clienti.
(X)inetd è un super server che resta in attesa dei client e quindi avvia il vero server biforcando in un nuovo processo. Il STDIN e STDOUT sono utilizzati per la comunicazione. (X)inetd ridireziona STDIN e STDOUT del server per la connessione di rete.

8.1 Linux

Per il pvserver multi threaded è possibile generare uno 'startscript' con pvdevelop utilizzando il menu 'Linux-WriteStartscript'.

Lo script di avvio generato da 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
```

Le variabili DAEMON_PATH e DAEMON possono essere adattate. Lo 'startscript' deve essere copiato in /etc/init.d dove tutti gli scripts di avvio dei server sono salvati. Ora è possibile effettuare le seguenti operazioni:

Uso dello startscript

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

Per avviare il VOSTRO_PVS nel runlevel corretto deve essere creato un link in 'rc5.d'. In openSUSE questa operazione può essere eseguita con YaST nel 'Runlevel Editor'. Scegliete VOSTRO_PVS e semplicemente attivatelo.
Se volete usare inetd o xinetd dovete effettuare le seguenti operazioni. Installare e attivare (x)INTED. Nel file /etc/services aggiungete le seguenti righe.

Definizione di un pvserver sulla porta 5051

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

Questo definisce un servizio pvsuper sulla posta 5051. In /etc/xinted.d si ha bisogno del seguente 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
}
```

Per attivare il pvserver si deve riavviare (x)inted.

Riavviare xinetd

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

8.2 Windows

Per avviare un pvserver come un 'servizio Windows' in background si può utilizzare xyntservice `http://www.codeproject.com/KB/system/xyntservice.aspx` .
L'uso di inetd non è possibile su Windows perchè non esiste un inetd standard disponibile per Windows.
Se si desidera avviare il pvserver dal 'Windows Autostart Directory' apparirà una 'DOS Box' in cui il vostro pvserver è in esecuzione. Se non ti piace la 'DOS Box' è possibile utilizzare il piccolo strumento PVB/winMinGW/bin/start_pvbapp.exe

8.3 OpenVMS

Si può usare loginout.exe per avviare il vostro pvserver in modo multi threaded. Si può inserire questo comando 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
$
```

È possibile utilizzare UCX SET SERVICE per avviare un pvserver in modalità inetd.

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
```

Il file di avvio per il servizio di ucx appare come segue.

pvserver_startup.com

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

8.4 pcontrol

In un tipico sistema di automazione si possono avere più processi che costituiscono la vostra automazione. La maggior parte di questi processi viene eseguito in background. Deve esistere un metodo per il monitoraggio e il controllo di tali processi. Inoltre dovrebbe essere possibile inviare messaggi di log di eventi a un'istanza centrale. I messaggi di log degli eventi deveno essere leggibili online e offline da archivi storici.

pcontrol è un pvserver che si basa su rllib e usa pvbrowser per rispondere completamente queste esigenze. Il 'pvserver - pcontrol' avvia i processi in background e li monitora. Se si desidera avviare automaticamente un complesso sistema di automazione in background è possibile utilizzare pcontrol. Pcontrol deve essere avviato in background, come descritto nella sezione precedente. Tutti gli altri processi possono ora essere avviati e controllati da pcontrol.

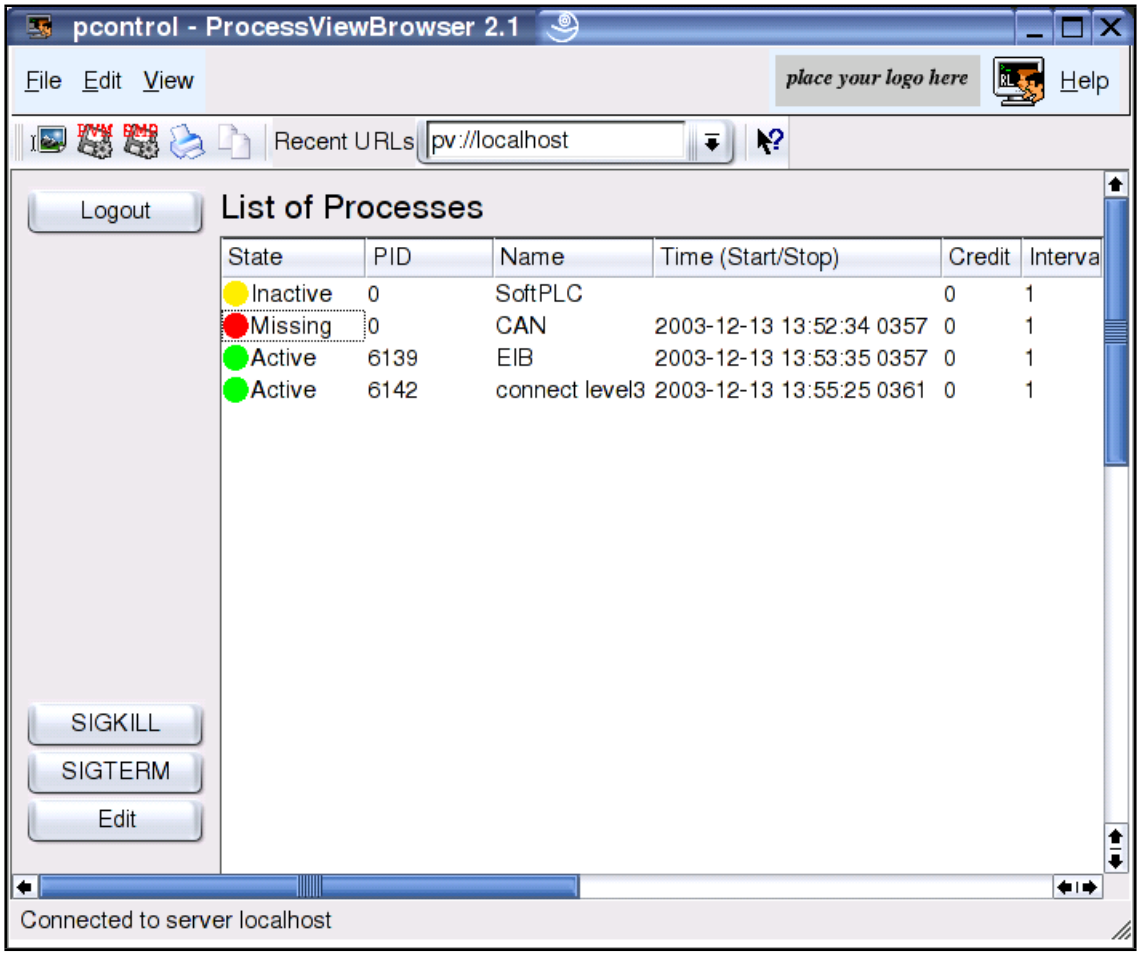

Figura 8.1: pcontrol controllo i processi in background

Con pcontrol è possibile ricevere messaggi di log eventi dagli altri processi (anche in rete) In rllib ci sono alcune funzioni che consentono a tali processi di inviare messaggi di log eventi. In primo luogo si definisce il server pcontrol con rlEventInit(). Quindi è possibile utilizzare rlEvent() per inviare messaggi di log simili a testo di output con printf(). La data/ora, il nome del file di origine e la riga nel file di origine in cui l'output è stato generato saranno inclusi nel messaggio di log eventi.

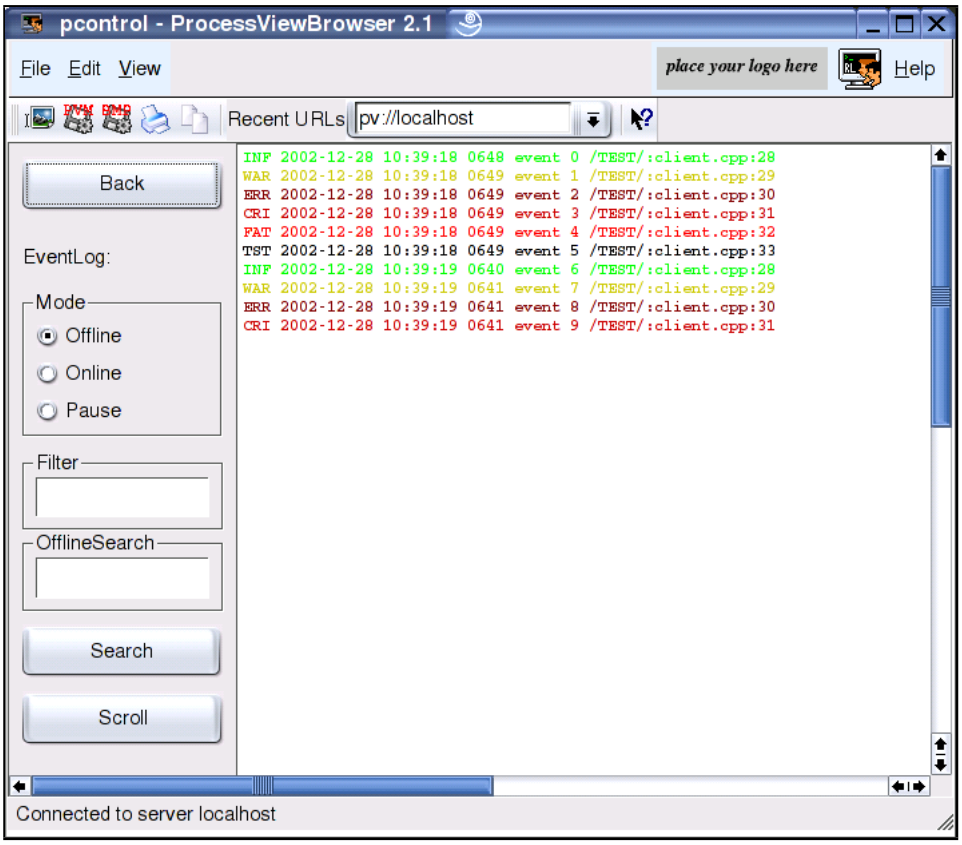

Figura 8.2: Si possono visualizzare i messaggi di log in pvbrowser

Outputs di messaggi di log

```
#include "rlevent.h"

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

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

8.5 Controllo degli accessi

Le funzioni seguenti vi permetteranno di definire quanti clients potranno connettersi al vostro pvserver da 1 indirizzo IP e quanti clients in totale avranno accesso al vostro pvserver in contemporanea. Il valore di

'max_clients' deve avere un valore tra 1 ed il valore di MAX_CLIENTS. Si consiglia di inserire queste funzioni in main(). Di default questi valori sono impostati su MAX $_CLIENTS = 100$.

Impostazione del numero massimo di clients ammessi

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

Al fine di ottenere una connessione sicura ad un pvserver tramite Internet può essere utile consentire esplicitamente alcuni intervalli di indirizzi IP o respingere un certo range di IP. Se si desidera utilizzare questi filtri è necessario creare il file 'deny.ipv4' ed il file 'allow.ipv4' all'interno della directory dove il vostro pvserver è in esecuzione.
Se non esiste nessuno di questi file ogni indirizzo IP è autorizzato a collegarsi.
Se esiste solo il file 'allow.ipv4' solo gli indirizzi IP in esso esplicitamente dichiarati potranno connetersi.
Se esiste solo il file 'deny.ipv4' solo gli indirizzi IP in esso non esplicitamente dichiarati potranno connetersi.
Se entrambi i file esisto il controllo degli accessi si comporta come se ci fosse solo il file 'allow.ipv4'.

Esempio: allow.ipv4

```
# Commento
# Qui potete inserire una lista di range IP che sono autorizzati a connettersi.
# Un indirizzo / range di indirizzi valido ha il seguente formato aaa.bbb.ccc.ddd/
    numero_di_bit_significativi.
192.168.1.0/24 # permette le connessioni dalla subnet privata 192.168.1.X
192.168.0.14/32 # permette le connessioni dall'indirizzo IP 192.168.0.14
# inserite quanti range desiderate.
```

Esempio: deny.ipv4

```
# Qui potete inserire una lista di range IP che non sono autorizzati a connettersi.
# Un indirizzo / range di indirizzi valido ha il seguente formato aaa.bbb.ccc.ddd/
    numero_di_bit_significativi.
100.101.0.0/16 # nega le connessioni da 100.101.xxx.yyy
200.201.0.0/16 # nega le connessioni da 200.201.xxx.yyy
```

Un possibile utilizzo potrebbe essere di consentire le connessioni al vostro pvserver solo dalla subnet privata. Se devono essere eseguite operazioni di manutenzione con connessioni da internet l'operatore dovrà autorizzare l'indirizzo IP del tecnico.
Le funzioni per IPv6 useranno i file 'allow.ipv6' e 'deny.ipv6' e funzioneranno come per l'IPv4.

Esempio: allow.ipv6

```
# Qui potete inserire una lista di range IP che non sono autorizzati a connettersi.
# Un indirizzo / range di indirizzi valido ha il seguente formato indirizzo_ipv6/
    numero_di_bit_significativi.
::1/128
0001:0002:0003:0004:0005:0006:0007:0008/128
0000:0000:0000:0000:0000:0000:1000:0000/128
# inserite quanti range desiderate.
```

Capitolo 9

Ulteriori suggerimenti

Pvbrowser è un framework per HMI/SCADA scritto in C/C++ . In questo modo non si è limitati dalle opzioni sopra descritte ma si è in grado di implementare ulteriormente da soli le caratteristiche con l'uso di librerie esterne.

9.1 Usare qualsiasi sistema di database

Per l'utilizzo di sistemi database si consiglia la libreria Qt perché questa viene fornita con il kit di sviluppo che usiamo con pvbrowser.
Il 'SQL Module' di Qt offre una comoda API per i database SQL. Con l'uso dei plugin vengono supportati quasi tutti i sistemi database più popolari. Per poter utilizzare la libreria Qt nel vostro pvserver è necessario rimuovere 'CONFIG -= qt' dal file di progetto. Poi qmake lincherà la libreria Qt e sarà possibile utilizzare le classi SQL Qt.
Un esempio per l'utilizzo del modulo SQL Qt in un pvserver può essere trovato in pvbaddon.tar.gz Vedi la directory:
pvbaddon/templates/qtDatabase

9.2 Utilizzo di un foglio di calcolo sul client

Se si desidera utilizzare un foglio elettronico al fine di consentire agli utenti di fare le proprie elaborazioni e valutazioni è possibile creare un file CSV sul vostro pvserver e copiarlo con pvDownloadFile() sul computer client. La funzione pvClientCommand richiamerà il programma di foglio elettronico per aprire questo file CSV.

pvClientCommand

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

Il 'visualizzatore-csv' è definito nelle opzioni del client pvbrowser. Qui è possibile inserire OpenOffice o Excel per esempio.

9.3 Generare reports con LaTeXcreando file PDF

Per la documentazione ed il controllo della qualità della produzione è spesso necessario generare dei report. Un buon formato per i report è il formato PDF. Ad esempio si potrebbe installare LaTeXsul proprio server, creare dei modelli di documento in LaTeXed inserire all'interno dei modelli i risultati della produzione. Dopo aver creato un file PDF con l'uso di LaTeXè possibile copiarlo con pvDownloadFile() sulcomputer client.
La funzione pvClientCommand può essere utilizzata per avviare il visualizzatore PDF con questo file.

pvClientCommand

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

Il 'visualizzatore-pdf' è definito nelle opzioni del client pvbrowser. Qui si puo inserire Okular o Acrobat Reader per esempio.

9.4 Valutazione statistica

Ci sono un sacco di programmi statistici che vengono eseguiti dalla riga di comando e generano grafica bitmap come output. È possibile integrare questi programmi statistici nello stesso modo degli strumenti di plottaggio esterno. Con i sistemi Unix-like sul lato server si consiglia di utilizzare la classe rlSpawn da rllib. Questa classe avvia un programma esterno e collega STDIN e STDOUT su una pipe con l'istanza rlSpawn. È ora possibile controllare l'applicazione esterna utilizzando rlSpawn dal pvserver. I risultati delle statistiche possono essere incluse nella visualizzazione o nei report generati con LaTeX.

9.5 Ram disc per directory temporanea di pvbrowser

Il client pvbrowser utilizza una directory temporanea per la memorizzazione dei file. Questa directory temporanea è specificata all'interno delle opzioni del client pvbrowser. È possibile impostare questa directory in un disco RAM per l'ottimizzazione. Inoltre questo avrebbe il vantaggio che questa directory è vuota dopo ogni riavvio del computer.

Creazione di un disk con capacità di 512MB su Linux nella sotto directory ram

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

Capitolo 10

Conclusioni

In questa documentazione vi abbiamo descritto pvbrowser. Con questo framework è possibile creare velocemente e in modo relativamente facile visualizzazioni e acquisizione di dati. Il tempo necessario per la creazione di una visualizzazione può essere paragonato allo sforzo di creare una pagina web. Le visualizzazioni create con pvbrowser sono sempre in grado di lavorare su una rete. Simile ad un browser web pvbrowser può essere utilizzato per saltare da una visualizzazione/pvserver ad un'altra e quindi di visualizzare e controllare un impianto completo con questo sistema distribuito. Poichè la comunicazione è su TCP/IP pvbrowser può essere utilizzato anche su internet.
Bisogna esplicitamente avvertirvi di non aprire le parti critiche di controllo di un impianto su Internet. Una visualizzazione a scopo interno dovrebbe usare porte che siano bloccate da un accesso a Internet da un firewall. Le parti della visualizzazione a cui si può accedere via Internet dovrebbero consentire la visualizzazione dello stato, ma non consentire di comandare l'impianto. Se avete bisogno di controllare l'impianto da internet si dovrebbe usare il protocollo pvssh:// (Secure Shell) e utilizzare una password complessa. Sarebbe una buona idea terminare la connessione se qualcuno inserisce una password errata inibendo così gli 'attacchi di forza bruta'.
Poichè pvbrowser può essere eseguito su tutti i sistemi operativi più diffusi non ci sono quasi limiti. Gli esperti possono utilizzare i vantaggi di Unix come sistema operativo per far funzionare il server considerando che gli utenti normali possono utilizzare i loro sistemi operativi. Una visualizzazione può essere avviata anche in modalità a schermo intero in cui gli utenti non possono sapere quale sistema operativo vi è alle spalle.
Ci auguriamo che possiate utilizzare pvbrowser con successo. Inoltre saremmo lieti di ricevere suggerimenti per nuove caratteristiche, segnalazioni di bug, il vostro contributo alla documentazione, l'invio di patch o che contribuiate anche allo sviluppo di nuovi driver/daemon per ulteriori protocolli.
Vi auguriamo successo usando pvbrowser.

La nostra comunità pvbrowser

`http://pvbrowser.org` 31 ottobre 2013

Appendix

Elenco delle figure