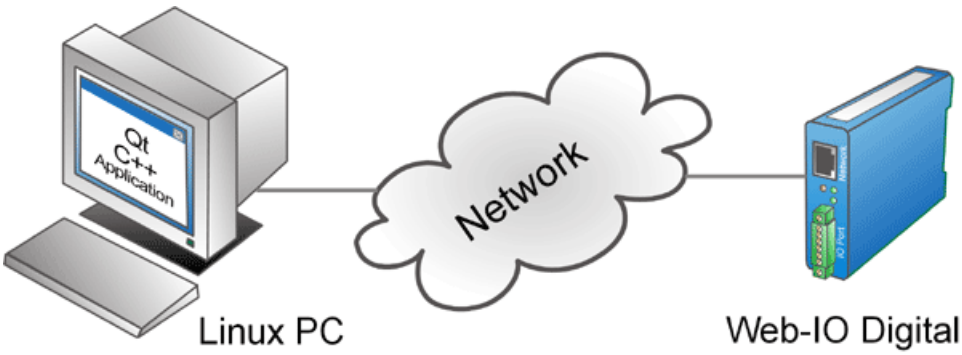
Application for the Web-IO Digital:

Control and monitor Web-IO Digital with C++ under Linux

The application described in the following enables you to control the Web-IO under Linux. The user interface was assembled using Version 3.3.5 of Qt-Designer.



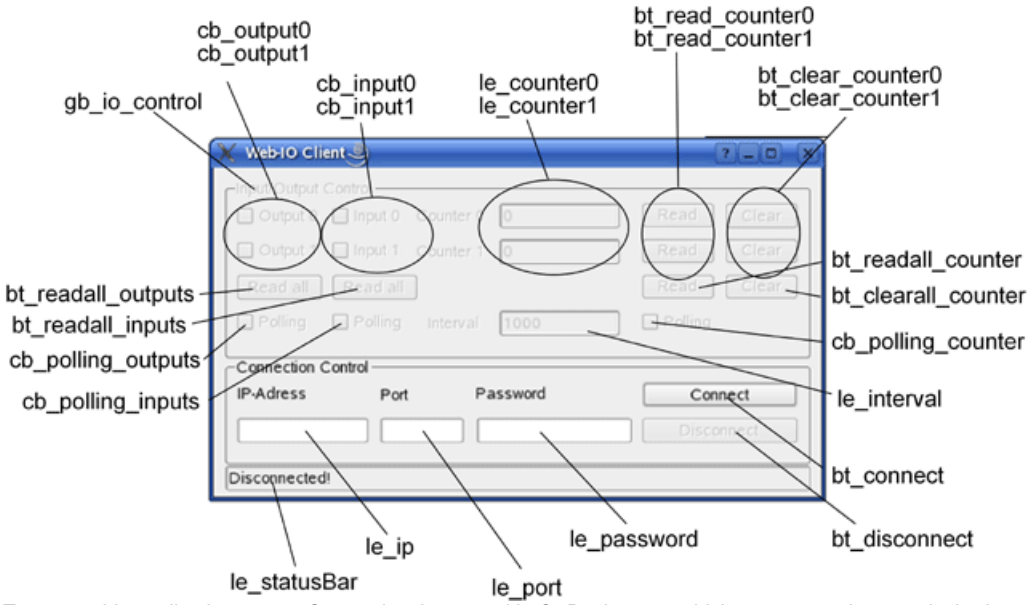
Using the following C++ program example you can represent your Web-IO Digital with its inputs and outputs in a Linux application. You can also switch the outputs on the Web-IO.

Preparations

You have already provided your Web-IO Digital

- with power,
- configured the inputs and outputs,
- connected it to your network,
- assigned it an IP address

1. Combining the various operating elements and display objects in Qt-Designer



To create this application, a new C++ project is created in Qt-Designer to which you can assign any desired name. We are using the name "Client". In the next step a main window is generated in which the graphical elements are placed and named according to their property.

To make compiling the application easier under a Unix system, we create two files called "Uic" and Qmake". This gives us functional C++ source text, which one can easily compile using a makefile.

You create a new text file and write the two following lines to it. Then save the file as "Uic".

Note: These lines only work if qt3 is used and installed at the same place in the system. Otherwise please adapt the two lines accordingly.

```
/usr/lib/qt3/bin/uic $1.ui -o $1.h
/usr/lib/qt3/bin/uic -impl $1.h $1.ui -o $1.cpp
```

Now you create another text file which is saved as "Qmake".

```
/usr/lib/qt3/bin/qmake -o Makefile $1.pro
```

Once both files have been created, place them both in the directly containing the project. Then configure the two files so that they are executable. For this you can use the chmod command.

```
-> chmod a+x Dateiname
```

In the following the project can be successfully converted into C++ source code and compiled. This only take three additional steps.

```
./Uic Dateiname(.ui)
./Qmake Projekt(.pro)
make
```

Now an executable file has been created which has the same name as the project.

2. Create the source code file for the graphical interface

If Qt-Designer is started and you have the main window open, double-clicking on the window creates a .ui.h source code file. In the following you are shown in detail which steps are necessary to get this application running.

To create new slots, right-click on the main window to open a properties window, and select the Slots entry. There you create the required slots, which then automatically appear with an empty body in the generated .ui.h file.

Now we include a few required header files. The clientDlg.h and the clientDlg.cpp are generated after the application with ./Uic from the .ui file (main window). To be able to access the elements, the header file is included here.

```
#include "clientDlg.h"
#include &lt;qsocket.h&gt;
#include &lt;qstring.h&gt;
#include &lt;qtextstream.h&gt;
#include &lt;qtimer.h&gt;
#include &lt;iostream&gt;
```

Then global variables for a TCP connection and polling are declared and thereby made accessible to every method in the class. Since we are not defining our own namespace, the standard namespace is used.

```
using namespace std;

QSocket* client;
QTimer* output;
QTimer* input;
QTimer* counter;
```

3. Starting the program

Setting up the operating elements
The group with the operating elements for the Web-IO is first blocked from operation. As soon as a connection is opened, all required elements are enabled. For this you can set the Enable function in the properties of the GroupBox cb_io_control, as well as for all other elements, to ’false’ or ’true’ .

The name of the respective operating element is derived from the element itself depending on the context. The first two characters in the name stand for the element type (cb -> Checkbox, bt -> Button, gb -> Groupbox and le -> LineEdit).

4. Connection control

Establishing the connection
After entering the IP address for the Web-IO in the text field li_ip and Port 80 in the text field li_port, clicking on the bt_connect button allows a connection to be opened. If no IP address or port is entered, a message is displayed in the status bar

Opening the connection
To be able to open a TCP connection, the already declared socket variable is initialized. Once everything has been correctly entered, an attempt is made to open a connection. To determine whether the connection has been established, the system waits for a signal from the socket which reports a successful connection opening.

```
void clientDlg::onConnect()
{
 client = new QSocket(this);
 bool ok;
 connect(client,
 SIGNAL(connected()), SLOT(connectDone()));
 connect(client,
 SIGNAL(error(int)),
 SLOT(connectError(int)));

 if(le_ip->text() == "")
 le_statusBar->setText("No IP entered!");
 else if(le_port->text() == "")
 le_statusBar->setText("No port entered!");
 else
 client->connectToHost(le_ip->text(), (le_port->text()).toInt(&ok,10));
}
```

If there was an error in opening the connection, a corresponding message appears in the status bar.

```
void clientDlg::connectError(int)
{
 le_statusBar->setText("Error while connecting!");
}
```

Connection is made

After successfully opening a connection, all useful operating elements for the application are enabled and the Connect button deactivated. In addition, the application immediately begins to go into receive ready mode.

```
void clientDlg::connectDone()
{
 le_statusBar->setText("Connected to " + le_ip->text());
 connect(client, SIGNAL(readyRead()), SLOT(onReceive()));

 output = new QTimer(this);
 connect(output,
 SIGNAL(timeout()),
 SLOT(timerEvent()));

 input = new
 QTimer(this);
 connect(input,
 SIGNAL(timeout()),
 SLOT(timerEvent()));

 counter = new
 QTimer(this);
 connect(counter,
 SIGNAL(timeout()),
 SLOT(timerEvent()));

 gb_io_control->setEnabled(true);
 bt_disconnect->setEnabled(true);
 bt_connect->setEnabled(false);
}
```

Disconnecting

The connection remains open until the user clicks on the Disconnect button or until it is ended by the Web-IO. After clicking on the button a message is displayed that the connection has been ended.

```
void clientDlg::onDisconnect()
{
 client->close();

 output->stop();
 input->stop();
 counter->stop();

 gb_io_control->setEnabled(false);
 bt_disconnect->setEnabled(false);
 bt_connect->setEnabled(true);
 le_statusBar->setText("Disconnected!");
}
```

When the connection is ended all elements are again blocked from being operated.

5. Operation and communication from the client side

As soon as a connection is made with the Web-IO, the user can use the corresponding program elements to send commands to the Web-IO

When sending a message to the Web-IO, an asynchronous mode is used. This ensures the application is not blocked during sending or receiving.

The outputs on the Web-IO can be switched using the two checkboxes IDC_OUTPUT0 and IDC_OUTPUT1.

In the next step a check is made to see whether the checkbox is already set and the corresponding output set to ON or OFF.

```
void clientDlg::onOutput0()
{
 QString message1 = "GET /outputaccess0?PW=" + le_password->text() + "&State=ON&";
 QString message2 = "GET /outputaccess0?PW=" + le_password->text() + "&State=OFF&";

 if(cb_output0->isChecked())
 client->writeBlock(message1, message1.length());
 else
 client->writeBlock(message2, message2.length());

 client->flush();
}
void clientDlg::onOutput1()
{
 QString message1 = "GET /outputaccess1?PW=" + le_password->text() + "&State=ON&";
 QString message2 = "GET /outputaccess1?PW=" + le_password->text() + "&State=OFF&";

 if(cb_output1->isChecked())
 client->writeBlock(message1, message1.length());
 else
 client->writeBlock(message2, message2.length());

 client->flush();
}
```

Query output/input status

```
void clientDlg::onReadallOutputs()
{
 QString message = "GET /output?PW=" + le_password->text() + "&";
 client->writeBlock(message, message.length());
 client->flush();
}
void clientDlg::onReadallInputs()
{
 QString message = "GET /input?PW=" + le_password->text() + "&";
 client->writeBlock(message, message.length());
 client->flush();
}
```

Query counters

```
void clientDlg::onReadCounter0()
{
 QString message = "GET /counter0?PW=" + le_password->text() + "&";
 client->writeBlock(message, message.length());
 client->flush();
}
void clientDlg::onReadCounter1()
{
 QString message = "GET /counter1?PW=" + le_password->text() + "&";
 client->writeBlock(message, message.length());
 client->flush();
}
```

Naturally all counter states can also be polled using one command.

```
void clientDlg::onReadallCounter()
{
 QString message = "GET /counter?PW=" + le_password->text() + "&";
 client->writeBlock(message, message.length());
 client->flush();
}
```

Reset counter

It is also possible to set the counters to 0 after reading.

```
void clientDlg::onClearCounter0()
{
 QString message = "GET /counterclear0?PW=" + le_password->text() + "&";
 client->writeBlock(message, message.length());
 client->flush();
}

void clientDlg::onClearCounter1()
{
 QString message = "GET /counterclear1?PW=" + le_password->text() + "&";
 client->writeBlock(message, message.length());
 client->flush();
}
```

Naturally all the counters can be reset together using one command.

```
void clientDlg::onClearallCounter()
{
 QString message = "GET /counterclear?PW=" + le_password->text() + "&";
 client->writeBlock(message, message.length());
 client->flush();
}
```

Since all counter states can be read or reset using one command, there must be a method implemented which processes the reply string from the Web-IO and assigns each counter in the application its specific state.

```
void clientDlg::readAndClearCounter(QString data)
{
 QString counter[12];
 int j = 0;
 int length = data.length();
 for(int i = 0; i < length; i++)
 {
 if(data[i] == ';')
   j++;
 else
   counter[j] += data[i];
 }
 le_counter0->setText(counter[0]);
 le_counter1->setText(counter[1]);
}
```

6. Data reception from the Web-IO

Process and display the received data

- All commands and requests to the Web-IO are acknowledged with a reply string. The replies have a specific structure depending on the type:
- For the outputs: output;<binary value of the output status in hexadecimal format>
- For a special output: outputx;<ON or OFF>
- For the inputs: input;<binary value of the input status in hexadecimal format>
- For a special input inputx;<ON or OFF>
- Then there is the reply string for a counter, which looks as follows.
- Counter: counterx;<decimal counter state>
- or counter;<decimal counter state 0 >; <decimal counter state 0 >; ... if you want to read all the counters at one time.
- All reply strings are finished off with a 0 byte.

In our application the method OnReceiver() is invoked to receive such a message. In this method the reply from the Web-IO is processed. The unique feature here is that this function invokes itself the entire time, i.e. is continually looking for messages which may have been sent by the Web-IO. We have established that directly after this a connection has been successfully initialized (connectDone()).

```
void clientDlg::onReceive()
{
 char buffer[50];
 client->readBlock(buffer, 49);
 QString rcv = buffer;
 bool ok;
 if(rcv[0] == 'o')
 {
  int value = rcv.right(rcv.length()-7).toInt(&ok, 16);
 if((value & 1) == 1)
  cb_output0->setChecked(true);
 else
  cb_output0->setChecked(false);
 if((value & 2) == 2)
  cb_output1->setChecked(true);
 else
  cb_output1->setChecked(false);
 }
 if(rcv[0] == 'i')
 {
  int value = rcv.right(rcv.length()-6).toInt(&ok, 16);
  if((value & 1) == 1)
  cb_input0->setChecked(true);
 else
  cb_input0->setChecked(false);
 if((value & 2) == 2)
 cb_input1->setChecked(true);
 else
 cb_input1->setChecked(false);
 }
 if(rcv[0] == 'c')
 {
 if(rcv[7] == '0') le_counter0->setText(rcv.right(rcv.length()-9));
 if(rcv[7] == '1') le_counter1->setText(rcv.right(rcv.length()-9));
 if(rcv[7] == ';') readAndClearCounter(rcv.right(rcv.length()-8));
 }
}
```

Cyclical polling of particular values

It is desirable that the status of an individual component be updated by the component itself. For this the program uses a timer which sends cyclical queries to the Web-IO at a user-defined time interval.

The cycle time is entered in the field le_interval.

Of course this also catches cases where the user enters a nonsense value, such as a negative time value. This is immediately followed by a message and the value is not applied.

7. Polling

```
bool clientDlg::checkRange()
{
 bool ok;
 int tmp = (le_interval->text()).toInt(&ok, 10);
 if(ok && tmp > 0)
 {
  le_statusBar->setText("Range changed to " + le_interval->text() + " ms!");
  return
  true;
 }
 else
 {
  le_statusBar->setText("Please type a positive integer value for the range of polling!");
  return
  false;
 }
}
```

To now carry out cyclical polling of the Web-IO states, the choose from between polling the outputs, the inputs or the counters.

Activating the checkbox cb_polling_outputs then means the outputs are polled.

```
void clientDlg::onPollingOutputs()
{
if(checkRange())
 {
 if(cb_polling_outputs->isChecked())
  output->start((le_interval->text()).toInt(), false);
 else
  output->stop();
 }
}
```

Activating the checkbox cb_polling_inputs means polling is applied to the inputs.

```
void clientDlg::onPollingInputs()
{
 if(checkRange())
 {
 if(cb_polling_inputs->isChecked())
  input->start((le_interval->text()).toInt(), false);
 else
  input->stop();
 }
}
```

To query the counters as well using polling, you can use the cb_polling_counter checkbox.

```
void clientDlg::onPollingCounter()
{
 if(checkRange())
 {
 if(cb_polling_counter->isChecked())
 counter->start((le_interval->text()).toInt(),false);
 else
 counter->stop();
 }
}
```

Three different timers were initialized which trigger an action at the set interval. To capture the events, we still need to implement a method.

```
void clientDlg::timerEvent()
{
 if(output->isActive()) onReadallOutputs();
 if(input->isActive()) onReadallInputs();
 if(counter->isActive()) onReadallCounter();
}
```

The sample program supports all common functions of the Web-IO in command string mode, optimized for the Web-IO 2x Digital Input, 2x Digital Output. For the other Web-IO models you may have to adapt the program. Additional program examples for socket programming can be found on the Tool pages for the Web-IO. A detailed description for the socket interface of the Web-IO Digital models can be found in the reference manual.

Download program example

You don’t have a Web-IO yet but would like to try the example out sometime?
No problem: We will be glad to send you the Web-IO Digital 2xInput, 2xOutput at no charge for 30 days. Simply fill out a sample ordering form, and we will ship the Web-IO for testing on an open invoice. If you return the unit within 30 days, we will simply mark the invoice as paid.

To sample orders

W&T
www.WuT.de

We are available to you in person:

Wiesemann & Theis GmbH
Porschestr. 12
42279 Wuppertal
Phone: +49 202/2680-110 (Mon.-Fri. 8 a.m. to 5 p.m.)
Fax: +49 202/2680-265
info@wut.de