



## Linux howto

Covering NI-VISA and the Agilent open source USBTMC driver.







All rights reserved. No part of this document may be reproduced, stored in a database or retrieval system, or published in any form or way, electronically, mechanically, by print, photo print, microfilm or any other means without prior written permission from the publisher.

All correspondence regarding copyrights :

Admesy B.V.

Beneluxstraat 7

6014CC Ittervoort

The Netherlands

Tel : +31 (0) 475 600232

Fax : +31 (0) 475 600316

URL : <http://www.admesy.nl>

# Contents

1	Admesy products and drivers	5
2	Installation of NI-VISA on Linux	6
2.1	Fedora 10 (both 32 and 64bit)	6
2.2	Fedora 7 & 8	6
3	Adding the Colorimeter to the NI-VISA configuration.	7
4	Installing the Linux application.	10
4.1	Installing the Labview runtime engine.	10
4.2	Installing the Colorimeter application.	10
5	Linux USBTMC kernel driver	12
6	Libusb	12
7	Using the Agilent open source USBTMC driver.	12
7.1	Installing and testing the Agilent USBTMC driver	12
7.2	Writing a small C program to control the Brontes	13
7.3	Graphical scripting for the Brontes	16



# 1 Admesy products and drivers

Admesy creates devices which are compatible with standards like the USBTMC class for Test & Measurement devices. One of the reasons to do so is the choice of drivers for the end user and the availability of these drivers on multiple platforms.

Currently there are a number of drivers that work with our products. Please note however that Admesy uses NI-VISA for all supplied applications.

The following list shows driver availability per platform :

Operating system	Driver name	Source	Comment
Windows	NI-VISA	National Instruments	
Windows	Libusb	Open source	
Linux	NI-VISA	National Instruments	
Linux	Kernel USBTMC driver	Open source	Kernel >= 2.6.28
Linux	Agilent driver	Open source	
Linux	Libusb	Open source	
OSX	NI-VISA	National Instruments	
OSX	Libusb	Open source	

Note that for embedded systems the driver support is processor dependant. NI-VISA should work on Win-CE but currently the best choice for embedded systems would be Linux with 4 choices of available drivers.

## 2 Installation of NI-VISA on Linux

In order to use the Brontes colorimeter, it is necessary to have a correct installation of NI-VISA. The installation depends on the nikal module, which should be loaded correctly at system initialisation. In case there are any trouble installing NI-VISA, it's best to search the NI forums since most issues probably already have been reported.

In short the installation comes down to the following steps :

- 1) Make sure you have the kernel source or kernel headers installed  
(On Fedora 10 it's sufficient to install the kernel-headers package for your kernel)
- 2) Install NI-VISA
- 3) if NIKAL fails in the NI-VISA install, try the latest NIKAL version  
(happened on Fedora 10, but NIKAL 1.9 worked).
- 4) Add the Admesy device to the configuration by using the AddUsbRawPermissions.sh script.
- 5) Install the colorimeter software and start using the colorimeter.

### 2.1 Fedora 10 (both 32 and 64bit)

#### Install NI-VISA 4.4:

The linux drivers are provided as iso packages. These can either be burned on CD or directly mounted and installed. The following describes to mount the package and install it.

```
$ mkdir /mnt/iso
$ mount -t iso9660 NI-VISA-4.4.0.iso /mnt/iso -o loop
$ cd /mnt/iso
$ ./INSTALL
```

(after installation : \$ umount /mnt/iso)

installation starts and should finish. Unfortunately for Fedora 10 NIKAL fails so it was necessary to install an updated version of NIKAL.

```
$ mount NIKAL19.iso
$ cd /mnt/iso
$ ./INSTALL
```

After this, the drivers should be installed. It may be necessary to restart the system (although that was wasn't necessary during our tests).

Please continue with chapter 2 : Adding the Colorimeter to the NI VISA configuration.

### 2.2 Fedora 7 & 8

Please note that the below text is outdated and the installation as mentioned in the previous chapter should be tested first, also for Fedora 7, 8 and 9. The following text is provided as it was valid with older NI-VISA and NIKAL versions.

Admesy has tested the installation on Fedora 7 and Fedora 8, which both needed modified nikal.c source. (remove or comment the line saying "pciDriver->enable\_wake = NULL" and it will compile on Fedora 8). Also, make sure that after you install the kernel source RPM that you prepare the source for use :

```
rpmbuild -bp --target=$(uname -m) /usr/src/redhat/SPECS/kernel.spec
```

After this, copy the config file of your current kernel to the kernel source directory. For example :

```
cd /usr/src/redhat/BUILD/kernel-2.6.23/linux-2.6.23.i686/configs/kernel-2.6.23.15-i586.config .config
```

After this, installation can be continued. When upgrading a kernel, the updateNIdrivers script can be used to update the NI driver for the new kernel.

After installation of the drivers either reboot the pc or restart the nival service :  
/sbin/service nival restart

### 3 Adding the Colorimeter to the NI-VISA configuration.

Using the NI script AddUsbRawPermissions.sh it is possible to add the Brontes colorimeter to your VISA configuration. The following shows a recoding of this session. Make sure you use the exact same VID's & PID's. The 0x0E92 device is only needed in case you intend to upgrade firmware on your Linux station.

```
# cd /usr/local/vxipnp/linux/NIvisa/USB/  
# ./AddUsbRawPermissions.sh
```

NI-VISA : Add Permissions for a Raw USB Device

This Script gives permission to all users to access a specific raw USB device using NI-VISA. Under the default configuration of most Linux distributions, only the root user can access all of the USB devices. Running this script is not necessary if NI-VISA will be run only by the 'root' user or if usbfs (formerly known as usbdevfs) is mounted with the "devmode=0666" option.

"udev" detected.

The Vendor ID (VID) of a USB device is a numeric identifier that may be up to 4 hexadecimal digits. It is found in the "device descriptor" of the USB Device. An example is "0xA1B2".

What is the Vendor ID (VID) of the USB Device? 0x1781

The Product ID (PID) of a USB device is a numeric identifier that may be up to 4 hexadecimal digits. It is found in the "device descriptor" of the USB Device. An example is "0xA1B2".

What is the Product ID (PID) of the USB Device? 0x0E92

The following USB Device information has been entered:

```
Vendor ID (VID) : "0x1781"  
Product ID (PID) : "0x0E92"
```

Continue? [Yn] y

If the USB Device is currently plugged in, please unplug and plug it back in for these changes to take effect.

USB Device successfully added.

```
# ./AddUsbRawPermissions.sh
```

NI-VISA : Add Permissions for a Raw USB Device

This Script gives permission to all users to access a specific raw USB device using NI-VISA. Under the default configuration of most Linux

distributions, only the root user can access all of the USB devices. Running this script is not necessary if NI-VISA will be run only by the 'root' user or if usbfs (formerly known as usbdevfs) is mounted with the "devmode=0666" option.

"udev" detected.

The Vendor ID (VID) of a USB device is a numeric identifier that may be up to 4 hexadecimal digits. It is found in the "device descriptor" of the USB Device. An example is "0xA1B2".

What is the Vendor ID (VID) of the USB Device? 0x1781

The Product ID (PID) of a USB device is a numeric identifier that may be up to 4 hexadecimal digits. It is found in the "device descriptor" of the USB Device. An example is "0xA1B2".

What is the Product ID (PID) of the USB Device? 0x0E93

The following USB Device information has been entered:

Vendor ID (VID) : "0x1781"  
Product ID (PID) : "0x0E93"

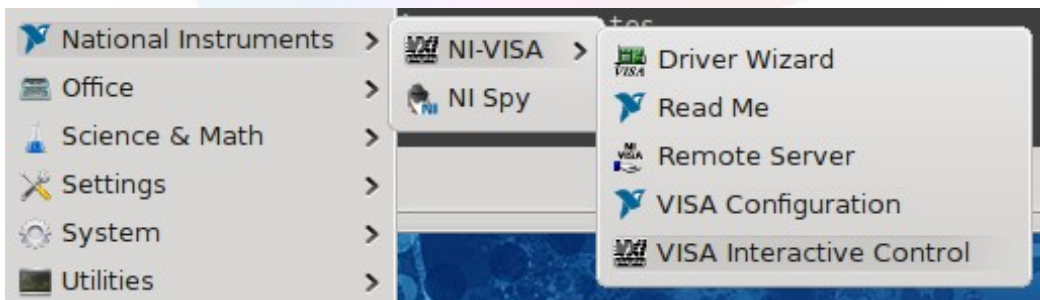
Continue? [Yn] y

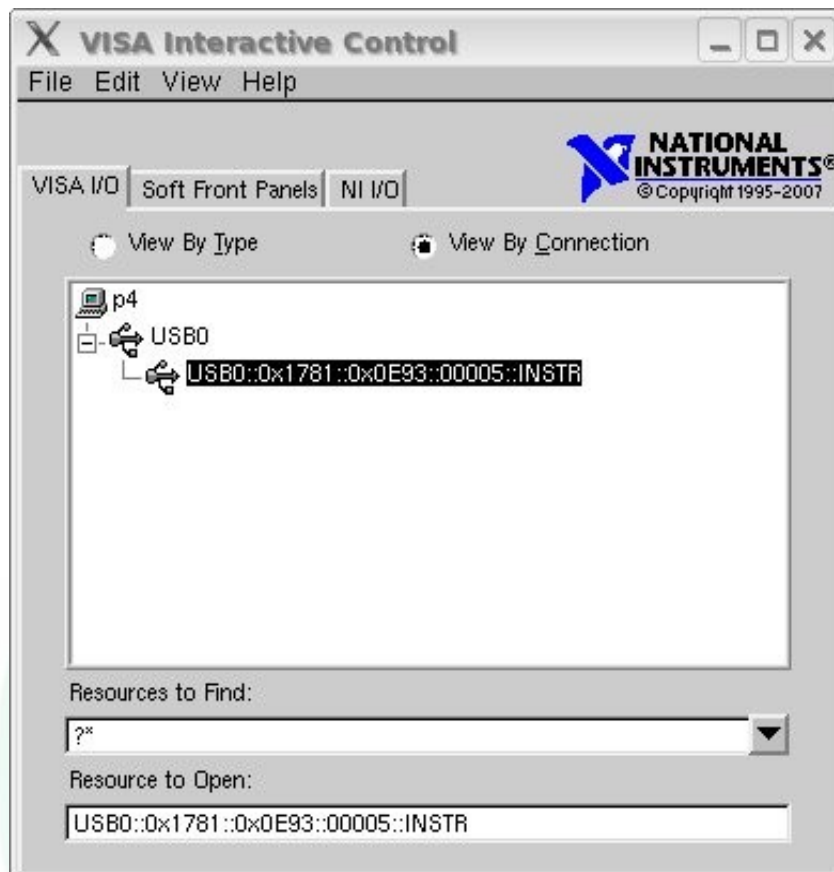
If the USB Device is currently plugged in, please unplug and plug it back in for these changes to take effect.

USB Device successfully added.

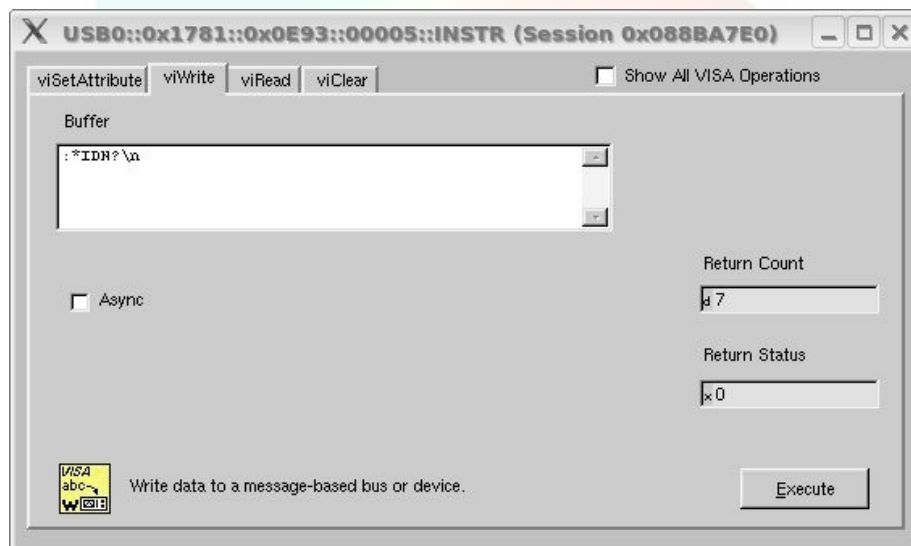
Restart the PC.

After this, the device should show up in the VISA interactive control dialog :





Double click the device to open a connection.  
Enter “:\*IDN?” to test if the Brontes works.



Go to the viRead tab to read back the information from the Brontes :



If this works than the Brontes is ready for use on your Linux machine.

## 4 Installing the Linux application.

The installation requires the Labview Runtime Engine to be installed on the target system since part of the application is made using Labview.

The Linux version of the colorimeter application comes is a tar.gz package. This contains only the application. The Labview runtime engine and NI-VISA and NIKAL are provided as separate packages.

### 4.1 Installing the Labview runtime engine.

Files are provided as RPM packages. To install them :

```
$ rpm -Uvh labview-rte-aal-1.1-1.i386.rpm
```

```
$ rpm -Uvh labview82-rte-8.2-1.i386.rpm
```

### 4.2 Installing the Colorimeter application.

The software is provided as a tar.gz package. Unpack it in your directory of choice as follows :

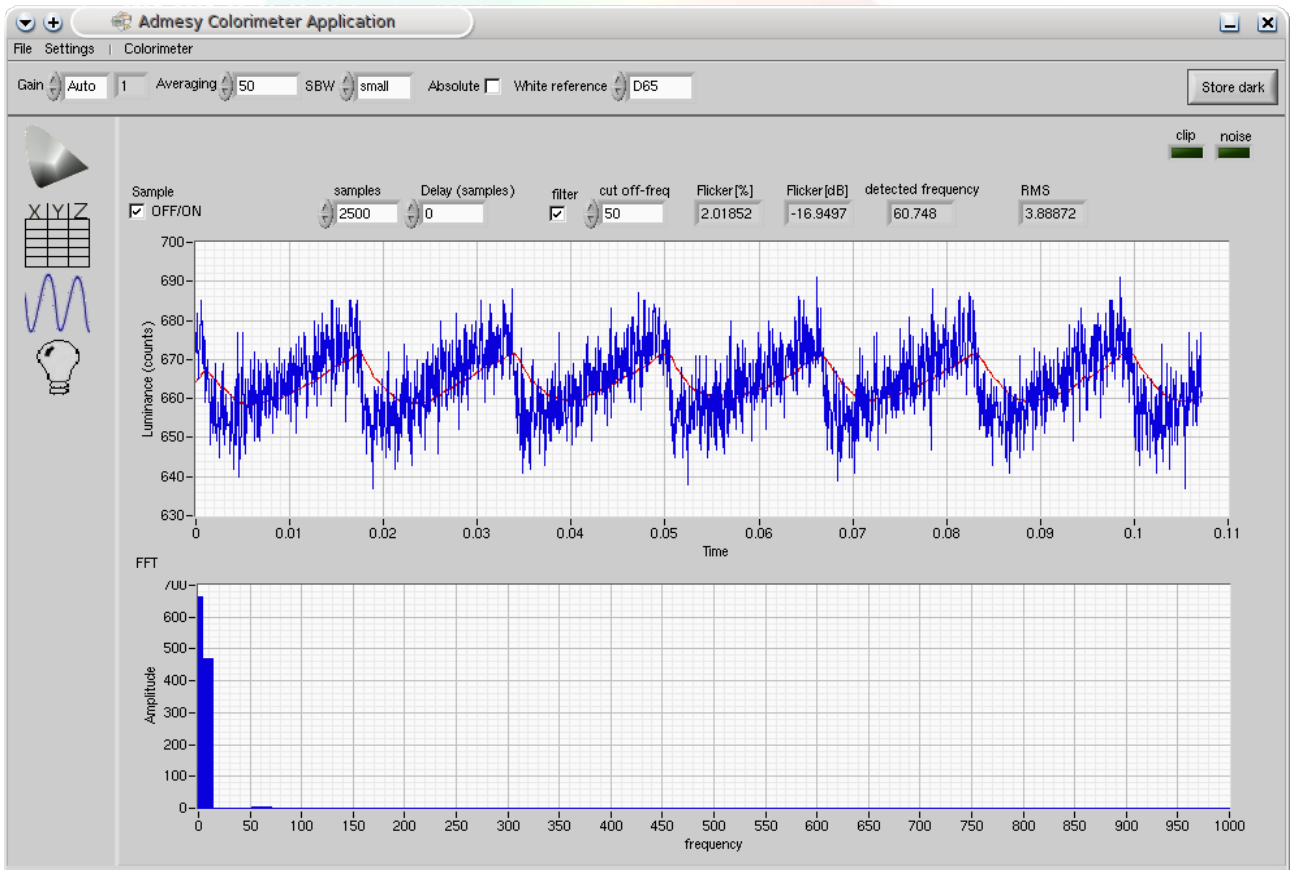
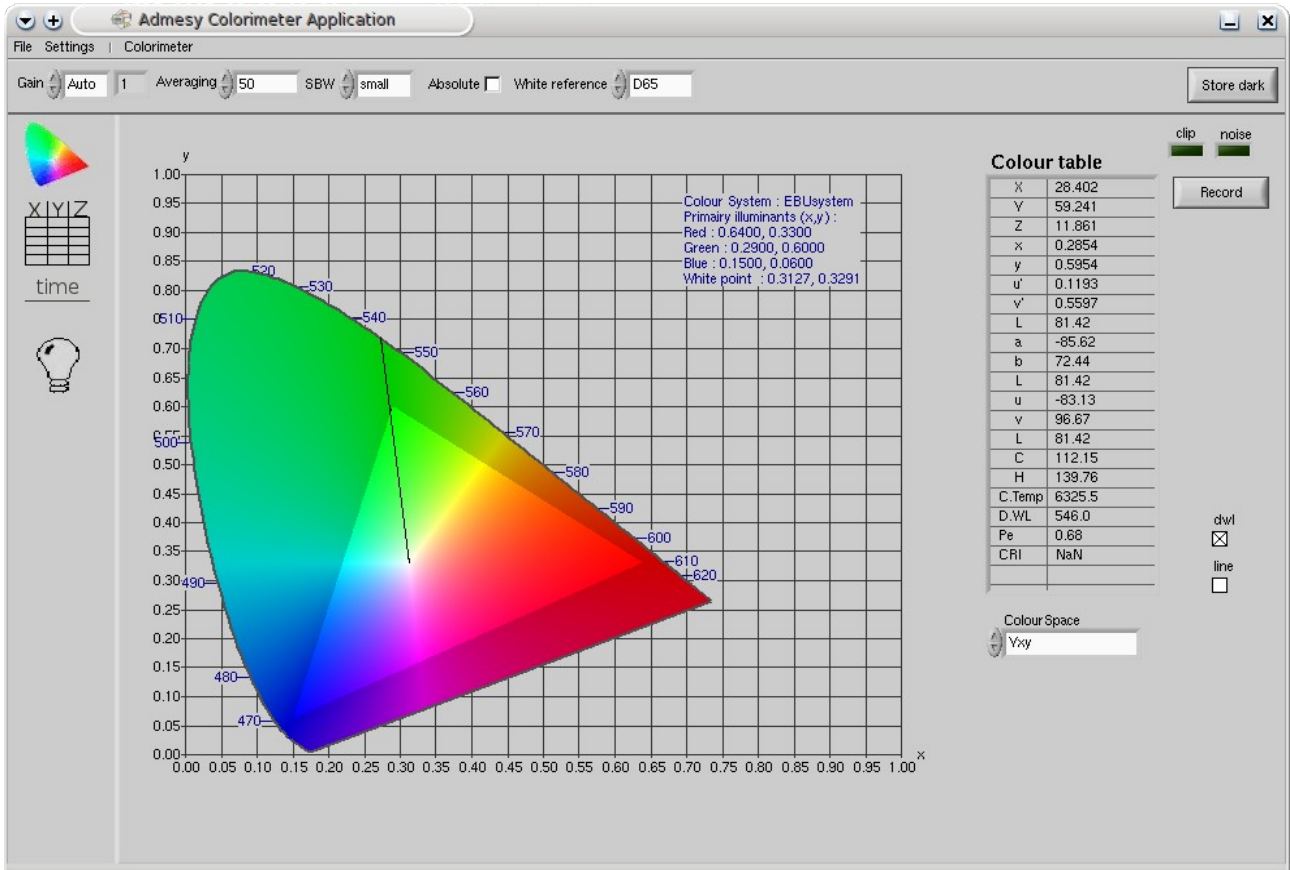
```
$ tar zxvf Admesy_Colorimeter_application_2.0.2_linux.tar.gz
```

To start the software :

```
$ cd Admesy_Colorimeter_application_2.0.2_linux
```

```
$ ./admesy_colorimeter
```

The next page shows some screenshots of the actual application.



## 5 Linux USBTMC kernel driver

As of kernel 2.6.28 an open source USBTMC driver is available. This is currently under test phase and will be Admesy's preferred driver once all testing has finished. We will at all times support NI-VISA for developers though.

## 6 Libusb

People who can't work with NI-VISA, The 2.6.28 or later kernels or the open source Agilent driver can use libusb, which is open source as well. It can be used on both Windows and Linux (and probably also OSX) and tests have shown no performance difference compared to the earlier mentioned drivers.

Explanation about libusb is out of scope for this document. Developers interested in this option should contact Admesy for more information.

## 7 Using the Agilent open source USBTMC driver.

Agilent provides a GPL licensed USBTMC driver for Linux which also works fine with the Brontes colorimeter. This driver has been tested successfully on a Fedora 8 x86\_64 system and is likely to produce similar results for embedded Linux applications that can act as a USB host (ARM platform for example).

**Please note that the colorimeter application requires NI-VISA and does not work with the following drivers.**

### 7.1 Installing and testing the Agilent USBTMC driver

Agilent provides very good documentation on their open source driver regarding compilation and usage. The driver compiled without issues on Fedora 8 and the provided scripts also worked out fine.

[http://www.home.agilent.com/upload/cmc\\_upload/All/usbtmc.html](http://www.home.agilent.com/upload/cmc_upload/All/usbtmc.html)

After compiling the driver, it can be loaded using the `usbtmc_load`. This will also make sure the device nodes exist.

The driver will show up all connected devices when calling the following command :

```
# cat > /dev/usbtmc0
Minor Number  Manufacturer  Product Serial Number
001  ADMESY  Brontes 00011
002  ADMESY  Brontes 00005
```

This shows two Brontes colorimeters attached to the host. Communication with the Brontes having serial number 00011 has to go via `/dev/usbtmc1` and the Brontes using serial number 00005 can be addressed via `/dev/usbtmc2`.

Testing the Brontes can be done via the command line in a very simple way :

```
# echo :*IDN?>/dev/usbtmc1
# cat /dev/usbtmc1
Admesy B.V. Brontes Colorimeter          <-- returned ID from Brontes
```

```
#echo :meas:xyz>/dev/usbtmc1
# cat /dev/usbtmc1
0.786,0.813,0.730,0,1                    <-- returned measurement value from Brontes
```

## 7.2 Writing a small C program to control the Brontes

In order to write a simple command line program that can measure colour using the Brontes, it is necessary to include the `usbtmc.h` header and some other system includes.

Writing code for the Brontes is as easy as sending the ASCII command strings to the brontes via the opened device file (`/dev/usbtmc1` in this case) and reading from the device will give the resulted measured or other information. Attached below is a simple Brontes test application that sets averaging and gain and measures XYZ.

Save the example code as “brontes\_test.c” and compile using :

```
gcc -Wall -O2 brontes_test.c -o brontes_test
```

When running the example, it should produce an output similar to this :

```
#!/brontes_test  
  
Number of instruments active: 2  
ID: Admesy B.V. Brontes Colorimeter  
MANUFACTURER: ADMESY  
PRODUCT: Brontes  
S/N: 00011  
Averaging set to 1000  
Gain set to 1  
Measure XYZ : 0.761,0.822,0.677,0,1
```

```

#include <unistd.h>
#include <stdio.h>
#include "usbtmc.h"
#include <fcntl.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>

int main()
{
int myfile;
char buffer[4000];
int actual;
int retval;
int en;
struct usbtmc_instrument inst;
struct usbtmc_attribute attr;

myfile=open("/dev/usbtmc1",O_RDWR); // Open instrument #1 for reading and
writing
if(myfile==-1) exit(1);

// Tell the driver we are using read(2), not fread(2)
attr.attribute=USBTMC_ATTRIB_READ_MODE;
attr.value=USBTMC_ATTRIB_VAL_READ;
ioctl(myfile,USBTMC_IOCTL_SET_ATTRIBUTE,&attr);

attr.attribute=USBTMC_ATTRIB_NUM_INSTRUMENTS;
ioctl(myfile,USBTMC_IOCTL_GET_ATTRIBUTE,&attr);
printf("Number of instruments active: %d\n",attr.value);

retval=write(myfile,"*IDN?\n",7); // Send *IDN? command
if(retval==-1)
{
en=errno;
printf("Error during write: %s\n",strerror(en));
}
actual=read(myfile,buffer,4000); // Read response
if(actual==-1)
{
en=errno;
printf("Error during read: %s\n",strerror(en));
printf("ID: %s\n",buffer);
}
else
{
buffer[actual]=0;
printf("ID: %s\n",buffer);
}

// Get info about instrument #1
inst.minor_number=1;
if(ioctl(myfile,USBTMC_IOCTL_INSTRUMENT_DATA,&inst)!=-1)
{
printf("MANUFACTURER: %s\n",inst.manufacturer);
printf("PRODUCT: %s\n",inst.product);
printf("S/N: %s\n",inst.serial_number);
}

```

```

retval=write(myfile,":sense:aver 1000\n",17); // Set averaging
if(retval==-1)
    {
        en=errno;
        printf("Error during write: %s\n",strerror(en));
    }
else printf("Averaging set to 1000\n");

retval=write(myfile,":sense:gain 1\n",14); // Set gain 1
if(retval==-1)
    {
        en=errno;
        printf("Error during write: %s\n",strerror(en));
    }
else printf("Gain set to 1\n");

retval=write(myfile,":meas:xyz\n",10); // Send *IDN? command
if(retval==-1)
    {
        en=errno;
        printf("Error during write: %s\n",strerror(en));
    }
actual=read(myfile,buffer,4000); // Read response
if(actual==-1)
    {
        en=errno;
        printf("Error during read: %s\n",strerror(en));
        printf("ID: %s\n",buffer);
    }
else
    {
        buffer[actual]=0;
        printf("Measure XYZ : %s\n",buffer);
    }

// Demonstrate timeout handling
actual=read(myfile,buffer,4000); // Provoke timeout
if(actual==-1)
    {
        en=errno;
        printf("Error during read: %s\n",strerror(en));
        printf("Recover from timeout using ABORT_BULK_IN\n");
        ioctl(myfile,USBTMC_IOCTL_ABORT_BULK_IN,0);
    }

close(myfile); // Done using the instrument

return 1;
}

```

### 7.3 Graphical scripting for the Brontes

On Linux it's possible to create software in a variety of software languages. Even shell scripts can make use of the usbtmc device. There are many tools available on Linux to create C/C++ software, but there are also tools that help less experienced programmers to create software. One of these tools is Kommander (<http://kommander.kdwebdev.org/>)

Below is a small example made in kommander which allows creating user interfaces in a very easy way and can use the device file with similar commands as shown earlier for the bash command shell.

