Miniforschung – Controlling the Goniometer at QCLAM with EPICS

Conducted from 26.09.2011 to 21.10.2011
by Henno Lauinger
Tutor: Jonny Birkhan, Simela Aslanidou

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

1 Aim of the project

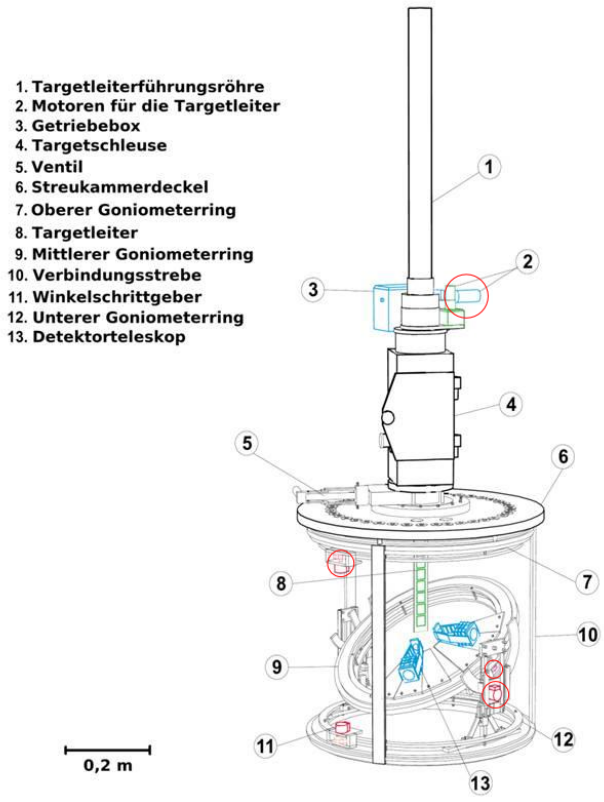
At QCLAM coincidence experiments are conducted, where electons and other particles are emitted. While the electrons are detected in a drift chamber in the focal plane, the other particles have to be detected in a full room angle of 4π . The mounting and motion of these detectors is provided by a goniometer.

Aim of the project is to be able to control the motors of the goniometer with EPICS. To be able to do so one must understand the basics of EPICS, build a device to test how to control a motor that is connected to a computer via a CAN bus and control it with EPICS.

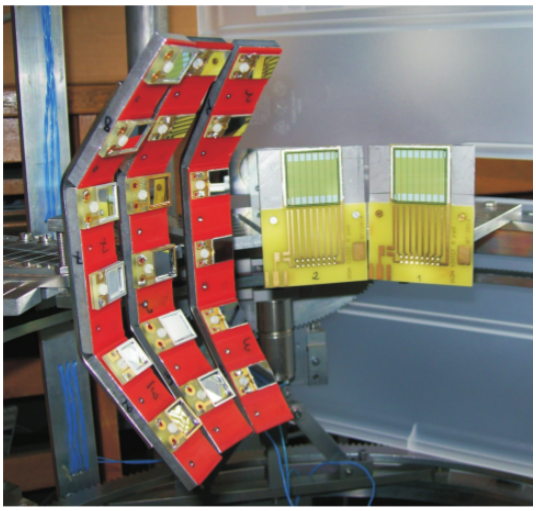
Furthermore the goniometer has to be rewired, so the connections in the top cover will be realised with new plugs.

This report should give an overview over what has been done so far and an introduction on how to work with EPICS.

1.1 Goniometer at QCLAM



(a) Sketch of the goniometer. The red circles mark the position of a motor.



(b) Detectors for additionally emitted particles in a partial room angle.

Figure 1: Sketch and image of the goniometer.

In the scattering chamber of QCLAM electrons are scattered on a target where protons, α-particles, deuterons and other charged particles can be seperated.

$$A(e, e'x)B \tag{1}$$

In this experiment the exact positions of the x-particles are meassured in a full room angle of 4π. Since the corresponding detectors only cover a partial room angle as shown in figure 1(b) they need to be rotated around the target to reach a 4π angle. This is achieved by three of the motors shown in figure 1(a), the fourth moves the target ladder into the scattering chamber. The trajectories and energies of the electrons are meassured in drift chambers in the focal plane of the spectrometer.

In the past years problems have piled up until it was decided to change some parts of the instrumentation, for example the connections of the motors and steppers. Additionally the motors are wanted to be controllable over the network with a graphical user interface, which will be achieved with EPICS.

2 Motor Testing Device

2.1 Construction

The testing device for a remotely controlled motor has been built from an old printer. A new motor (Faulhaber 1624 E 012 S) with built in stepper was attached and switches added near both ends of the track. A thin metal plate with slightly bent ends was put on the bottom of the slide so that it would slowly press the switch and be able to move off it.

A micro controller was programmed in the electronics workshop [2], where the additional objects were also put to the printer. The board is referenced as `motctl` in the header files `cps_can.h`.

The circuit board is supplied with 24 V, that are regulated to a maximum of 12 V via pulse width modulation (PWM), since the motor has a working voltage of 12 V. The motctl board has a temprature chip that can be easily used to test commands that should get information from the board.

The board is connected with an ethernet cable to a CAN bus, that can be connected to a computer with a USB cable. Communication between computer and micro controller is realised with CAN commands, explained in section 2.2.

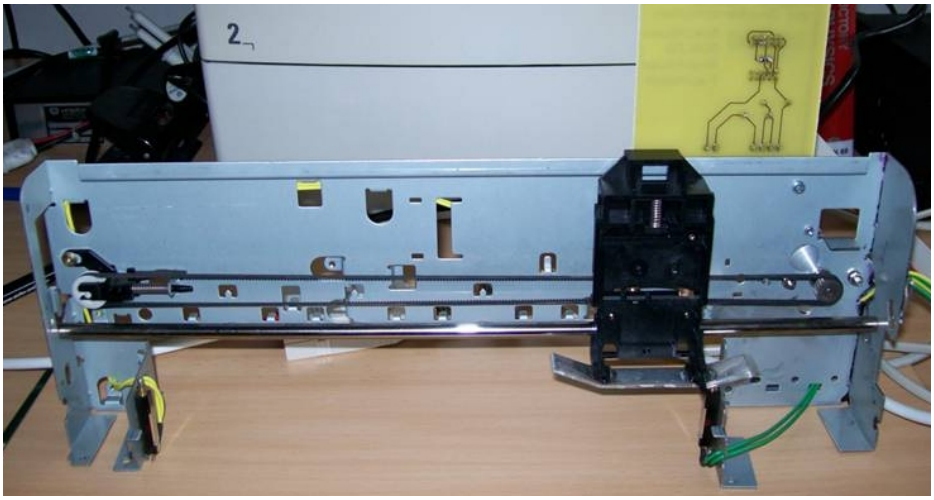
2.2 Local control: CAN Bus

Communication between computer and microcontroller is realised with CAN commands. Even EPICS will use these commands, but then the user doesn't have to take care of it. The drivers [3] that are needed for linux can be installed with the help of a tutorial [4], some additional command line utilities will be needed, too [5].

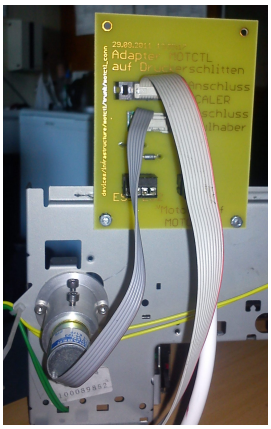
There may occur problems while installing the CAN drivers. If the file `popt.c` is missing you need to install the package `libpopt-dev` with the command `sudo apt-get install libpopt-dev`. To permanently set the bitrate of the adapter you have to add it to a config file, e.g.

```
sudo echo "options pcan bitrate=0x0014" >> /etc/modprobe.d/pcan.conf
```

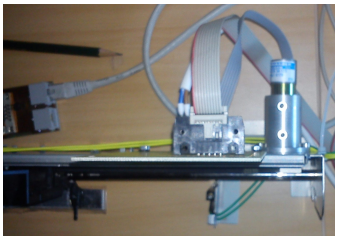
To download the CAN utilities you need to have the package `subversion` installed. The utilities have to be compiled with



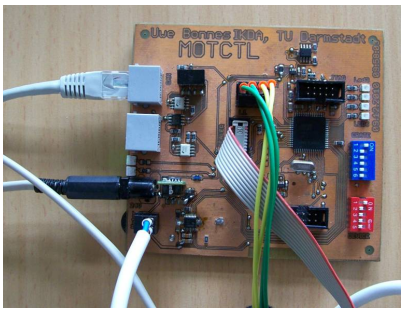
(a) The motor on the right moves a belt that is connected to a slide. [1]



(b) The motor is connected to an adapter circuit board that is connected to the mtctl board.



(c) The motor is screwed to the back of the printer.



(d) Mtctl board with microcontroller, temperature chip and RJ-45 sockets for connection to the CAN bus. [1]

Figure 2: Testing device built from parts of a printer.

```
cd <svn download folder>/trunk/can-utilities
make
sudo make install
```

All connected CAN adapters can be shown with `cat /proc/pcan`. They're called *can0*, *can1*, *can2* and so on. Everytime an adapter is connected to the computer it has to be activated with `sudo ifconfig can0 up` before commands can be sent. The red LED should blink now. Now you can show all messages of all CAN adapters with the command `candump any` from the CAN-utilities. A command can be sent with `cansend <can name> <command>`.

Every command consists of the address, the type of the command and optional data.

```
<address><type>[#<data>]
```

The address begins with `06` if a command will be sent and with `07` if an answer is expected. The following three numbers identify the device. They consist of the number of the power supply and the number of

the slot in the crate, where the device is put. Even if the device isn't inside of a crate, there are DIP switches on the circuit board, that give those numbers.

```
<#power supply/crate> * 0x40 + <#position in crate> * 0x02
```

The number of the crate or power supply is encoded with 6 bits, the number of the position with 5 bits. The command type is encoded with 13 bits, e.g. `00f` to move the motor to a certain position. All commands can be looked up in the header file of the micro controller. The hexadecimal numbers of the data bits are switched in pairs, i.e. they consist of `<LowByte><MiddleByte><HighByte>`, e.g. instead of `0x12 89 EF` you send `#EF8912`. To move the motor 100 steps if it's connected to power supply number 1 at position 1 you send

```
cansend can0 0604200f#640000
```

to move it 65792 steps if its position is 2 at power supply 1 it is

```
cansend can0 0604400f#000101
```

A full list of commands can be found in the CAN header file in

```
svn://130.83.133.64/acs/devices/firmware/nut/trunk/inc/cps_can.h
```

The commands that reffer to the test device are listed as *motctl*.

2.3 Remote Control: EPICS and CSS

All remote control is realised with EPICS. CSS is a GUI that connects with one or multiple IOCs and was used to build a visual interface that controlls the motor. In a later conversation it turned out that S-DALINAC and DESY are the only institutions that still use CSS, which also shows some amount of errors. For future developments one may consider to use the SNS control system *BOY* from Oak Ridge National Laboratory [6].

2.3.1 Experimental Physics and Industrial Control System – EPICS

The idea of EPICS is to be able to control your experiment remotely over the ethernet. An EPICS system consists of the database (*DB*) – the experiment with all controlling and messuring devices and a server that's connected to them and has an input/output controller (*IOC*) installed on it –, the channel access (*CA*) – the physical network and all network protocols – and the operator interfaces (*OPIs*) – computers with a user interface that can communicate with the IOC over CA.

Every computer that is part of the EPICS system or uses it in any way needs an EPICS base installed on it. You can download the source from the official webpage [7] or from the IKP svn server in `acs/frameworks/epics/base/trunk`. The EPICS base has to be compiled and installed as explained in section 4.2, but before that some system variables must be set:

```
export EPICS=<TOP>
export EPICS_HOST_ARCH=<host arch>
export EPICS_BASE=<TOP>/base
```

where `<TOP>` is the folder that contains epics and `<host arch>` can be `linux-x86` (32 bit Linux), `linux-x86_64` (64 bit Linux), etc. Since these variables have to be set every time something EPICS

related is compiled it is useful that they are set every time the computer is booted. In Ubuntu there's a file called `.bashrc` that is executed whenever a new terminal is opened, so also on logon. The user related file is located at `~/.bashrc` (`~/` is the home folder of the user, e.g. `/home/user`), the system wide file is located at `/etc/bash.bashrc` but you need root privileges to edit it.

2.3.1.1 Structure of an IOC

An IOC contains information about how to address which process variable (*PV*). A PV can be a physical variable, like a current or a voltage, or a calculated value, like a power. It is stored in a *record*, that contains all information that belongs to the PV, like its value (*VAL*), the engeneering unit (*EGU*), high and low operating ranges (*HOPR* and *LOPR*), a status (*STAT*), for example if it has already been set, if an alarm has been occured and so on. Each value of a distinct information is called a *field*.

A PV – and therefore a record – can have different types: It can be a calculated value (*calc*), an analog – decimal number – input (*ai*) or output (*ao*) or a digital – 'integer' – input (*longin*) or output (*longout*). *Long* refers to the memory size of the number, in this case four bytes. But every record needs a unique name since this is the way EPICS can tell them apart and how you address the record you want to get information from.

Each record has a field called *SCAN* that tells it when it should be processed. If its value is *Passive* it is only processed when the value of the record (the VAL field) is changed, for example by a user. A time – like 10 s or 0,2 s – means that the record is processed every ten or 0,2 seconds. *I/O Intr* tells it to be processed every time the hardware sends new information.

The information about how the record communicates with which hardware is set in the *OUT* field of an output record or the *INP* field of an input record. The syntax is

```
@<can interface> {06/07} <crate id> <slot id> <command> <skip> {s/u}<size> <timeout>
```

with the following variables

- `<can interface>` the name of the can interface the hardware is connected to, e.g. *can0* or *can1*
- `06` or `07` is the code for sending or receiving information as explained in section 2.2
- `<crate id>` and `<slot id>` are the number of the crate and the slot of the hardware like in section 2.2 but not calculated to the hexadecimal number
- `<command>` is the hexadecimal number of the command; section 2.3.2 shows that it is of better use to use the word placeholder of the command and replace it with a script later on
- `<skip>` is a decimal number of how many bytes should be skipped; this is important when reading for example only the fifth byte
- `{s/u}<size>` is the signed or unsigned size of the value that will be sent or read; `<size>` can be `0` if no information is sent or read, `c` for *character* (1 byte), `s` for *short* (2 byte), `m` for *mid* (3 byte, added since it is often needed) or `l` for *long* (4 byte)
- `<timeout>` in seconds tells the record after which amount of time it should set an error state if nothing has been sent or read

The first example in section 2.2 would look like this

```
OUT = "@can0 06 01 1 15 0 um 50"
VAL = "100"
```

If another record should be processed after processing a record, its name has to be put into the *FLINK* field. This can be used to calculate the power after current and voltage have been read. To re-

fer to the value of a field in another record one can use <record name>.<field name>, for example motctl1:getPosition.VAL for the position of motor 1 on the motctl board.

A full list of record types and their fields can be found in [8]. More examples on how to create records, especially for the controll of the test device can be found in section 2.3.3.1.

2.3.1.2 Channel Access and IOC Commands

As mentioned before, the built-in tools to access the database in EPICS are called channel access. They can be found in <epics base>/bin/<host arch>/. The most important ones are

```
caget <record name>[.<field name>]
caput <record name> <value>
camonitor <record name>
```

caget prints the value of a field of a record, default is VAL, caput updates the VAL field of a record to a new value and camonitor prints information about a record every time something is changed. These tools can be used from every workstation that has an EPICS base installed on it and is connected to the server with an IOC over the network, or from the server itself.

The IOC binary (located at <IOC base>/bin/<host arch>/<IOC name>) has a built-in command line, too, which can of course only be used on the server. Some useful commands are

- help [<command>] prints all commands or information and syntax for one command
- dbl lists all record names in the database
- dbpr <record name> [<level>] shows some fields of a record, dependant on the information level
- dbpf <record name>[.<field name>] <value> changes the value of a field of a record, default field is VAL
- dbLoadDatabase <.dbd file> loads a database definition file
- dbLoadTemplate <.substitutions file> loads a database that is created by replacing variables in template files with information from the substitution file

The last two commands are usually executed by a script in <IOC base>/iocBoot/<host arch>/st.cmd. More information about the template and the substitution file can be found in section 2.3.3.

2.3.2 Building an Input/Output Controller

If you want to build an IOC, be able to list and test the built-in commands or to learn how to work with CSS, the quickest way is to compile the example server in the EPICS base package. The report of Dirk Martin [9] gives a quick tutorial.

If you allready have a configured IOC you can compile it with

```
cd <IOC base>/trunk
make clean
make
```

and run it with

```
cd <IOC base>/trunk/iocBoot/<host arch>
./st.cmd
```

If you want to create an IOC for a new experiment the easiest way is to copy and customise an existing one. The IOCs of the IKP can be found at

```
svn://130.83.133.64/acs/frameworks/epics/iocApps
```

You need an account to access the files on the svn server. Since you have to compile a *tudSocketCan* driver and you need the header files of the micro controllers it is recommended to download all files with

```
cd <svn base folder>
svn co svn://130.83.133.64/acs
```

The files can be updated with `svn up`. The *tudSocketCan* driver can be compiled with

```
cd <svn base folder>/acs/frameworks/epics/support/socketCan/trunk
make clean
make
sudo make install
```

2.3.3 Configuring an IOC

The best way to understand how to configure an IOC is to try to understand an existing one. Basically an IOC is configured with `.dbd` and `.db` files and with the start script `st.cmd`.

The `.dbd` (*d*ata*b*ase *d*efinition) files define the fields of the different record types. It's usually sufficient to use an existing one. The `.db` files define all records of PVs that will be contained in the IOC. The start script can be found at `<IOC base>/trunk/iocBoot/<host arch>`. Its main purpose is to start the IOC and load the database definition, the database files and the SocketCAN driver.

If there are a lot of PVs in your database you might want to generate them from templates. The templates contain variables for names, addresses, limits and values, that are replaced by the names in a substitution file located in `<IOC base>/trunk/<name>App/Db`. The substitution is executed in the start script. Variables in the templates located in `<svn base>/acs/frameworks/epics/iocApps/templates/trunk` look like `$(name)` or `$(can_interface)`, the structure of the substitution file looks like

```
file "db/motctl.template" {
    {
        name = "motctl1"
        can_interface = "can0"
        crate_id = "01"
        slot_id = "1"
        scan_fast_single_device = "1 second"
        scan_slow_single_device = "10 second"
    }
    {
        name = "motctl2"
        can_interface = "can0"
        crate_id = "01"
        slot_id = "2"
```

```
        scan_fast_single_device = "1 second"
        scan_slow_single_device = "10 second"
    }
}
```

This generates all records from `db/motctl.template` twice, once for each device on the CAN adapter.

2.3.3.1 Editing the Database with the Visual Database Configuration Tool

The best possibility to edit the template files is with the java editor *Visual Database Configuration Tool* (*vdct*). It can be downloaded from the web [10] or from the IKP package sources:

```
echo "deb http://debianfai.ikp.physik.tu-darmstadt.de/epics.nsls2.bnl.gov/
squeeze main contrib" >> /etc/apt/sources.list
echo "deb-src http://debianfai.ikp.physik.tu-darmstadt.de/epics.nsls2.bnl.gov/
squeeze main contrib" >> /etc/apt/sources.list
sudo apt-get update
sudo apt-get install nsls2-archive-keyring
sudo apt-get update
sudo apt-get install visualdct
```

The editor can be run with `vdct` or more convenient with a `.dbd` file allready loaded, e.g:

```
java -cp /usr/share/visualdct/VisualDCT.jar
  -DEPICS_DB_INCLUDE_PATH=$EPICS_DB_INCLUDE_PATH com.cosylab.vdct.VisualDCT
  --dbd-file <IOC base>/trunk/dbd/<name>.dbd
```

Vdct shows a graphical Layout of all records of a template, their fields and the connections between them. If one record refers to the value of another one or if there is a record name in the *FLINK* field, there will be lines connecting the records. Only fields that differ from the default value will be shown in the overview. By double clicking a record all possible fields are shown, including a short description when selected.

This provides a clearer possibility to change a database than by editing the text file.

Figure 3 shows an example of connected records in vdct, in this case setting the position of the test motor in `$(name):setPosition` of type *longout*, telling it to process `$(name):calcSlideTarget` of type *calc* afterwards – by putting this name into the *FLINK* field – which then converts the target position from steps to centimeters. The field *CALC* contains a formular with `A` and `B` which are taken from *INPA* and *INPB*. *INPA* reads the position in steps from the other record, *INPB* contains the slope set in the substitution file.

Note that the example shows a `.template.pre` file, which is usually edited, and thus contains variables like `$(name)` and `$(slope)`, which are set in the substitutions file. When the IOC is compiled with `make` the pre-template file is copied to `<IOC base>/db/<template-name>.template` and the command placeholder (in this case `CMD_SET_DAC`) is replaced with the controlbyte (in this case `15`). This is done automatically by a script. When the template is loaded into the IOC all variables are replaced with the values defined in the substitution file, which finally makes the file understandable for EPICS.

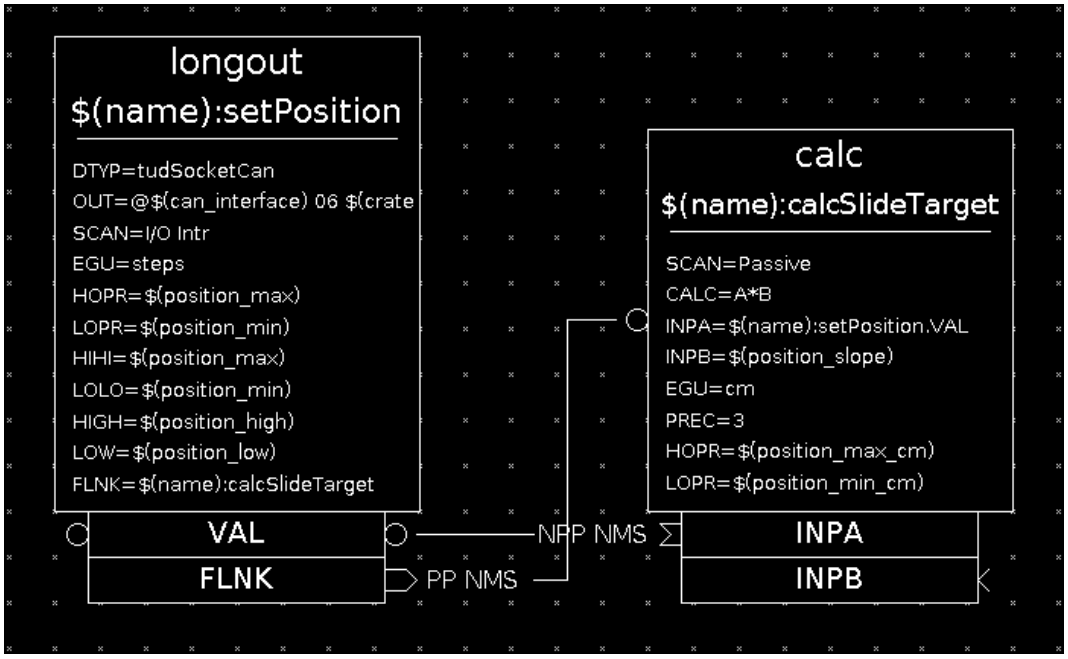


Figure 3: Two connected records in VDCT that set a position of the slide in steps and convert it to centimeters. In the left record – $(name):setPosition – the field *OUT* reads *@$(can_interface) 06 $(crate_id) $(slot_id) CMD_SET_DAC 0 um 0*.

2.3.4 Creating a New IOC

There are some binary files delivered with EPICS base, that can help create a new IOC. Some scripts have to be copied and some files changed to your needs.

2.3.4.1 Create Basic Folders and Files

First of all the directories for the new IOC have to be created – in a file browser or with

```
mkdir <svn base>/acs/frameworks/epics/iocapps/<IOC name>
mkdir <svn base>/acs/frameworks/epics/iocapps/<IOC name>/branches
mkdir <svn base>/acs/frameworks/epics/iocapps/<IOC name>/tags
mkdir <svn base>/acs/frameworks/epics/iocapps/<IOC name>/trunk
```

Then the source code for the IOC can be created with

```
makeBaseApp -t ioc <IOC name>
```

executed from `<IOC base>`. Afterwards the `iocBoot` folder can be created with

```
makeBaseApp -i -t ioc -a <host arch> -p <IOC name> <host arch>
```

2.3.4.2 Add CAN driver support

To add the socket-CAN driver support, `<IOC base>/configure/RELEASE` has to be edited (information about the syntax used can be found in section 4.1.1):

```
 # If using the sequencer, point SNCSEQ at its top directory:
 #SNCSEQ=$(EPICS_BASE)/../modules/soft/seq

+SOCKETCAN=$(TOP)/../../../support/socketCan/trunk
+CAN_INCLUDE_DIR=$(TOP)/../../../../../devices/firmware/nut/trunk/inc
+
 # EPICS_BASE usually appears last so other apps can override stuff:
 EPICS_BASE=/usr/lib/epics
```

The two folders have to exist on the computer that compiles the IOC. There are also changes in <IOC base>/<IOC name>App/src/Makefile:

```
 # motctlTest.dbd will be made up from these files:
 motctlTest_DBD += base.dbd
+motctlTest_DBD += tudSocketCan.dbd

 # Include dbd files from all support applications:
 #motctlTest_DBD += xxx.dbd

 # Add all the support libraries needed by this IOC
 #motctlTest_LIBS += xxx
+motctlTest_LIBS += tudSocketCan

 # motctlTest_registerRecordDeviceDriver.cpp derives from motctlTest.dbd
 motctlTest_SRCS += motctlTest_registerRecordDeviceDriver.cpp
```

and in <IOC base>/iocBoot/<host arch>/st.cmd:

```
 ## Load record instances
 dbLoadRecords("../../db/<IOC name>.db","user=<user name>")

+devSocketCanInit
+
 iocInit()

 ## Start any sequence programs
```

2.3.4.3 Automatically Add Template

make has to know where the pre-template is located and where it has to be copied. Add the template folder to <IOC base>/configure/RELEASE:

```
 SOCKETCAN=$(TOP)/../../../support/socketCan/trunk
 CAN_INCLUDE_DIR=$(TOP)/../../../../../devices/firmware/nut/trunk/inc

+DBTEMPLATES=$(TOP)/../../templates/trunk
+
 # EPICS_BASE usually appears last so other apps can override stuff:
 EPICS_BASE=/usr/lib/epics
```

and the name of the templates to <IOC base>/<IOC name>App/Db/Makefile:

```
 # Create and install (or just install) into <top>/db
 # databases, templates, substitutions like this
 #DB += xxx.db
+DB += <template name>.template

 #----------------------------------------------------
 # If <anyname>.db template is not named <anyname>*.template add
```

In the same file the rule for the replacement of the command placeholders has to be added:

```
 #----------------------------------------
 #  ADD RULES AFTER THIS LINE

+../O.Common/<template name>.template: $(DBTEMPLATES)/<template name>.template.pre
+ cd ../O.Common && $(TOP)/<IOC name>App/Db/replaceCmds.pl -I
$(CAN_INCLUDE_DIR)/cps_can.h $< $@
```

Of course the script, that does the replacement has to be copied to the given location (or the location has to be changed in the rule above).

```
cp /frameworks/epics/iocapps/gun/trunk/gunApp/Db
 '<IOC base>/<IOC name>App/Db/replaceCmds.pl'
```

The IOC gun is just one location of the script. Every other IOC should have it, too.

2.3.4.4 Adding Substitutions

Place the <IOC name>.substitutions file in <IOC base>/<IOC name>App/Db/ and tell make that it should copy it when it compiles. Add to <IOC base>/<IOC name>App/Db/Makefile

```
 # databases, templates, substitutions like this
 #DB += xxx.db
 DB += <template name>.template
+DB += <IOC name>.substitutions

 #-------------------------------------------------------
 # If <anyname>.db template is not named <anyname>*.template add
```

Finally change `st.cmd` to tell the IOC to load your template with the substitutions:

```
 ## You may have to change motctlTest to something else
 ## everywhere it appears in this file

-#< envPaths
+< envPaths

+cd ${TOP}
+
 ## Register all support components
-dbLoadDatabase("../../dbd/<IOC name>.dbd",0,0)
+dbLoadDatabase("dbd/<IOC name>.dbd",0,0)
 <IOC name>_registerRecordDeviceDriver(pdbbase)

 ## Load record instances
-dbLoadRecords("../../db/<IOC name>.db","user=<user name>")
+dbLoadTemplate "db/<IOC name>.substitutions"

 devSocketCanInit
```

If there are multiple substitution files the last line has to be added for every one of them and the `Makefile` needs to know all their names (step one in this section).

2.3.4.5 Set File flags

If you are working on an svn server you might want to ignore all files generated automatically during compiling with `svn propset svn:ignore <file>`. Also `<IOC base>/iocBoot/<host arch>/st.cmd` and `<IOC base>/<IOC name>App/Db/replaceCmds.pl` have to be executable:

```
svn propset svn:executable <IOC base>/iocBoot/<host arch>/st.cmd
svn propset svn:executable <IOC base>/<IOC name>App/Db/replaceCmds.pl
```

2.3.4.6 Compiling

Now the IOC is ready to be compiled and started:

```
cd <IOC base>
make
<IOC base>/iocBoot/<host arch>/st.cmd
```

Compiling will copy the pre-template, replace the command placeholders and copy the substitution files to the IOC working directory. `st.cmd` will then start the IOC, load the paths to the `.dbd` file, templates and substitutions and will load them into the IOC, that will then show its command line and start working.

2.3.5 Controll System Studio

Controll System Studio (CSS) is an IDE to create graphical user interfaces (GUI) that can interact with an EPICS database.

First of all the IP address of all IOCs has to be set in *CSS → Preferences... → General → Network Connections → Add Host...*. Then the connection can be tested with a *probe*. A probe is a meter that shows the value of a PV and refreshes every time its value changes. It can be added via *CSS → Diagnostic Tools → Probe* and just needs the full name of the PV to operate.

A complete GUI can be created with *File → New → Other... → Display → Synoptic Display*. It should appear as a new tab with the name *<display name>.css-sds* in the last active part of the window. A list witch objects should be on its right side, if it's not it cann be added with the arrow button in the upper right corner or via *Window → Show View → Other... → General → Palette*. If the *Widget Properties* won't show up, usually below the editor, they can be added via *Window → Show View → Other... → Synoptic Display Studio → Widget Properties*. The whole layout can be rearanged, for example by changing to the *Display Development* perspective (*Window → Open Perspective... → Other... → Display Development*).

Somewhere in the middle of the window should be a grey space, the editor. Objects or *widgets* can be added by selecting it from the palette and drawing a frame on the editor where and how big it should be. A window will pop up and prompt for a PV name and a *behaviour*. The PV name is the full name of a PV on an IOC, the behaviour can be chosen from a drop down list (I couldn't find any differences in the two options). Some tweaks can be made in the widget properties but most of the times it should already work.

The display can be started by saving the file and clicking the green *Display in Run Mode* button.

More information can be found in [9], [11] or in CSS via *Help → Welcome → First Steps*.

2.3.6 Important Folders when Working with EPICS on the Svn Server

Section 4.3.2 shows gathered information about important folders and files you will need on the svn server. Except for the creation of a brand new IOC only very few files are edited:

- `<...>/iocApps/templates/<template name>.template.pre` is the pre-template file that contains all information about records on the IOC. There can be multiple pre-templates, also located in subfolders.
- `<...>/iocApps/<IOC name>/trunk/<IOC name>App/Db/<substitutions name>.substitutions` contains the substitutions that can be edited.
- `<GUI name>.css-sds` is the graphical user interface that can be located anywhere on a computer.

All other files and folders are either automatically updated by scripts or contain permanent information.

For the use of an IOC it is important to know what the folders in `<...>/iocApps/<IOC name>/trunk/` (from here on called `<IOC base>`) contain. This is shown in figure 4. Folders that begin with `0.` are

created during the compilation of the IOC. If something would be edited ther it will be lost the next time the IOC is compiled. All `Makefile` and `RULES` files are created when the IOC is created and control the compiling.

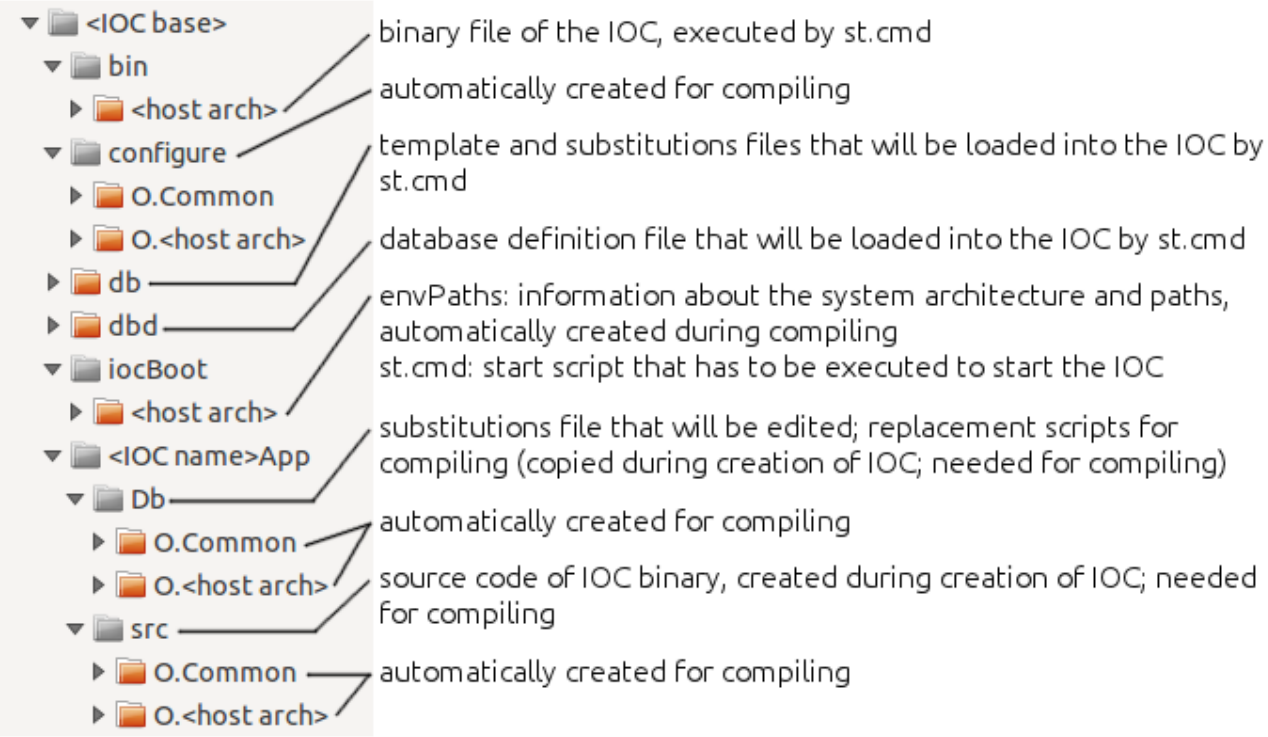


Figure 4: Folder structure of an IOC

Most information about EPICS has been taken from [9, 12, 13], information about the hardware and communication via a CAN bus are from [2].

2.4 Issues

There are still some issues with the test device. When the motor gets the command to stop it has still got some momentum. Usually this is no problem, because when it sopped it will move slowly backwards until it reaches the target position within a given range. But if it is sent near an end switch the slide might press the switch. It then moves slowly in the positive direction, regardless of which switch it pressed, and when the switch is released the current position is defined as the new zero. This causes the slide to get stuck if it reaches the switch on the far end. Furthermore the switches might get pressed – by accident or on purpose – even if the slide is in the middle of the track. This also defines a new zero position and the device has to be rebooted to find the real zero position.

It would also be a nice feature to be able to stop the motor in case something goes wrong.

3 New Connections in the Top Plate

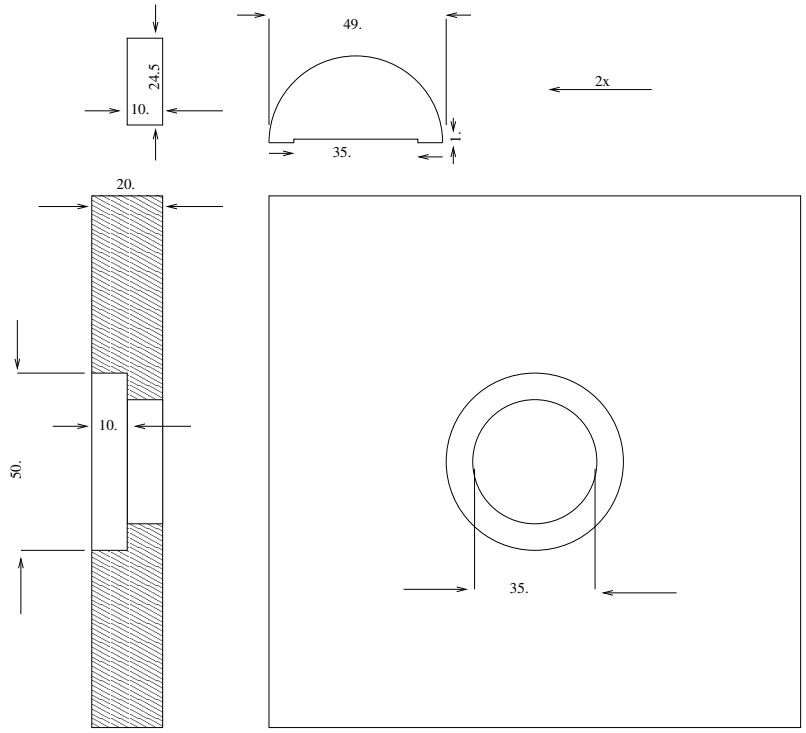
Since the soldered connections to the motors easily break when the top plate is taken off there has been a draft for new connections. The idea is to glue the cricuit boards directly into the top plate and attatch sockets to it. The cables can be plugged in and out when removing the plate.

To test if the adhesive would be able to endure the vacuum inside the goniometer a small replica of the top plate was built. It contains just one hole from the new draft, where two half cylinders were put inside and fixed with the adhesive. The adhesive was put on the circuit board which was slid into the leftover slit. It was attached to a frame with clamps to give the adhesive enough time to harden and to be able to seal holes in the layer of adhesive.

When the adhesive hardened a vacuum pump was pressed onto the plate and started to see if the adhesive would rip.

Apart from some problems with the seal of the vacuum pump the test went well. The adhesive could hold the pressure of the vacuum pump so one can assume that it will be able to endure the vacuum in the goniometer.

As a result of the test it was decided to first weld the half cylinders into the top plate and then glue the circuit boards that already contain one half of the electronics on it into the slits. The other electronics will be soldered onto the boards afterwards. It is necessary to put a thic layer of adhesive on the circuit board that there won't be created any holes when the circuit board is wobbled. It's also important to pay attention that the conducters on the board don't touch the top plate because this would cause a short circuit later on.



(a) Blueprint for the mechanical workshop made with *xfig*.

(b) Milled hole in top plate (left) and glued circuit board inside it (right).

Figure 5: New fittings for the connections in the top plate.

List of Figures

Bibliography

[1] Personal communication with Jonny Birkhan.

[2] Personal communication with Uwe Bonnes.

[3] http://www.peak-system.com/fileadmin/media/linux/index.htm, 23.11.2011.

[4] http://2codeornot2code.org/?p=400, 23.11.2011.

[5] http://developer.berlios.de/svn/?group_id=6475, 23.11.2011.

[6] http://sourceforge.net/apps/trac/cs-studio/wiki/BOY, 23.11.2011.

[7] http://www.aps.anl.gov/epics/base/index.php, 23.11.2011.

[8] http://www.aps.anl.gov/epics/wiki/index.php/RRM_3-14, 23.11.2011.

[9] D. Martin, Test des Goniometers und seiner Anbindung an EPICS für Elektronenstreuexperimente am QCLAM-Spektrometer, Miniforschung, IKP, 2010.

[10] http://www.cosylab.com/resources/downloads_/, 23.11.2011.

[11] http://ikpweb.ikp.physik.tu-darmstadt.de/mediawiki/index.php/CSS, 23.11.2011.

[12] Personal communication with Martin Konrad.

[13] Personal communication with Christoph Burandt.

[14] http://css.desy.de, 23.11.2011.

[15] Personal communication with Simela Aslanidou.

[16] http://subversion.apache.org/, 23.11.2011.

[17] http://www.ccp2.ac.uk/svn_workflow.pdf, 23.11.2011.

4 Attachments

4.1 Syntax Used in this Document

- `monosized typeset` depicts folders, filenames or commands
- < > sharp brackets contain short explanations for a placeholder, such as the name of a variable or the path to a folder that can change from one user to another
- [ ] square brackets contain optional arguments
- { } pointed brackets contain different choices for a nonoptional argument unless they are surrounded by square brackets
- → shows the separation between submenus or items in preference windows

4.1.1 Diff Syntax

`diff` is a tool used by svn to find and display changes between two files. The logfiles contain all these changes and look something like this:

```
Index: trunk/configure/RELEASE
===================================================================
--- trunk/configure/RELEASE (Revision 4435)
+++ trunk/configure/RELEASE (Revision 4436)
@@ -24,6 +24,9 @@
 # If using the sequencer, point SNCSEQ at its top directory:
 #SNCSEQ=$(EPICS_BASE)/../modules/soft/seq

+SOCKETCAN=$(TOP)/../../../support/socketCan/trunk
+CAN_INCLUDE_DIR=$(TOP)/../../../../../devices/firmware/nut/trunk/inc
+
 # EPICS_BASE usually appears last so other apps can override stuff:
 EPICS_BASE=/usr/lib/epics

Index: trunk/motctlTestApp/src/Makefile
===================================================================
--- trunk/motctlTestApp/src/Makefile (Revision 4435)
+++ trunk/motctlTestApp/src/Makefile (Revision 4436)
@@ -14,12 +14,14 @@

 # motctlTest.dbd will be made up from these files:
 motctlTest_DBD += base.dbd
+motctlTest_DBD += tudSocketCan.dbd

 # Include dbd files from all support applications:
 #motctlTest_DBD += xxx.dbd
```

```
 # Add all the support libraries needed by this IOC
 #motctlTest_LIBS += xxx
+motctlTest_LIBS += tudSocketCan

 # motctlTest_registerRecordDeviceDriver.cpp derives from motctlTest.dbd
 motctlTest_SRCS += motctlTest_registerRecordDeviceDriver.cpp
```

This example shows two changed files: `trunk/configure/RELEASE` and `trunk/motctlTestApp/src/Makefile`. The header of each file contains its location, the old and the new revision and information about the number of lines of these files. Then a piece of the file follows where changes have been made. The first character of each line indicates what has been changed:

- (blank or space) indicates no changes in this line. This can be used to locate the surrounding lines of the file.
- + shows that the line has been added to the new version.
- - means that the line has been removed.

4.2 Compiling from Source Code in Linux

A lot of software is only distributed as source code and you have to compile the binary files for your system structure (Linux/Windows/Mac, 32 bit/64 bit) yourself. In Linux you usually use `make` to compile the source code. In a command line you first change to the folder with the source, then run the compiler and tell it to install the software (i.e. copy the binaries and configuration files to the system directories):

```
cd <source folder>
make
sudo make install
```

If you changed something in the source code and want to make sure that the changes are applied and no old files will be kept delete them with

```
make clean
```

If `make` doesn't work you can also use a different compiler, for example `cmake`.

These compilers are configured with files called `Makefile`. They can contain rules about copying and renaming files, which is used to build an IOC.

4.3 Subversion and the IKP server

Subversion (*svn*) is a version control system designed to make merging of projects with multiple contributors easy [16]. Usually an institute has an svn server where all projects are centrally stored. Whenever a contributor wants to change something on a project he downloads the latest files from the server, makes his changes, compares his new file with the version on the server in case someone else made changes at the same time, optionally updates these changes into his file and uploads it to the server.

In Ubuntu it can be installed from the software-center or from the command line with

```
sudo apt-get install subversion
```

4.3.1 Workflow

If you are new to a project you have to get your copy of the files, this is called checking out

```
cd <svn base folder>
svn co --username <name> <address>
```

The address can be in the local network or on the internet. All files will be coppied to the current working directory, in this case `<svn base folder>`. To get all files from the IKP server use

```
svn co svn://130.83.133.64/acs
```

when you are connected to the IKP network. You may need an account first. You can also check out for example just epics related folders

```
svn co svn://130.83.133.64/acs/frameworks/epics
```

If the username is ommitted the system username will be taken instead. If this user can't be found on the server you will be asked to enter a new username.

If you want to check if your copies are still the same as the files on the server you can use

```
svn status [-u]
```

to show all files that have been changed. the `-u` option will show revision and server out-of-date information.

To actually show the differences between the files use

```
svn diff
```

There is also software with graphical user interfaces that use `diff` like *Meld* on Ubuntu or *WinMerge* on Windows.

To download any changes on the server to your local machine use

```
svn update
```

With most commands you can add the `-r <revision number>` option to refer to a certain revision of the project.

When you want to upload your changes to the server you have to tell it first if you added or deleted files, then commit the changes

```
svn add <filename>
svn delete <filename>
svn commit
```

When committing changes you will be prompted for information about the changes and should describe it for other contributers to understand these changes.

4.3.2 Svn Folder Structure

Important main folders and files:

- `acs/devices/` contains files that have something to do with hardware

  `./firmware/nut/trunk/inc/cps_can.h` is the header file of hardware made by Mr. Bonnes; it contains the word placeholders for commands that control a device
- `acs/frameworks/` contains EPICS related files and user interfaces made with CSS-SDS

  `./epics/base/` contains source and binaries of the EPICS base

  `./epics/iocapps/` contains all IOCs that are used at the IKP

  `./templates/` contains all template files for the IOCs

Each project folder contains three folders:

- `branches/` contains released versions of a project. Only bugfixes should be applied there but no new development.
- `tag/` can contain tags for the project.
- `trunk/` is the folder with the development version of the project. Most of the work will take place there.

More detailed information about svn can be found in [17] or with the command `svn help [<subcommand>]` in a command line.