Bachelor Thesis

Development of Components of the Detector Control System for the Prototype of the Second Station of the Micro Vertex Detector of the Compressed Baryonic Matter Experiment

Ole J. Artz
aus Frankfurt am Main

Institut für Kernphysik
Johann Wolfgang Goethe-Universität Frankfurt am Main

Erstprüfer: Prof. Dr. Joachim Stroth
Zweitprüfer: Dr. Jan Michel

July 25, 2018 (v2)

Abstract

This thesis describes the development of components of the detector control system (DCS) for the prototype of the second station (PRESTO) of the Micro Vertex Detector (MVD) of the Compressed Baryonic Matter Experiment (CBM). The upcoming CBM experiment at the Facility for Antiproton and Ion Research (FAIR) in Europe GmbH will study strongly interacting matter in heavy-ion collisions. For the identification of rare particles by the spacial displacement of their decay vertices, the MVD of the CBM experiment was designed. Also, the MVD is an integral part of the CBM tracking system. The prototype called PRESTO was conceived to study multiple aspects such as integrating sensors on two sides of a carrier. Furthermore, a continuous and autonomous operation is to be tested with PRESTO. To accomplish this, a control system has to be implemented.

Within the framework of this thesis, the implementation of the essential components of the detector control system (DCS) for PRESTO was achieved within the Experimental Physics and Industrial Control System (EPICS) framwork. All PRESTO relevant devices were added to the EPICS control system with the help of so called Input/Output Controllers (IOCs). Graphical user interfaces (GUIs) like Control System Studio (CS-Studio), RDB Archiver and, in addition, a Web Dashboard as an mobile GUI. With some additional contributions not covered in this thesis, such as full alarm handling this detector control system will be ready to operate PRESTO safely with little human interaction. A short discussion quickly flashes those additionally required components.

This thesis is structured as follows. The first part introduces the CBM experiment, its MVD detector and the prototype PRESTO. It also motivates the need for a reliable control system for the prototype. The second part is the main part of this thesis. It is in most parts written in a guiding style, to set up an EPICS based DCS from scratch, including front ends and archiving. And the last part summarizes the results and states the outlook of this thesis.

Contents

1 Introduction

1.1 The Compressed Baryonic Matter Experiment

The Compressed Baryonic Matter experiment (CBM) is one of the four pillars of physic programs at the Facility for Antiproton and Ion Research (FAIR)[1]. Those are very diverse:

- Atomic & Plasma Physics & Applications (APPA): "high charged atoms, plasma physics, radiobiology, material science" [2].
- Nuclear Structure, Astrophysics and Reactions (NUSTAR): "rare isotope beams, nuclear structure far off stability, nucleosynthesis in stars and supernovae" [2].
- Anti-Proton Annihilation at Darmstadt (PANDA): "charmed hadrons (XYZ), gluonic matter and hybrids, hadrons structure, double Lambda hypernuclei" [2].
- The Compressed Baryonic Matter experiment (CBM): "nuclear matter at neutron star core densities and phase transition from hadrons to quarks" [2]

FAIR is being built next to "GSI, Helmholtzzentrum für Schwerionenforschung" in Darmstadt, Germany and is one of the largest international research projects worldwide. This new international accelerator facility will provide the infrastructure necessary to the aforementioned unprecedented variety of experiments [3].

CBM will focus on different aspects of studying the strong force and quantum chromodynamics (QCD). Being a well established part of the standard model of elementary particles, the theoretical foundation of the strong force, namely quantum chromodynamics, is based on quarks and gluons as its fundamental degrees of freedom. For a profound understanding of phenomena like the confinement of quarks and gluons into hadrons, the generation mass and phase changes in strongly interacting matter, a good collaboration between experiment and theory is needed for new scientific progresses in this field [4]. All around the world, there are heavy-ion collisions experiments (like the Relativistic Heavy-Ion Collider) with fascinating, spectacular new insights, including the discovery of new forms and properties of matter. CBM is designed to continue this tradition and increase our knowledge with a unique experiment design (and a powerful accelerator). The evolution of the early universe is another quest often mentioned. Along the line of investigating phase changes, new insights into the structure of hot deconfined

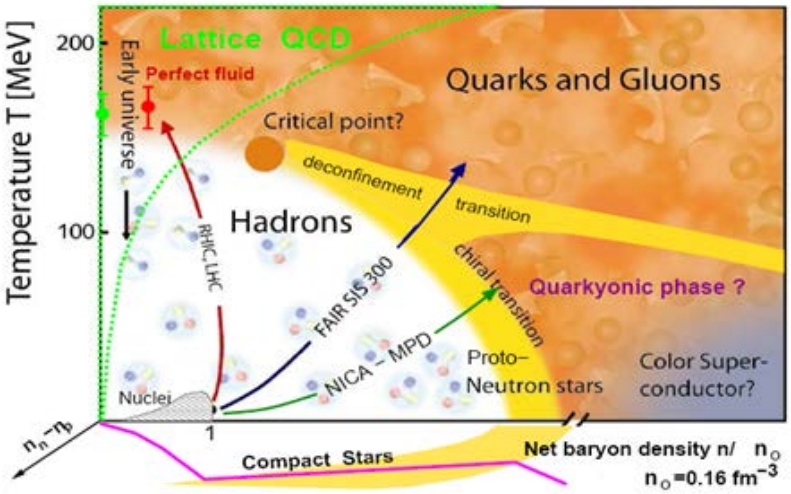


Figure 1: Three dimensional QCD phase diagram (taken from [5]).

matter or Quark-Gluon-Plasma (QGP) can be obtained if the temperature reached in the experiment is high enough, which CBM might not reach. CBM will, however, enter and explore the regions of high baryo-chemical potentials in the phase diagram of QCD matter shown in figure 1.

One of the experimental observables accessible with CBM are multi-differential analyses of particle productions, for example of the production of multi-strange hyperons and so called "rare probes" such as short-lived light vector mesons (ρ, ω, ϕ) and charmonium (J/ψ) with unprecedented precision and statistics [6].

Of course, investigating the yields of produced hadrons in heavy ion collisions is another interesting topic. A result that has been published recently is the analysis of the hadron yields in Ar+KCl and p+Nb measured with the HADES experiment [7] based on the tool of a statistical hadronization model. By using such a thermal model (THERMUS v3.0) the hadron yields of many identified particles were fitted using four parameters such as the chemical freeze-out temperature T_{chem}, the bary-chemical potential μ_B and two further model-dependent parameters. This fit works surprisingly well. Figure 2 shows the fitted yields for the Ar+KCl collision system. The origin of this apparent chemical equilibrium is not clear. CBM measurements to get insights with penetrating probes how the system evolves microscopically towards the final equilibrium state. It seems desirable to do such measurements with CBM, too. CBM being a high rate experiment, it will collect huge amounts of statistics and can possibly include more rare particles in the analysis.

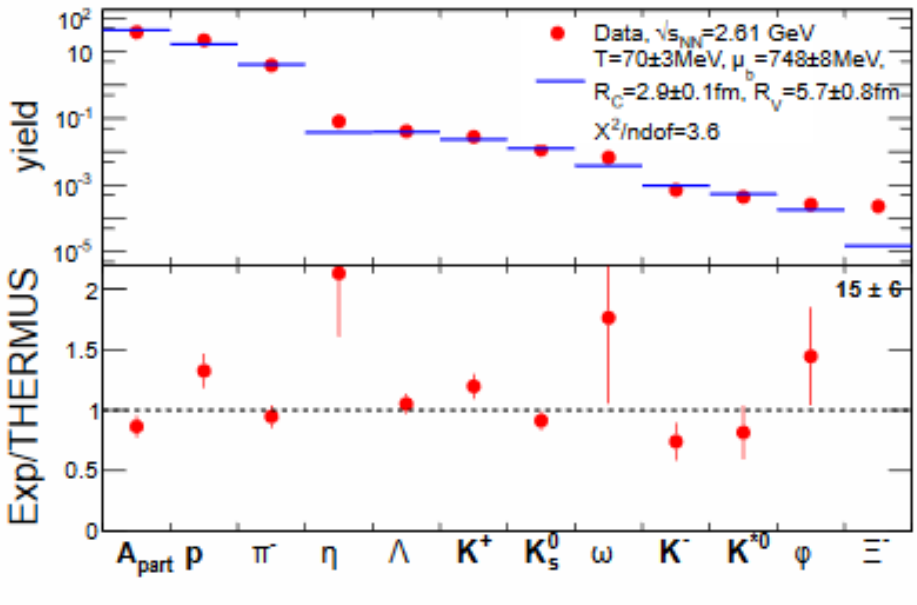


Figure 2: Yields (red circles) of hadrons in Ar+KCL reactions and the corresponding THERMUS fit values (blue bars). The upper plot shows the yields for every measured particle and their fit with THERMUS simulated yields. The lower plot shows the ratio of the experimental value and the THERMUS value which visualizes the quality of the agreement. Taken from [7].

CBM would be the very first experiment with as high interaction rates as ∼10 MHz for their collisions.The MVD works only with interaction rates as high as 100kHz for nucleus-nucleus collisions (Au-Au) and nucleus-proton collisions (p-Au). Such high interaction rates can be achieved by adjusting two parameters in fixed-target experiments. The fixed target's thickness can be adjusted to an interaction probability as high as 1% for each impinging ion. In other words, the beam particles collide with a very high chance with the target particles. Also, raising the accelerator intensity scales the interaction rates. Therefore, the interaction rates of fixed-target-experiments are higher than in the usual collider experiments resulting in larger numbers of collected events and thus more statistics for the analysis of the aforementioned rare particles.

1.2 CBM Structure

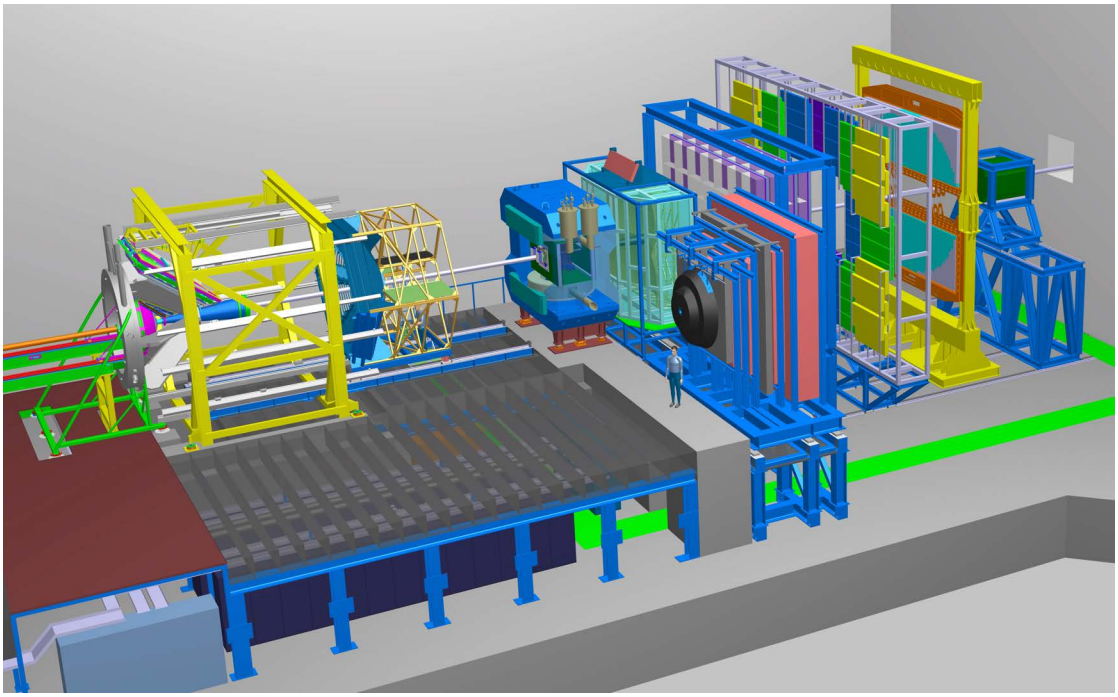


Figure 3: Left - HADES and right - CBM structure (taken from [8])

In the CBM cave (see in figure 3) are two experiments: the High Acceptance DiElectron Spectrometer (HADES) [1] and CBM. The HADES results are promising and a reason for the CBM experiment. The interaction rate of CBM is much higher than these from HADES. The higher the interaction rate, the higher the integrated luminosity, and in turn, the higher the probability for new knowledge will be [9].

CBM has two research configurations: Electron and muon configuration. For both, a variety of detectortypes are required. In the following an overview about the setup is given [10]. CBM starts with the Micro Vertex Detector (MVD) in vacuum and a Silicon Tracking System (STS). Both are inside a superconducting dipol magnet field. The next (looking downstream the target) detector can be exchanged depending on the physics goal. The Ring Imaging Cherenkov detector (RICH) configuration focuses on physics topics with di-electrons, the Muon Chamber System (MUCH) configuration targets di-muon physics. The next detector is a Transition Radiation Detector (TRD) which is followed by a Resistive Plate Chamber (RPC) as a Time of Flight detector (TOF). After the Electromag-

[1] HADES currently still operated at the SIS-18 accelerator of the GSI

netic Calorimeter (ECAL) follows the last detector, a Projectile Spectator Detector (PSD). In the muon configuration, the ECAL will not be used [10].

CBM is a triggerless experiment because the high interaction rate and the associated rising pile-up ruin the ideas of triggering, which are reducing the data volume and bandwidth, elimination of dead time and finally, temporal synchronisation of the detector. Thus, CBM has to be designed as a challenging high-performance free-streaming read-out experiment.

1.2.1 Micro Vertex Detector

The MVD is located close to the collision point and is improving the primary and secondary vertex resolution. The MVD requires high radiation hardness in order to operate with high particle fluxes $\sim$70 MHz/cm^2 [11]. A good spatial precision and a low material budget are required to reach the vertex resolution. The detector is operating in a moderate vacuum $\sim 10^{-5}$ mbar to minimize multiple scattering. So, the 4 stations with Complementary Metal-Oxid Semiconductor (CMOS) Pixel Sensors have to be vacuum compatible. Station 0 (nearest to collision point) has 8 sensors, the following stations have 40, 84 and 160 sensors.

1.2.2 Other Subdetectors

- Silicon Tracking System:

  STS is the core tracking detector of the experiment. It is used for reconstruction of the tracks of all charged particles flying through the detector acceptance and to determine their momenta. The detector features eight tracking stations with double-sided Silicon Microstrip Sensors with a single-hit resolution $\sim$25 μm [6].

- Ring Imaging Cherenkov Detector:

  For the identification of electrons and suppression of pions in the momentum range below 8 GeV/c is the RICH detector needed. It comprises a vessel filled with nitrogen as a radiation material, a glas mirror and two photo detector planes. The photo detector planes detect photon rings with Hamamtsu multianode photo multipliers [12]. The photons are created by the particles which are faster then the speed of light in the respective medium (Cherenkov effect).

- Muon Chamber System:

  The concept of CBM for the experimental challenge for muon measurements to identify low-momentum muons in an environment of high particle densities will be realized by segmenting the hadron absorber in several layers, and placing triplets of tracking detector planes in the gaps between the absorber layers. In order to reducing the meson decays into muons, the absorber has to be compact as possible. The detector development concentrates on the design of fast and highly granulated gaseous detectors based on Gas Electron Multiplier (GEM) technology [12].

- Transition Radiation Detector:

  The TRD is used to track particles and to identify particles as electrons and positrons with p $>$ 1.5 GeV/c. This would be able by three stations each with 4 layers and a total active detector area about 1100 m^2 [12].

- Time of Flight:

  For the hadron identification via TOF measurements a Restive Plate Chamber (RPC) will be used. The required time resolution is 80 ps. For 10 MHz minimum bias Au+Au collisions the innermost part of the detector has to work at rates up to 20 kHz/cm^2 [12].

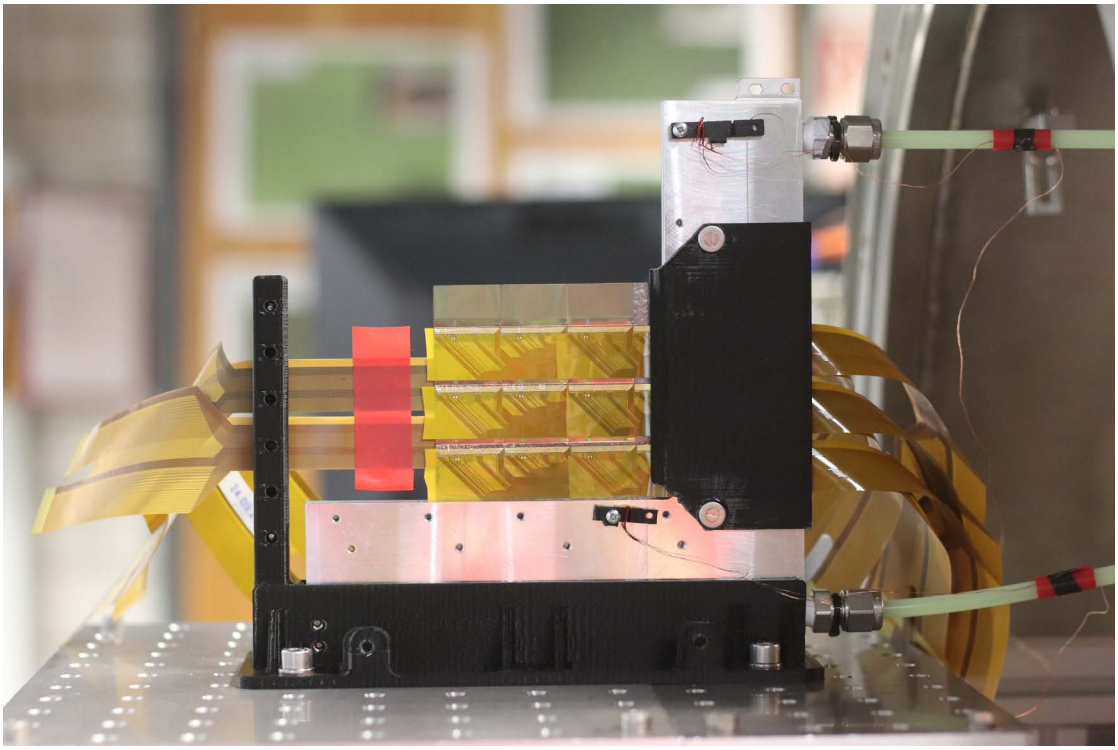


Figure 4: Fully assembled PRESTO in front of the vacuum chamber [9].

1.3 PRESTO - Prototype of the Second Station of the CBM-MVD

The Prototype of the Second Station of the CBM-MVD (PRESTO) includes the heatsink, the carrier, flexible printed circuit (FPC) cables and sensors which together form a quadrant of the MVD [13, 14]. The carrier is a 8x8 cm^2 and 380 μm thick bare Thermal Pyrolytic Graphite (TPG) plate which holds 15 MIMOSA-26 sensors, 9 at the front with a 3x3 arrangement and 6 in the back with a 2x3 arrangement. Figure 4 shows a photograph of PRESTO.

The sensors are positioned that the active area of the sensors has an overlap by 500μm on the front and back side to fill the non-sensitive area of the on-sensor front end electronic (FEE). PRESTO has an active area of 6x5 cm^2 [13]. The MVD would have 16 of these quadrants (four per station), up to a carrier size of 14x12cm^2 with up to 28 final MIMOSIS sensors with an active area of 12x9 cm^2. The MVD would then have a total of 288 MIMOSIS sensors.

PRESTO is needed to study the feasibility of integration many thinned sensors on a carrier to be operated in vacuum, the material budget of the sensors, the ability of cooling the heatsink and finally the control system needed to operate it safely 24/7. Hence, it allows testing many relevant aspects of the hard- and software for safe operation.

Devices are needed for vacuum, cooling and voltage to make this possible. Also, gauges are required that provide independent values of devices. The interaction of both allows independent monitoring and controlling by detector control systems (see more in chapter 1.4). The following list states the device categories and devices of our system for PRESTO.

- *Low Voltage:* A **Hameg HMP 4030** power supply (I/O: USB via FTDI chips is providing the voltage for the converter boards, so in the end for powering the sensors. In addition, it powers the TRB3 FPGA boards for the read-out of the sensors.
- *Cooling Circuit:* **Huber CC-405** (I/O: RS-232, 700W at +20 °C, 5L heat bath capacity) is our cooling system for PRESTO. For measuring the flow of the coolant, two **Kobold DPM** flow meters are used. They are placed in the supply and return lines between the heat sink, of PRESTO and the cooling system. The flow meters in turn are connected to an ATmega 32u4 micro controller (I/O: USB-Serial). Monitoring the temperatures is done with a custom PT100 board (see environment monitoring)
- *Vacuum System:* We use a turbo vacuum pump and a rough vacuum pump. The vacuum gauge controllers allowing to monitor the pressure are a **Vacom MVC-3** (I/O: RS-232) and two **Balzers PKG 020** (Analog $V_{(}out)$).
- *Environment Monitoring:* The in house (IKF) developed 8 channel **PT100 temperature measurement board** (ATmega 328p with I/O: USB Serial) is used. And for the relative/absolute humidity and for the ambient temperature, we use a **Lufft OPUS 20** (I/O: TCP/IP).
- *DAQ bridge:* For the bridge between DCS and DAQ it is planed to use TRB3/trbnet registers (I/O: trbnet)

In table 1 all PRESTO relevant devices are listed with information about the number of channels, the in-/output connection, the measurable values and the precision.

In figure 5 a layout of the interaction of all subsystems and their devices is shown is giving hints to the existing dependencies in our system. For safe operating states, dependencies between the systems (low voltage, cooling system, vacuum system and environment monitoring) need to be considered. An example for such a dependency would be the cooling system that has to perform properly when the chamber is pumped to vacuum and the sensors are powered and dissipating heat. Otherwise, the sensors could over heat and be destroyed. One possible way to cope with this dependency within the DCS would be to disallow switching on the sensors when the chamber is pumped and the cooling system is not running. Hence, combining data from multiple systems is favourable and even needed for ensuring the safety of the detector.
An additional way of using this combined data is to use it for plausibility checks. For example the values measured by devices which are measuring the same quantity independently can be compared and potential inconsistencies found. In some cases, such inconsistencies are easy to interpret, in other cases it could be more complex. An example inconsistency in the cooling circuit would be if the set temperature of the cooling system deviates strongly from the temperature sensor on the heat sink. Possible reasons are listed in the following:

- leak in the cooling circuit (check by comparison of the flow rates from the dedicated flow meters and the flow rate provided by the cooling system → leak in the area between the cooling system and on of the flow meters or between the flow meters themselves)
- temperature sensor on the heat sink or of the cooling system device is defective (check by comparison with yet another temperature sensor)
- cooling capacity is overloaded (check by comparison of the bath temperature and the set temperature of the cooling system)
- cooling system pump is off or defective (check if the flow rates of the flow meters are dropping)

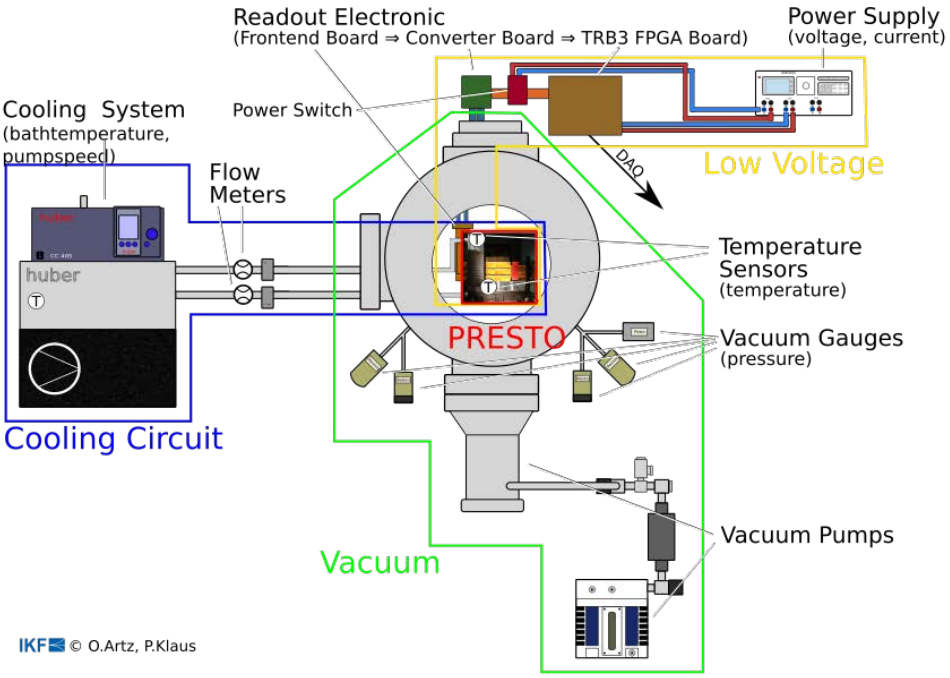


Figure 5: Schematical layout of the interacting systems (Low Voltage - yellow, Cooling Circuit - blue and Vacuum System - green) and of PRESTO (red). An additional arrow indicates the data acquisition (DAQ), which is not covered in this thesis though.

Manufacturer	Device	Channels	I/O	Measurable values	Precision or Accuracy [2]
Hameg (now R&S)	HMP 4030	3	USB (FTDI)	voltage	1mV/1mA
Huber	CC-405	(1)	RS-232	settable: cooling temperature	±1%
Kobold	DPM	(2)	USB (ATmega 32u4 μC)	volumetric flow	±2.5 %
Vacom	MVC-3	(3)	RS-232	pressure	±0.1%
Balzers	PKG 020	3	analog V_{out}	pressure	±1.5% for TPR-Gauge/ ±1% for IKR-Gauge
IKF	Pt100 board	8	UART (FTDI)	temperature	∼10 mK
IKF	ADCuC	8/2	USB Serial	voltage/current	12/16 bit (depend on chip)
Lufft	OPUS 20	n/a	TCP/IP	temperature, humidity	0.3°C, 2% RH

Table 1: Table of PRESTO devices with the EPICS relevant informations from data sheets (taken from [9, 15–20].

[2] whichever is worse

1.4 The need for a Detector Control System

To operate the prototype with all devices involved in the most autonomous manner a Detector Control System (DCS) is needed to ensure the safety of machine and its users. Once established, this DCS may guide the way towards the DCS of the final MVD.

This thesis describes the setup of many required components of the DCS for the prototype, which follows in the following main section. It will start with a general description of DCS followed by a specific description of our implementation based on EPICS.

2 Detector Control System

The main tasks of control systems compare monitoring, visualization, archiving, tracing (protocolling), control (manual, automatic, open or closed loop) and alarming/alerting [21].

A detector has a large phase space of operating states. For protection and safe operation, this phase space must be limited as not all working points are allowed states. In an abstract way, detector control systems can be seen as a mean to ensure that the detector will be in the safe part of the possible operating phase space.

Control systems usually consist of multiple different subsystems with different devices based on different technologies, control architectures and operational requirements. Such a detector control system is needed for a coherent system with centralized operations for the system. DCS is originally a term from the industrial setting and meant Distributed Control System. Distribution is needed for systems which need to be up and running 24/7 reliably, because if a subsystem (smallest distributed unit) has a blackout, the global system has to continue. In industrial systems, a total blackout could result in capital damage. Nowadays, scientific systems are more and more complex and huge financial and scientific damage would be the result, if the experiments have blackouts during beam times. Without system specific know-how it shall be possible for operators to control the system if provided with an informative yet unobtrusive control interface together with concise instructions. If there are device specific issues they have to consult the experts. Thus, DCS works as a translator between the different subsystems and the operators (shifters as well as detector experts).

DCS is traditionally split in the following two categories: slow and fast control. The boundaries between the two was changing with time. With ever faster computers and electronics, systems formerly called "fast" would now be considered "slow". In addition, the limits are different for every system. In the following, I will give two examples for requirements on our DCS and how they can be classified as slow/fast control:

- Slow Control (SC): Set Cooling System on/off
  - reaction time: $\sim$ few seconds
  - reaction by software/human

- Fast Control (FC): Shut down sensors as a protection against local electrical or thermal overload in the event of latch-up
  - reaction time: $\sim 10 \mu s$
  - reaction by firmware/hardware(/software)

2.1 EPICS - Experimental Physics and Industrial Control System

For the DCS of PRESTO we have implemented the Experimental Physics and Industrial Control System (v3.15.5) for 24/7 reliability tests and also to ensure a safe operation of the device at any time [22]. It is being used in many research facilities. EPICS is provided under an open source licence called EPICS Open License, which is similar to the BSD licence [23]. Consequently, downloading and using EPICS is free of charge (under the terms of the license). EPICS uses its own protocol called Channel Access (CA) for its communication. The user can use the given tools and also can program his own tools [24]. The initial design and implementation of EPICS was done in 1989 by the controls group of the Los Alamos Accelerator Technology Division. The project's website states that "EPICS is currently being co-developed by the Accelerator Technology controls group at Alamos National Laboratory and the Advanced Photon Source control group at Argonne National Laboratory" [25].

The original paper from 1991 establishing EPICS in the scientific community was describing the system as follows:

"It is a distributed process control system built on a software communication bus. The functional subsystems, which provide data acquisition, supervisory control, closed loop control, archiving and alarm management, greatly reduce the need for programming. [...] The system is scalable from a single test station with a low channel level count to a large distributed network with thousands of channels. [...] The hardware and software was selected for each functional subsystem for rapid application development, modification and extensibility at all levels to meet the demands of experimental physics." [25]

See figure 6 for an architectural overview of EPICS and its functional subsystems.

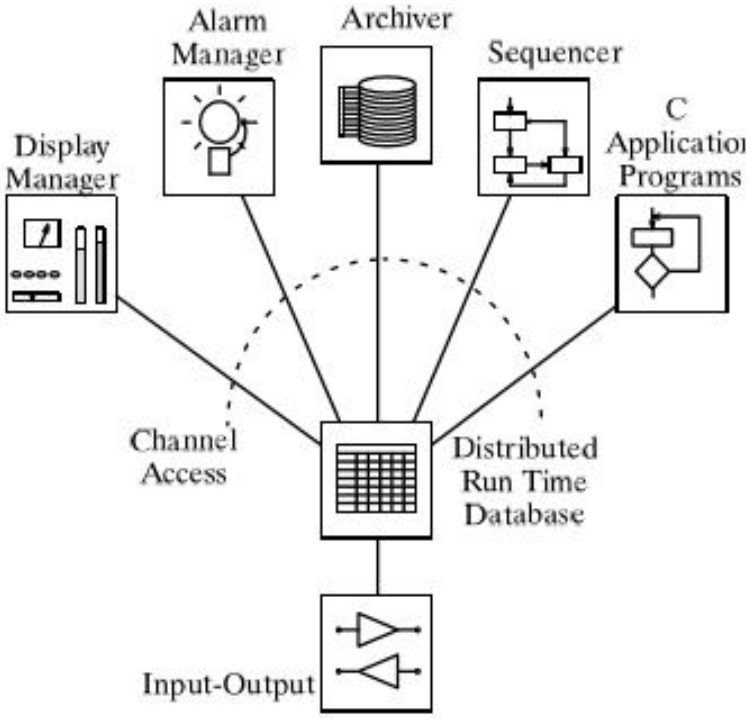


Figure 6: Functional subsystem layout of EPICS (taken from [25])

2.1.1 Install EPICS base and the most commonly used EPICS modules

Before the work with EPICS could started, EPICS is needed. In the following the install of 'EPICS base' and the most commonly used EPICS modules are shown.

How to install EPICS base: To install EPICS on a system, the following steps are needed: First of all, a C/C++ compiler and often-used packages for development, such as the make utility need to be installed. Our choice was to store all EPICS relevant software components in one main folder. By setting an environment variable with this path we make it configurable. The following commands will make use of this environment variable. In the first step we create this main folder.

```
EPICS_DIR=~/EPICS
mkdir "$EPICS_DIR"
```

After that, EPICS base is being installed in a sub folder. To do so, we use the `curl` command to download the source code archive, extract it to the folder 'base-3.15.5' and set a soft link pointing from 'base' to 'base-3.15.5'.

```
cd "$EPICS_DIR"
curl -O https://www.aps.anl.gov/epics/download/base/base
   ↪ -3.15.5.tar.gz
tar -xf base-3.15.5.tar.gz
ln -s ./base-3.15.5 ./base
cd base-3.15.5
```

Before compiling, the EPICS internal environment variable `EPICS_HOST_ARCH` should be set according to the CPU architecture. The following command find out the right one by itself (e.g. `linux-x86\_64` or `linux-arm`). The compilation itself is done with the make command.

```
export EPICS_HOST_ARCH=$(./startup/EpicsHostArch)
make
```

A short test to check if EPICS will running with a basic command line prompt (`epics>`) could be done with `./bin/$EPICS_HOST_ARCH/softIoc`, see [26] for more detailed information.

EPICS relys on a couple of environment variables to be set in order to work properly. These can put into `/etc/bashrc` so they do not have to be entered again for every new terminal:

```
export EPICS_ROOT=~/EPICS
export EPICS_BASE=${EPICS_ROOT}/base
export EPICS_HOST_ARCH=`${EPICS_BASE}/startup/EpicsHostArch`
export EPICS_BASE_BIN=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}
export EPICS_BASE_LIB=${EPICS_BASE}/lib/${EPICS_HOST_ARCH}
if [ "" = "${LD_LIBRARY_PATH}" ]; then
  export LD_LIBRARY_PATH=${EPICS_BASE_LIB}
else
  export LD_LIBRARY_PATH=${EPICS_BASE_LIB}:${LD_LIBRARY_PATH}
fi
export PATH=${PATH}:${EPICS_BASE_BIN}
```

How to install the most commonly used EPICS modules: Commonly, EPICS base is being supplemented by additional modules to facilitate supporting new device.
As the most prominent example here, the generic EPICS device support for 'byte stream' based communication interface devices (controllable by sending and receiving strings) is provides by the additional module 'StreamDevice 2' [27]. In principle it provides the support for 'byte stream' based devices and allows to decouple the protocol definition of the device from the underlying communication bus. 'StreamDevice' is a supplement for the 'asynDriver', which is a general purpose facility for interfacing device specific code to low-level communication drivers [28]. Also the collection of software tools named 'synApps' are needed for EPICS, because it contains beamline-control and data-acquisition components for EPICS based control systems [29]. 'StreamDevice' and 'Asyn' together with many other commonly used modules are part of the 'synApps' extension collection. Only some of these are necessary in the context of this thesis: 'StreamDevice', 'Asyn' and the 'Sequencer' module, while the latter is important to implement finite state machines but was not used so far. 'SynApps' is being installed with the following commands:

```
# First, some packages required for the compilation of the
   ↪ relevant modules need to be installed. StreamDevice e.g.
   ↪ needs the following on a Debian based computer:
#sudo apt-get install libpcre3-dev libpcre++-dev
```

```
# Now, creating a sub folder for the modules, downloading and
   ↪ unpacking the archive:
mkdir $EPICS_ROOT/modules
cd $EPICS_ROOT/modules
curl -O https://www1.aps.anl.gov/files/download/BCDA/synApps/
   ↪ tar/synApps_5_8.tar.gz
tar -xf synApps_5_8.tar.gz
```

In the files `config/RELEASE` and `synApps/Makefile`, any modules which are not relevant should be removed. Finally, the `make` command is used to compile the modules.

2.1.2 IOC - Input/Output Controller

EPICS has a network-based client server architecture. A central element of any EPICS system are the so-called Input Output Controllers (IOCs) which provide access to the process variables (PVs) and act as servers in the local network. IOCs are directly connecting to the laboratory hardware and implement the protocols of these device. IOCs are the smallest unit of a distributed EPICS system. A client is a program which can access and process data of the EPICS servers (IOCs) [30]. The IOCs of EPICS are not limited to act as servers but can also act as clients in addition, as they can be configured to access and process data from another IOC [30].

The relevant protocol for the client-server communication in an EPICS systems is called Channel Access (CA) described in the following section.

CA - Channel Access: The protocol of EPICS is called Channel Access (CA). CA uses both, the transmission control protocol (TCP) as well as the user datagram protocol (UDP) for the transport layer. CA has a simple communication structure with only three exposed methods between the CA-Server and CA-Client [30]. The naming is relative to the client's perspective:

- **get**: Get the value of the PV of an IOC (server) [30, 31].
  (command line tool: caget)

- **put**: Set a PV to a new value. The client instructs the server to set the value of a PV to the specified value. The server answers with the new setpoint, which could differ from the client given value, e.g. if the given value is higher than the maximal possible value or the given value precision is higher than that of the IOC [30, 31].

  (command line tool: caput)

- **monitor**: The client subscribes to updates of a PV. If there are any modifications of a PV (e.g. new measurement) then the server of the PV sends the modification to the client. Monitor is in a way similar to polling (client asks the server for modifications permanently) but much more efficient and scalable, making it suitable for real-time applications. It is comparable to push-notifications on mobile devices (server actively notifies clients about modifications) [30, 31].

  (command line tool: camonitor)

CA and other programs like Control-System-Studio (CS-Studio) are working with TCP and UDP. Both protocols work with the concepts of ports to distinguish endpoints further than just by IP addresses. EPICS and CA by default uses ports 5064 and 5065. Hence, all IOCs communicate on the same ports so it is needed that every EPICS-IOC has its own IP-address if they should run on the same physical host computer. Additional alias IP-addresses can be assigned in the file `/etc/network/interfaces` on Debian-based Linux computers. For an example, it could look like the following:

```
auto eth0:1
allow-hotplug:eth0:0

iface eth0:6 inet static
address 192.168.10.101
netmask 255.255.252.0
```

The first line defines the acquisition of an IP via DHCP for the main network interface.
For more IOCs the number x in `auto eth0:x` and in `address 192.168.10.xxx` have to change with the same number.

How to get a basic IOC up and running: In the following the commands for the boilerplate of a basic IOC are shown. In the stated commands, `EXAMPLE` is a placeholder for the actual desired name for the IOC.

```
# Make sure the environment variables such as EPICS_BASE and
   ↪ EPICS_HOST_ARCH are set as described in chapter "How to
   ↪ install EPICS base".

# Create a new folder for the EXAMPLE IOC and initiate a
   ↪ bootstrapping.
mkdir -p ~/EPICS/IOCs/EXAMPLE
cd ~/EPICS/IOCs/EXAMPLE
makeBaseApp.pl -t ioc EXAMPLE
make
makeBaseApp.pl -i -t ioc EXAMPLE
make

# Make the startup script executable:
cd ~/EPICS/IOCs/EXAMPLE/iocBoot/EXAMPLE
chmod u+x st.cmd

# Test run the freshly created IOC with: ./st.cmd
```

After running the commands above, a typical IOC files structure is created ("bootstrapped") in the folder `~/EPICS/IOCs/EXAMPLE`. Figure 7 shows the relevant files and folders. Note that compilation output generated by running the `make` command is omitted in the illustration.

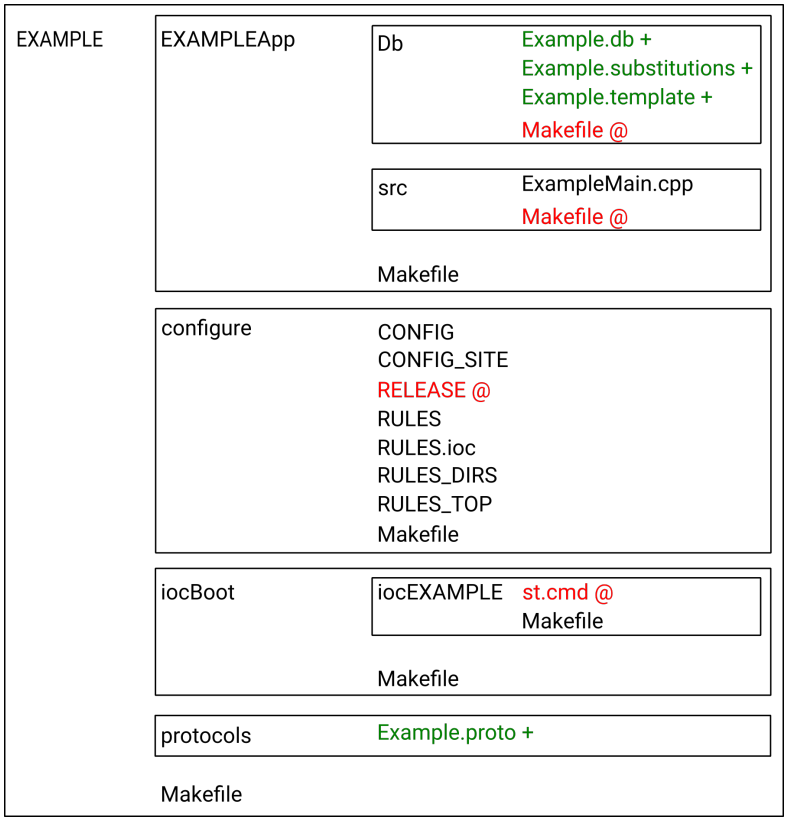


Figure 7: File system tree of an EPICS IOC: files whoch were changed in the process of creating the example IOC in this thesis are marked in red ("@") and the files which were are marked in green ("+").

recordtype	full name	use case
ai	analog input	"obtain an analog value from hardware and then convert it to engineering units"
ao	analog output	"output values to digital-analog converters"
bi	binary input	"obtain a binary value of 0 or 1"
bo	binary output	"store a simple bit (0 or 1) value to be sent to a Digital Output module"
Calc	Calculation	"perform algebraic, relational, and logical operations on values retrieved from other records"
...	...	a complete and detailed list can be found in [32]

Table 2: Table of mainly used recordtypes with their use case

Detailed Description of the IOC Components: For a typical real-life IOC some changes and additions are needed (see figure 7). In the following, the meaning and content of the IOC specific files are detailed:

.db :

The IOC includes the 'IOC database' which is a central part of any EPICS-based control system. One of the design principles of EPICS being 'configure - don't program', many problems can be tackled by just writing such a .db file which is a configuration file interpreted by an EPICS IOC. Their main content are named 'records'. Every 'record' consists of one 'recordtype', has some different 'fieldtypes' which are assigned 'fieldtypevalues' and has its own PV name which is its unique identifier in the EPICS system and is needed to access and process the 'record' by other IOCs [30]. In the following, the basic structure of a record is shown:

```
record (recordtype, "PV_name") {
        field (fieldtype1, "fieldtypevalue1")
        field (fieldtype2, "fieldtypevalue2")
                .
                .
                .
        field (fieldtypen, "fieldtypevaluen")
        }
```

The 'recordtype' is a short sequence of characters usually an abbreviation of their full name. Table 2 shows the most frequently used ones and their use case.

fieldtype	function	fieldtypevalue example	available in recordtype
DTYP	Device Type	stream	any
INP	Input Link	@HAMEG_HMP4030.proto Get-Voltage	all 'in'
OUT	Output Link	@HAMEG_HMP4030.proto Set-Voltage	all 'out'
SCAN	Scanning Algorithm	2 second	ai, ao, bi, calc
FLNK	Forward Link	$(sys):$(sub):COOLING:$(csys):T0	ai, ao ,bo
EGU	Engineering Units	V	ai, ao, calc
PREC	Display Precision	3	ai, ao, calc
ADEL	Archive Deadband	0.005	ai, ao, calc
MDEL	Monitor Deadband	0.001	ai, ao, calc
ASLO	Adjustment Slope	0.001	ai, ao
AOFF	Adjustment Offset	0	ai, ao
HOPR	High Operating Range	100.0	ai, ao, calc
LOPR	Low Operating Range	-50.0	ai, ao, calc
DRVH	Drive High	70.0	ao
DRVL	Drive Low	0.0	ao
more information and detailed list to be found in [32]			

Table 3: Table of mainly used fieldtypes with examples and the availability in recordtypes

The mainly used 'fieldtypes', as well as their function, an example for a suitable 'fieldtypevalue' and their availability in 'recordtypes' is shwon in the table 3.

Further, an excerpt of the source code for an analog output and an analog input record for our power supply HAMEG HMP4030 is shown:

```
record(ai,"$(sys):$(sub):POWER:$(vsys):$(CHAN_NAME):
    ↪ GetVoltage") {
        field (DTYP, "stream")
        field (INP,  "@HAMEG_HMP4030.proto GetVoltage($(
            ↪ CHAN)) $(PORT)")
        field (SCAN, "2 second")
        field (EGU, "V")
        field (PREC, "3")
        field (ADEL, "0.005")
        field (MDEL, "0.001")
        }

record (ao, "$(sys):$(sub):POWER:$(vsys):$(CHAN_NAME):
    ↪ SetVoltage") {
        field (DTYP, "stream")
        field (OUT,  "@HAMEG_HMP4030.proto SetVoltage($(
            ↪ CHAN)) $(PORT)")
        field (EGU, "V")
        field (PREC, "3")
        field (ADEL, "0.005")
        field (MDEL, "0.001")
        }
```

The `GetVoltage` record is used to read the set voltage from the device while the `SetVoltage` record is used to change ("set") the set voltage of the device. While the `GetVoltage` record is being updated automatically every two seconds (see 'SCAN'), the `SetVoltage` record is lacking the `SCAN` field (because it would result in constantly updating the set voltage). The 'fieldtypevalue' of `OUT` and `INP` is a link to the protocol file (filetype:'.proto') and the internal working name, in which the communication (in-/output) required to obtain or set the value from/on the device is described. All identifiers starting with the '$' character are macros. Macros are placeholders which are later replaced - either by running the `make` command or the latest at the startup of the IOC via 'st.cmd' see below on page 26.

Alternativly to the directly writing a '.db', a '.template' and '.substitutions' file can be used to create a '.db' file upon compiling. They also have to be placed in the 'Db' folder. Figure 7, which shows a typical IOC file system tree, includes those. The '.template' file includes the records of a channel. The channel specific parts (e.g. channel number) consist or macros. The macros are replaced in the '.substitutions file' (also possible, to replace macros in the 'st.cmd' file). After using the command `make` a '.db' file is created with all replaced macros of the '.substitutions' file in the subfolder '.db'.

This programming style is useful because it reduces time for preparation and makes it more flexible to implement devices with many equal channels where only the channel number differs.

In the following is the '.substitutions' file for the extract of the '.template' file for the analog in-/output record of the HAMEG HMP4030:

```
file HAMEG_HMP_CHANNEL.template {
        pattern { CHAN_NAME, CHAN }
                { "CHAN1",      1 }
                { "CHAN2",      2 }
                { "CHAN3",      3 }
        }
```

Makefiles :

The Makefiles are used to compile the source code. In the following, a list of all lines for the 'Makefiles' which have to be added to the basic IOC (here called EXAMPLE) Makefiles is given:

- Db/Makefile:

  ```
  DB += EXAMPLE.db
  ```

- src/Makefile:

  ```
  # Add the default database entries for the additonally
     ↪ used modules:
  EXAMPLE_DBD += stream.dbd
  EXAMPLE_DBD += asyn.dbd
  EXAMPLE_DBD += drvAsynSerialPort.dbd

  # Add the compiled libaries of the additonally used
     ↪ modules:
  EXAMPLE_LIBS += stream
  EXAMPLE_LIBS += asyn
  ```

RELEASE :

The RELEASE file lists the paths of the EPICS modules which are needed by the IOC. The following lines have to be added to the basic RELEASE file, as we need the 'seq', 'asyn' and 'stream' modules:

```
MODULES = $(EPICS_ROOT)/modules

# The following commands set the path for the frequently
   ↪ required the sequencer, the asyn and the StreamDevice
   ↪  modules:
SNCSEQ = $(MODULES)/synApps_5_8/support/seq-2-2-1
ASYN = $(MODULES)/synApps_5_8/support/asyn-4-26
STREAM = $(MODULES)/synApps_5_8/support/stream-2-6a
```

As the 'RELEASE' file is later imported into the 'Makefile' when running `make`, the $(.) identifiers are stated in the makefile syntax and will be replaced - either by environment variables or by specifying then after the `make` command (for example `make EPICS_ROOT=/home/scs/...`).

st.cmd :

The 'st.cmd' file is the startup file. If you start EPICS, the 'st.cmd' file lists all needed '.db' files to run EPICS-IOC. The open macros must be replaced in the 'st.cmd' file. Consequently, the 'st.cmd' file is the highest hierarchical file in the sense of macro substitutions, hence the last chance where the number of PVs of an IOC can be customized. This file is also needed to define the connection settings with any hardware/device. The relevant file descriptor of a device can be found on a Linux computer by using the command `ls -l /dev` after connecting the device. This file descriptor has then to be specified in the `drvAsynSerialPortConfigure` field of the 'st.cmd' file. It is possible to assigned alias names for the file descriptors in `/etc/udev/rules.d/99-XX.rules` (XX could be any name).

```
# The environment variable STREAM_PROTOCOL_PATH defines the
   ↪  search folder for protocol files:
epicsEnvSet ("STREAM_PROTOCOL_PATH", "${TOP}/protocols")

# Initialize the IOC's database with the default entries:
dbLoadDatabase "dbd/EXAMPLE.dbd"
EXAMPLE_registerRecordDeviceDriver pdbbase
```

```
# Define the connection to the hardware device and the
   ↪ relevant properties:
drvAsynSerialPortConfigure("EXAMPLE_PORT","/dev/
   ↪ EXAMPLE_9479")
asynSetOption ("EXAMPLE_PORT", 0, "baud", "115200")
asynSetOption ("EXAMPLE_PORT", 0, "bits", "8")
asynSetOption ("EXAMPLE_PORT", 0, "parity", "none")
asynSetOption ("EXAMPLE_PORT", 0, "stop", "1")
asynSetOption ("EXAMPLE_PORT", 0, "clocal", "N")
asynSetOption ("EXAMPLE_PORT", 0, "crtscts", "N")

# Here, the custom .db file is loaded and the final macros
   ↪ substitutions are defined:
dbLoadRecords("db/EXAMPLE_CHANNEL.db", "PORT=EXAMPLE_PORT,
   ↪ sys=CBM,sub=MVD,vsys=PRESTO_RECIPIENT")
```

.proto :

This file implements the device-specific protocol (thus named '.proto') and represents an easy way to formulate the (expected) conservation with devices having a serial byte-stream communication interface. This requires the 'stream' EPICS module. The communication is split into small, self-consistent named units which are called "protocol" in the 'Stream Device 2' terminology. Ideally, each "protocol" covers one action like setting or getting a value, although more complex scenarios can also be covered. Within a "protocol", multiple subsequent `in` and `out` statements define the communication. The `out` statements define commands (bytes) to be sent by the IOC to the device whereas the `in` statements define the expected output/response and (if applicable) can be used to parse back values. The "protocol" could contain several "out"s and "in"s e.g. if commands to selecting a channel and changing its value have to combined.

In the following, the basic structure of a "protocol" with the name `GetWork` ↪ `ingNameStatus` with its `out` and `in` statements is shown:

```
GetWorkingNameStatus {
  out "GIVE STATUS";
  in  "%i"
}
```

For a more complex example, an excerpt of the `HAMEG_HMP4030.proto` file, implementing the "protocols" needed for the records specified in the earlier '.db' record example (see page 22) is shown:

```
GetVoltage {
  out "INST OUT\$1";
  out "VOLT?";
  in "%f";
}

SetVoltage {
  out "INST OUT\$1";
  out "VOLT %f";
  @init { GetVoltage; }
}
```

Both "protocols" contain two "out"s. The first `out` has the function parameter '\$1' which is needed to select the channel. And `SetVoltage` contains `@init { GetVoltage;}` which is needed to get an initial value for the record `SetVoltage` when starting the IOC, because of the output recordtype (ao).

If there is no device manual, the method of 'Reverse Engineering' helps to find the right "in"s and "out"s.

All created PVs of an IOC are list by using the command 'dbl' from within the shell provided by the IOC (see table with all PRESTO including PVs in chapter 5 on page 60). See more about the EPICS shell in chapter 2.3 on page 34.

2.2 Archiver

Modern physics experiments become larger and larger and use many devices. Every device has many specific settings and values which are relevant for the analysis of the experimental data of the experiment. An archiver is usable to explain unwanted changes of values by chronological correlation/search of the archived values. With this chronological correlation knowledge it is possible to investigate and improve the settings for the next beam time. Also, the time evaluation of the values is needed for trend plots for the operator and to ensure the reproducibility of the scientific measurements.

Therefore, the task for the archiver in our DCS is storing time-series data of selected or all PVs in a PostgreSQL database. It has to acquire value updates as an EPICS client and store any relevant changes in the database.

As a side remark, the archiver will not store the experimental data, which instead is sent to a server processing farm First-level Event Selector (FLES).

For our usage we want an archiver that is compatible to connect to EPICS-IOCs/PVs and provide the time-series data to our front-end GUI application CS-Studio (an example is shown in figure 12 on page 42). The software of choice for our archiver is the CS-Studio RDB archiver application [22]. The installation and configuration was done with the help of Peter Zumbruch and scripts of Martin Mitkov [33, 34].

With vertical database partitioning, the archiver increases its performance. Further, partitioning makes the database more scalable and eases periodic backups, as well as data pruning.

2.2.1 Installation of the RDB Archiver

As first step, a PostgreSQL database server has to be set up (minimum version 9.7.). In the next step, an environment value should be set, pointing to the hostname of the database server.

```
DBSRV=cbmmvd_pgsql_server
```

To resolve the hostname to its IP address, it can be specified in the `/etc/hosts` file if no DNS server is available to be configured accordingly. In our case, the archiver engine is running on the same host as the PostgreSQL so the hostname is resolved to 127.0.0.1 (localhost), so that our line in `/etc/hosts` reads: `127.0.0.1 cbmmvd_pgsql_server`

Now the archiver 'config' and 'engine' are download in the archiver folder (more information in [34]):

```
mkdir ~/CSS-Archiver
cd ~/CSS-Archiver
unzip "https://hades-wiki.gsi.de/foswiki/pub/DaqSlowControl/
   ↪ SCSArchiverCSStudio/archive-config-4.1.0-linux.gtk.x86_64
   ↪ .zip"
unzip "https://hades-wiki.gsi.de/foswiki/pub/DaqSlowControl/
   ↪ SCSArchiverCSStudio/archive-engine-4.1.0-linux.gtk.x86_64
   ↪ .zip"
```

The files 'pluginCustomization_dcs_engine.ini', 'postgres_schema.sql' and 'postgres_partitioning.sql'found in section A.2 on pages 61, 70, 70, have to be put in the folder `~/CSS-Archiver`. The file `postgres_schema.sql` contains the SQL code with the schema (structures like tables and relationships) to setup the archiver database. The `postgres_partitioning.sql` modifies the schema to make use of table partitioning for the table holding the individual samples/readings. Those files are injected into the database by loading them with the `psql` PostgreSQL command line client:

```
psql -h $DBSRV -p 5432 -U postgres -a -f postgres_schema.sql
psql -h $DBSRV -p 5432 -U archive -d archive -a -f
   ↪ postgres_partitioning.sql
psql -h $DBSRV -p 5432 -U postgres -a -f postgres_schema.sql
```

In the second command, the 'archive' user is specified with the `-U` parameter in the database 'archive' is selected with the `-d` parameter, which both have been created when running the first command with the administrative user `postgres`. To fix missing database user permissions, the first command has to execute once more after setting up the partitioning (which will be improved in the future).
The partitioning is done with a stored procedure that needs to be executed once to be initialized with the start day of the current week:

```
psql -h cbmmvd_pgsql_server -p 5432 -U archive -d archive -a -c
   ↪  \
"SELECT archive.sample_update_partitions('2017-06-01'::
   ↪ timestamp, 'archive', 'archive', 'week');"
```

To update the database partitioning regularly, a 'cronjob' can be used to call the stored procedure regularly at every full hour (minute = 0):

```
0 * * * * /usr/bin/psql -h cbmmvd_pgsql_server -p 5432 -U
   ↪ archive -d archive -a -c "SELECT archive.
```

```
    ↪ sample_update_partitions('2017-06-01'::timestamp, 'archive
    ↪ ', 'archive', 'week');"
```

2.2.2 Configuration and Customization

After that, the `pg_hba.conf` access policy file of PostgreSQL has to be adjusted in `/var/lib/pgsql/data` by adding the following two lines, to allow access as archive user (write access) from the same host and as report user (read access) from the same network:

```
hostnossl   all   archive   samehost   trust
hostnossl   all   report    samenet    trust
```

As the RDB archiver engine is written in Java using the Eclipse framework, but running in headless mode, 'customizations' (settings) have to be provided in a specific configuration file, here called `pluginCustomization_dcs_engine.ini` contains the following lines:

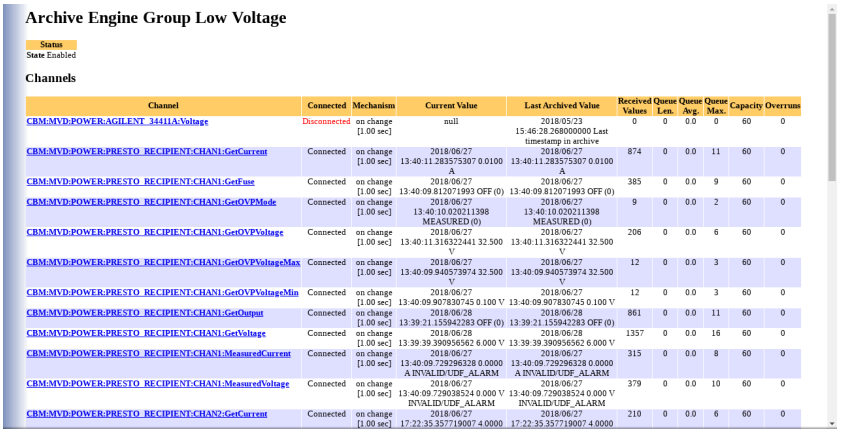
```
org.csstudio.archive.rdb/url=jdbc:postgresql://
   ↪ cbmmvd_pgsql_server:5432/archive
org.csstudio.archive.rdb/user=archive
org.csstudio.archive.rdb/schema=
org.csstudio.security/secure_preference_location=Instance
org.csstudio.platform.libs.epics/addr_list=192.168.10.101
   ↪ 192.168.10.102
```

The last line `/addr_list` contains the IP-addresses of all IOCs which the archiver should connect to. The archiver also need information which PVs of the IOCs it is supposed to archive and how often (archiving period). These and other main settings (buffer-reserve · write-period / period will provide the buffer size [35], and some more) are to be set in the `dcs_engine.xml` file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE engineconfig SYSTEM "./engineconfig.dtd">

<engineconfig>

<buffer_reserve>4</buffer_reserve>
<ignored_future>1</ignored_future>
<write_period>60</write_period>
<file_size>61</file_size>
<get_threshold>20</get_threshold>
<max_repeat_count>120</max_repeat_count>
```

```
<group>
        <name>Groupname
        </name>
        <channel>
                <name>PV_name
                </name>
                <period>1
                </period><monitor/>
        </channel>
</group>

</engineconfig>
```

The settings are from our original `dcs\_engine.xml` file of our archiver for PRESTO. The values are chosen by the two aspects: what is needed and what is technical possible. One <channel> represents one PV. All PVs of the same function like cooling, voltage, pressure and some else are collected in a <group>. The figure 8 shows the status interface of the archiver process pushing new entries into the PostgreSQL database. The 'Low Voltage' group is shown. In the 9th line, the PV ending in 'GetVoltage' for the channel 1 ('CHAN1') introduced in chapter 2.1.2 is listed.



Archive Engine Group Low Voltage

Status
State Enabled

Channels

Channel	Connected	Mechanism	Current Value	Last Archived Value	Received Values	Queue Len.	Queue Avg.	Queue Max.	Capacity	Overruns
CBM:MVD:POWER:AGILENT_34411A:Voltage	Disconnected	on change [1.00 sec]	null	2018/05/23 15:46:28.268000000 Last timestamp in archive	0	0	0.0	0	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetCurrent	Connected	on change [1.00 sec]	2018/06/27 13:40:11.283575307 0.0100 A	2018/06/27 13:40:11.283575307 0.0100 A	874	0	0.0	11	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetFuse	Connected	on change [1.00 sec]	2018/06/27 13:40:09.812071993 OFF (0)	2018/06/27 13:40:09.812071993 OFF (0)	385	0	0.0	9	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOVPMode	Connected	on change [1.00 sec]	2018/06/27 13:40:10.020211398 MEASURED (0)	2018/06/27 13:40:10.020211398 MEASURED (0)	9	0	0.0	2	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOVPVoltage	Connected	on change [1.00 sec]	2018/06/27 13:40:11.316322441 32.500 V	2018/06/27 13:40:11.316322441 32.500 V	206	0	0.0	6	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOVPVoltageMax	Connected	on change [1.00 sec]	2018/06/27 13:40:09.940573974 32.500 V	2018/06/27 13:40:09.940573974 32.500 V	12	0	0.0	3	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOVPVoltageMin	Connected	on change [1.00 sec]	2018/06/27 13:40:09.907830745 0.100 V	2018/06/27 13:40:09.907830745 0.100 V	12	0	0.0	3	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOutput	Connected	on change [1.00 sec]	2018/06/28 13:39:21.155942283 OFF (0)	2018/06/28 13:39:21.155942283 OFF (0)	861	0	0.0	11	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetVoltage	Connected	on change [1.00 sec]	2018/06/28 13:39:39.390956562 6.000 V	2018/06/28 13:39:39.390956562 6.000 V	1357	0	0.0	16	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:MeasuredCurrent	Connected	on change [1.00 sec]	2018/06/27 13:40:09.729296328 0.0000 A INVALID/UDF_ALARM	2018/06/27 13:40:09.729296328 0.0000 A INVALID/UDF_ALARM	315	0	0.0	8	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:MeasuredVoltage	Connected	on change [1.00 sec]	2018/06/27 13:40:09.729038524 0.000 V INVALID/UDF_ALARM	2018/06/27 13:40:09.729038524 0.000 V INVALID/UDF_ALARM	379	0	0.0	10	60	0
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetCurrent	Connected	on change [1.00 sec]	2018/06/27 17:22:35.357719007 4.0000	2018/06/27 17:22:35.357719007 4.0000	210	0	0.0	6	60	0

Figure 8: Status interface of the archiver for the Low Voltage group

The table in the figure 8 has 11 columns shown. The 'channel' column is for the PV names. 'Connected' is self-explaining; it gives information about the connection status (dis/connected). The third column 'Mechanism' provides information when values goes in the cache (here on change by maximal one second). In the 'Current Value' column the latest value together with its timestamp is started. The 'Last

Archived Value' indicts the value and the associated timestamp for the last time, a value was actually written to the archiver database. This entry in this column is not always identical to the 'Current Value' entry as only changes will be recorded and for some recordtypes (ai, ao) a deadband can be configured to only store a new value if it differs more than a given delta (ADEL - archiver delta) from the last archived value (compare with chapter 2.1.2). The total number of values which the archiver received from EPICS (thus named 'Received Values') is in the sixth column. The 'Queues' columns gives information about current, average and the maximum queue sizes which are the 'Capacity' of the queue (in the second to last column). Information about the 'Capacity' allows knowledge about the storage size. Each queue is a FIFO or ring buffer for an individual PV with a fixed number of entries to ensure a deterministic memory footprint of the archiver process. If a queue runs full during the configured 'write_period' its corresponding 'overrun' content is being incremented (last column). If a channel has a high number of 'overruns', 'ADEL' or 'capacity' should be increased.

2.3 Startup with systemd

For more safety control we operate the EPICS IOCs on a RaspberryPi3 single-board computer (SBC) and a regular x86-64 Linux computer for archiving. But if there is a electricity failure, all EPICS-IOCs and the archiver would have to be restarted by hand. To handle this we creating systemd service files.
Systemd is daemon (program working in background) for Linux-systems. If your system reboots '.systemd' starts and all enabled '.service' files start automatically, too. The '.service' files contains all needed information to start and monitor a program/process. In the following, our `epics_env.systemd` file is shown contain the usual EPICS environment variables:

```
EPICS_ROOT=/home/scs/EPICS
EPICS_BASE=${EPICS_ROOT}/base
EPICS_HOST_ARCH=linux-arm
EPICS_BASE_BIN=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}
EPICS_BASE_LIB=${EPICS_BASE}/lib/${EPICS_HOST_ARCH}
LD_LIBRARY_PATH=${EPICS_BASE_LIB}:${LD_LIBRARY_PATH}
PATH=${PATH}:${EPICS_BASE_BIN}
```

Here, the `EPICS_Host_ARCH` is set manually to `linux-arm` and needs to be adapted to the host architecture, see the paragraph on the internal environment variables `EPICS_Host_ARCH` found in chapter 2.1.1 on page 17 for more information.

Finally, a .service file for every IOC and for the archiver is needed. These have to be on the system (where the programs run) in the folder `/etc/systemd/system`. In the following is our `hameg_ioc.service` file shown:

```
[Unit]
Description=EPICS IOC "HAMEG_HMP4030"
After=network.target

[Service]
Type=simple
User=scs
EnvironmentFile=/home/scs/EPICS/env/epics_env.systemd
Environment="EPICS_CAS_INTF_ADDR_LIST=192.168.10.106"
WorkingDirectory=/home/scs/EPICS/IOCs/HAMEG_HMP4030/iocBoot/
   ↪ iocHAMEG_HMP4030
ExecStart=/usr/bin/procServ -n hameg_ioc -f 20106 /home/scs/
   ↪ EPICS/IOCs/HAMEG_HMP4030/iocBoot/iocHAMEG_HMP4030/st.cmd
Restart=on-abort
```

```
[Install]
WantedBy=multi-user.target
```

New .service file will enable by command `systemctl enable hameg_ioc.service` ↪ . To enable changes of .service files the following commands are needed:

```
systemd daemon-reload
systemd stop hameg_ioc.service
systemd start hameg_ioc.service
```

With `systemd status hameg_ioc.service` the journal (log) output could be check.

The ExecStart statement in the .service file describes the startup command to run the software, here the IOC. Instead of directly running the ./st.cmd startup script, we use procServ as a process manager for two reasons: to allow local users to connect to the IOC's built-in shell by serving it on a Telnet session on localhost. Secondly, to automatically restart the IOC in case it crashes.

To open the EPICS command-line, the command `nc localhost 20xxx` in the console while replacing the 20xxx with the port number started in the procServ command in the 'ExecStart' line. We set the convention to use 20000 + last byte of IP address, the IOC runs on. In this example, the IP is 192.168.10.101 and that way, the port is set to 20000 + 101 = 20101.

2.4 CS-Studio

To hold on the overview of a handful of PVs monitoring them in a terminal window would be possible, but if are fast reaction by hand are needed, it would be difficult. To handle this issue, it is possible to use some scripts. A system with hundreds of PVs needs more than scripts. A kind of dashboard is needed. In our system we use Control-System-Studio (CS-Studio) version 4.5.0 as our graphical user interface (GUI) for EPICS. CS-Studio makes it possible to have a high information density via widgets like colored buttons, coloured (flashing) border alarms, spinners to change values and many more.

The collaboration of Deutsches Elektronen Synchrotron (DESY), Spallation Neutron Source (SNS) and Brookhaven National Laboratory (BNL) is developing and maintaining CS-Studio [36]. The first official version of CS-Studio came out in 2007 [37, 38].

How to install CS-Studio: For installing CS-Studio on a system, 'Oracle Java (1.8)' is needed. The decision was made to use the pre-compiled version provided.

```
curl -O https://ics-web.sns.ornl.gov/css/nightly/sns-css-4.5.0-
   ↪ linux.gtk.x86_64.zip
unzip sns-css-4.5.0-linux.gtk.x86_64.zip
```

Start CS-Studio in with `./sns-css-4.5.0/css` and remove all SNS URLs: all entries in the preference settings 'Archive Data Server URL's' and 'Default Archive Data Sources' in *Edit* → *Preferences* → *CSS Applications* → *Trends* → *Data Browser* and also the entry in the 'Address List' field in *Edit* → *Preferences* → *CSS Core* → *Data Sources* → *Channel Access*. Then CS-Studio needs to be restarted.

2.4.1 Configuring CS-Studio to reach the EPICS IOCs

Even though EPICS has built-in means of discovering IOCs on the local network, we advise to add all IPs of the EPICS IOCs manually in the 'address list' filed in the preference pane found under the path *Edit* → *Preferences* → *CSS Core* → *Data Sources* → *Channel Access*.

2.4.2 Creating Individual Operator Interfaces

To build a new Operator Interface (OPI) in CS-Studio, *File* → *new* and select the OPI file. By clicking 'next' (a new folder with the name of the project could be added) an OPI file name will be set.

This way, a blank OPI will be created. The window style could be changed under *Window* →*Open Perspective*. In default perspective is 'CS-Studio' and splits the window in three sections. In the left section is the navigation, where the file system tree is shown and with right click the kind for opening can be chosen (e.g. OPI editor or OPI display). The editor window is in the middle. All tools for the design of the OPIs are on the righthandside of this middle section. In the right section the 'Properties' of all OPI widgets are displayed in a context sensitive manner and also the opened OPI Displays are shown there. If the navigator and the properties windows are not there, under *Window* → *Show View* they can be activated. Figure 11 shows how the editor for a full OPI looks like.

It's useful to create the GUI like a pyramid with the top of the pyramid being a 'top-level' OPI including further 'sub' OPIs allow including other OPIs via so-called 'Linking Container's. It is possible to work with macros to safe time and storage like in the .db file for the IOCs. The basement are the OPIs for every PV or 'Grouping Container' which includes a variety of PVs. The used monitors, controls and other widgets of the OPI will be linked with the EPICS-PV via the properties field called 'PV Name'. The attached PV name have to fill in there. If PVs or a 'Grouping Container' have to look or work like others, than it is useful to use macros in the 'PV Name' and also 'Name' field. Figure 9 shows the OPI of a 'Grouping Container' with macros for a channel of Hameg HMP 4030. The macros have the same syntax as the macros in the .db file, e.g. $(macroname). A level higher are the OPIs for every IOC. These OPI includes via the editor tool 'Linking Container' all PV OPIs of the IOC. In the properties of the 'Linking Container' the macros in the 'Macros' field could replaced by {Parent Macros}{macroname=test} (shown in figure 10).

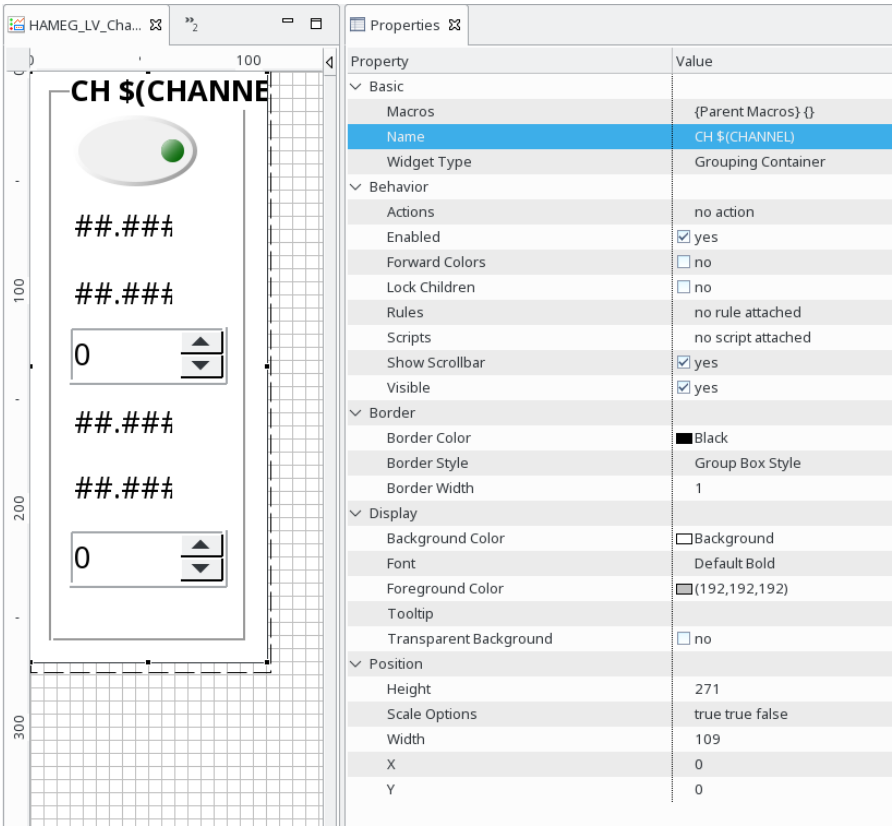


Figure 9: OPI representing a single channel of the Hameg HMP4030 low-voltage power supply. It is framed by a so-called 'Grouping Container' indicating all widgets inside belong together, here to a single power supply channel. This OPI was written to be included by others as the 'macro' $(CHANNEL) has to be substituted. It appears for example in the title of the 'Grouping Container'.

At the highest level is the 'top-level' OPI where all IOC OPIs included. The following shows our first PRESTO Control Operatormode:

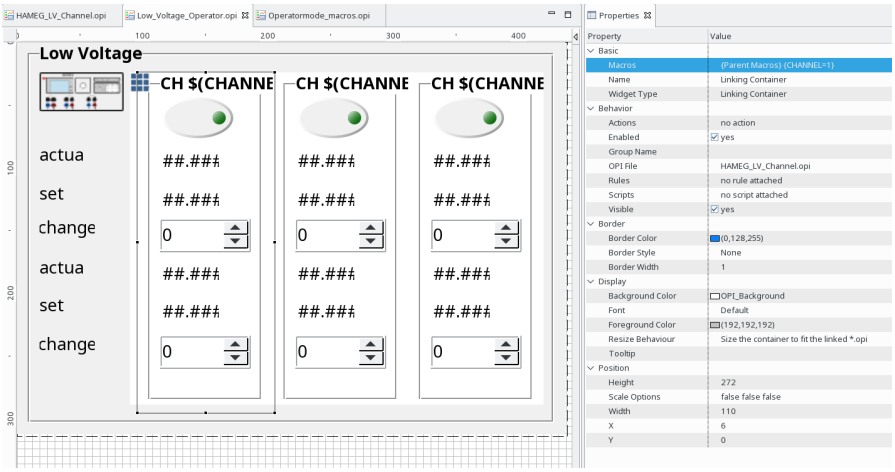


Figure 10: OPI showing how the lower-level single-channel OPI shown in figure 9 is included three times in 'Linking Containers'. For each of them, the 'macro' $(CHANNEL) is substituted with a different value.

The global GUI of our system comprises three control modes:

- **PRESTO Control Overview** is used to get a global detector status where the user has only the authority of reading values.
- **PRESTO Control Operator** is where users are authorized to read and write experimental needed values.
- **PRESTO Control Expert** is the mode which allows users to read and write all device specific details.

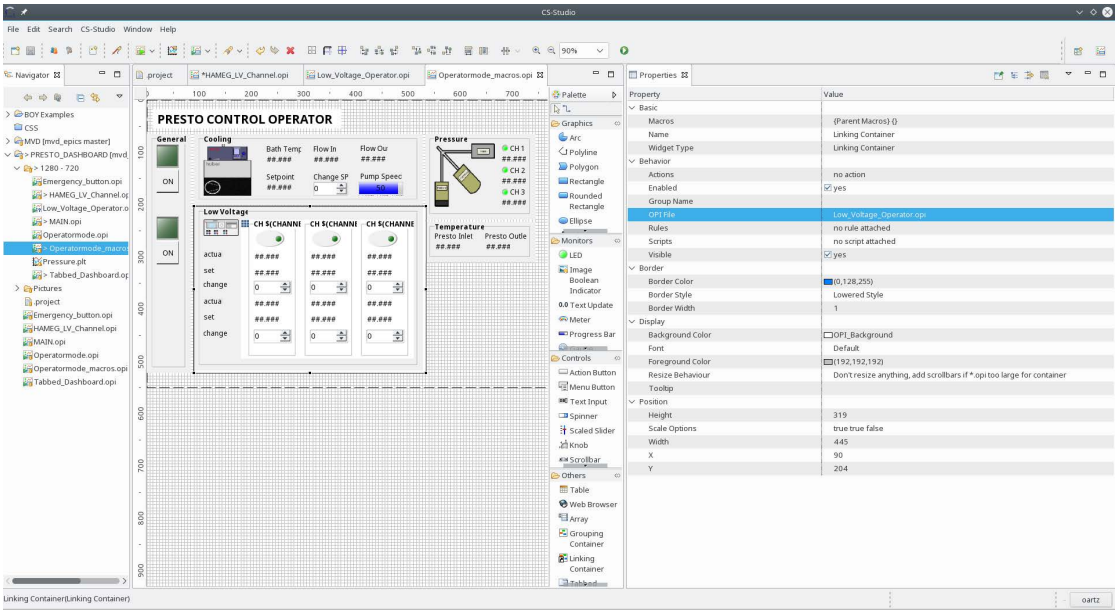


Figure 11: OPI Operatormode with all OPIs if the devices for PRESTO.

The idea is that it is easy to switch the mode from only reading to full control (including write access) of settings in one (global) OPI. But the expert and operator mode should be saved with different passwords (not implemented yet) as a way only give control to operators after sufficient training. For its implementation the global OPI uses 'Tabbed Containers' which includes the different sub-OPIs by using 'Linking Containers', which was explained before.

All modes (also the PRESTO Control Overview) have a centrally placed 'Emergency Button' which shuts down the supply of the sensors. This was done by the properties of the 'Action Button' in the '*Behavior* $\rightarrow$ *Actions*' field. By clicking opens a 'Set Actions' window, where actions can be added with a tabular of properties like 'PV Name', 'Value', 'Timeout (second)' and so on. The PV will set to the 'Value' by clicking the 'Action Button' and the values of the PV can reset after the 'Timeout' entry.

2.4.3 Using the Archiver in CS-Studio

CS-Studio is able to show plots of the archiver data which are used for trend plots of PVs and investigate issues. CS-Studio comes with full support for the RDB archiver engine introduced in 2.2. Only the archiver database connection needs to be configured. This is done in the known preference settings 'Archive Data Server URL's' and 'Default Archive Data Sources' in *Edit* → *Preferences* → *CSS Applications* → *Data Browser*. In the 'Archiver Data Server URLs' settings the 'URL' of the archiver and a 'Server Alias' are needed. With the settings of the archiver in chapter 2.2.2 on page 31 the url looks like `jdbc:postgresql` ↪ `://cbmmvd_pgsql_server:5432/archive` and for the 'Server Alias' entry we enter the internal name of the computer hosting the database, in our case 'jspc57'. The next table 'Default Archive Data Sources' needs the following entries: 'jspc57_def' for 'Name', '1' for 'Key' and in 'URL' the same entry as the table before. In ... → *CSS Applications* → *RDB Archive* are the reader settings, where the user with its password are deposited. The 'Database Schema' and 'Stored procedure' fields have to be empty.

A new plot file (.plt) would by choosing *CS-Studio* →*Trends*→ *Data Browser*. Adding PVs to the plot would be done by right click in 'Traces' properties. The color, choice of axis, and more can be changed in the Trace properties. If the time of date range is large or data was frequently archived, the plot of its time-series could potentiality contain a huge amount of data points. By default, CS-Studio offloads the task of averaging the data and creating minimum - maximum bands to the database server. If it is desired to see the full raw data (=every single data point), the 'Request' entry has to be changed to 'Raw data'. In figure 12 such a plot here with the 'optimized' mode of correlated temperatures is shown. It illustrates how data stored in the archiver database can be visualized with CS-Studio.

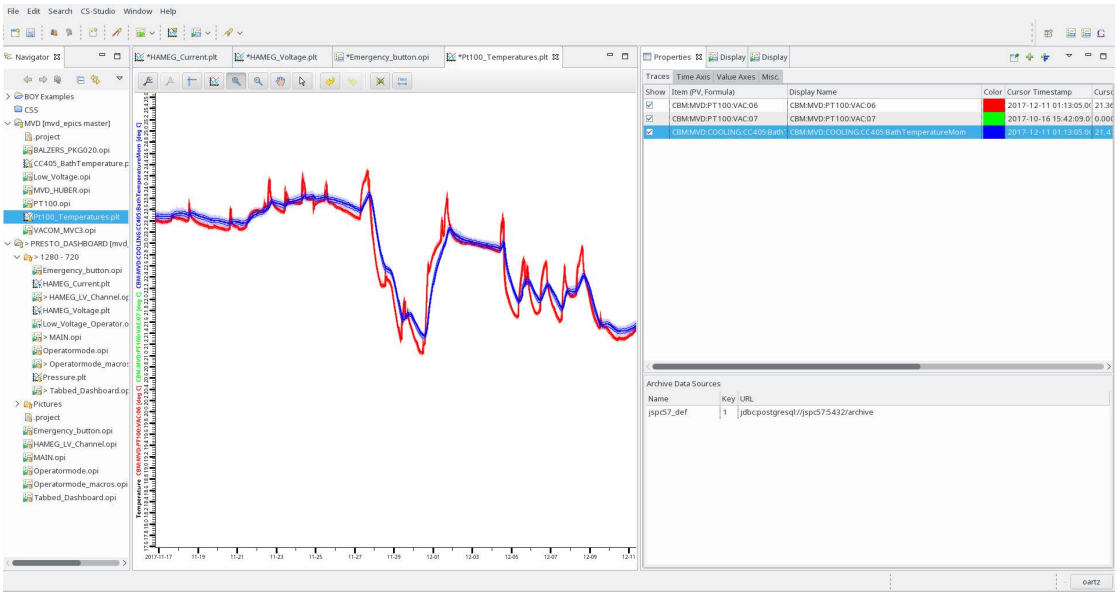


Figure 12: CS-Studio plot showing historic values stored in the archiver database. Here, the optimized mode is used resulting in a smooth (averaged) line within a min-max band. As evident from the plot the two shown temperatures are strongly correlated.

2.5 Alarm-Handling

What are alarms and what is alarming? Alarms indicate that something is not normal. E.g. fire alarms have the function to indicate fire. The fire alarm is composed of alarm ringing bells and flashing (fire-) alarm lights. The fire alarm was triggered by a smoke detector or a fire alarm button. Some fire alarms are directly connected to the fire brigade and with the fire sprinklers. Finally, alarms react by a trigger, briefing the affected people by audio and visual signals and share the new status with experts. So, complex systems need a alarm- and alarm-handling system.

On the **EPICS-side** there exists a kind of alarm-handling. In principle every PV has one of the following four severities: 'NO_ALARM', 'MINOR', 'MAJOR' and 'INVALID' and in addition on the client side (e.g. CS-Studio) 'DISCONNECTED' if its does not find PV in network. The ranges of 'MINOR' and 'MAJOR' alarm are defined in the .db file with the following 'fieldtypes':

fieldtype	function	fieldtypevalue example	severity
HIGH	High Alarm Limit	30.0	MINOR
HIHI	High High Alarm Limit	35.0	MAJOR
LOW	Low Alarm Limit	-20.0	MINOR
LOLO	Low Low Alarm Limit	-25.0	MAJOR

Table 4: EPICS severities for alarm-handling

The 'INVALID' state emerges if the device's sent message differs from the 'in' of the IOC-'.proto' file. If the device is connected and there are no alarm states like 'MINOR', 'MAJOR' or 'INVALID' then the severity is 'NO_ALARM'.

While the EPICS'severities 'MINOR', 'MAJOR', 'INVALID' (and 'DISCONNECTED') can be considered to be EPICS implementation of alarm states, they are not yet sufficient in their basic form to build an advanced DCS. Alarm states that span across multiple devices are an important stop further (Inter-IOCs). To set-up an autonomous control system, finite state machines play an important role to react to those alarm states and to bring the system back in a safe operational mode in such an event.

Running experiments in autonomous mode needing full alarm handling which is more complex than the EPICS severities. It contains alarm propagations with different finite state machines (FSM) and Inter-IOC alarms. Also needed, a kind of

silencing alarms if the cause of the alarm can not be fixed easily or a local malfunction is not relevant for the global detector status e.g. one of many thermometers has an issue and sends a non realistic value and the experiment is still running.

Beside the EPICS internal alarm handling should the operator get a graphical (in CS-Studio) and acoustical (f.g. a ringing bell) alarm. For involving operators, which are not in the control office, some (mobile) alarm messages via f.g. SMS, Mail or Push Messages on smartphone are needed.

On client-side, **CS-Studio** also provides some basic tools. In the properties of every widget representing a PV are fields for the border settings such as an 'alarm sensitive' setting (yes/no, default: yes). If the PV severity is something other than 'NO_ALARM', the border of the widget would be visible in red for 'MAJOR', orange for 'MINOR' and magenta for 'INVALID' and also for 'DISCONNECTED'. Another display setting concerning alarm features is called 'Alarm Pulsing' (yes/no, default: no) and 'ForeColor Alarm Sensitive' (no/yes, default: no). The PV widget like 'Meter', 'Progress Bar', 'Gauge' and 'Thermometer' show with colors whether the current value is within the bands specified 'LOLO', 'LOW', 'HIGH' and 'HIHI' which by default translates into 'MINOR' r 'MAJOR' severity states if the value is beyond those limits. Continuously in the aforementioned colors ('NO_ALARM' equal 'OK' colored in green). Therefore, CS-Studio uses the IOC-.db file fieldtypes in previously shown table 4 and additional the upper and lower display limits from the fieldtypes 'HOPR' and 'LOPR' (compare with tabular in chapter 2.1.2). Alternatively, this values could be set in the editor tool properties.

More detailed alarm-handling in CS-Studio brings the SNS (one of the CS-Studio developers) created distributed alarm system called Best Ever Alarm System Toolkit (BEAST) which currently is not implemented in PRESTO DCS. BEAST has features like a live-data-supported alarm server, CS-Studio user interface as a table or hierarchical tree for viewing current alarms, a relational database for configuration and logging and web reports for the alarm server status. These features allows operators to "access guidance on how to handle a specific alarm, invoke links to related operator interfaces or other CS-Studio tools for the alarm trigger PVs, acknowledge alarms and edit the configuration" [39]. More information for installing and configuration can be taken from [40]. In view of the features and the resulting valuable information, PRESTO's DCS should implement BEAST in future.

2.6 Web Dashboard

Recently, PRESTO has a portable/mobile Web Dashboard serving as a monitoring interface (vs. control interface) as it was decided that data would only be readable. This highly configurable web dashboard for EPICS PVs is an in-house development at the Institut für Kernphysik, Frankfurt and running in any modern browser [22].

For a detector status beyond the control room, a kind of mobile GUI is desirable. For PRESTO, a web GUI was considered, that was going away from OPIs, giving way to the freedom of modern web technology, allowing to use animations, svg-graphics, Javascript frameworks like d3 for plotting (e.g. sparklines) and a responsive design for small and large screens. The density of information should be the same as in the CS-Studio OPIs. If this Web GUI runs only in a secured/closed network then the option to write values by providing a password as in the operator views depending on the user's role in CS-Studio (compare with chapter 2.4.2 on page 39) could be considered. But this Web GUI would then have to be restricted to a secured/closed network and not completely mobile.

The navigation bar consists of three drop-down menus: 'Home', 'PV Overviews' and 'GViews' which stands for graphical views. The 'Home' page is linked to the the first drop-down menu option of 'PV Overviews' called 'General System Overview' which is comparable with the OPI 'PRESTO Control Overview' where only selected PV for a global detector status are shown (see in chapter 2.4.2). In figure 13 a cropped version of the 'General System Overview' page of the PRESTO Web Dashboard is shown.

The other entries of 'PV Overviews' show the different subsystems like cooling, vacuum and low voltage in detail. On those pages, all readable PVs of the subsystem are accessible. And the final drop-down menu 'GViews' shows by choosing one of the following options 'PRESTO Cooling System (schematic)' and 'PRESTO Cooling & Vacuum System (pictorial)' (see figure 14) a dynamic picture with live data. Both are based in SVG [3] images, to form interactive visualizations showing the detector's data in real-time. A heartbeat indicates the live-status. The 'PRESTO Cooling System (schematic)' feature an animation of the cooling flux controlled by the PV value of the cooling pump.

On the PV Overview, every PV has its own box (in figure 13 shown) which contains an alias name like 'Set Voltage (Channel 1)', live status of their value, also

[3] SVG standing for the Scalable Vector Graphics standard which is a XML-based vector image format for two-dimensional graphics.

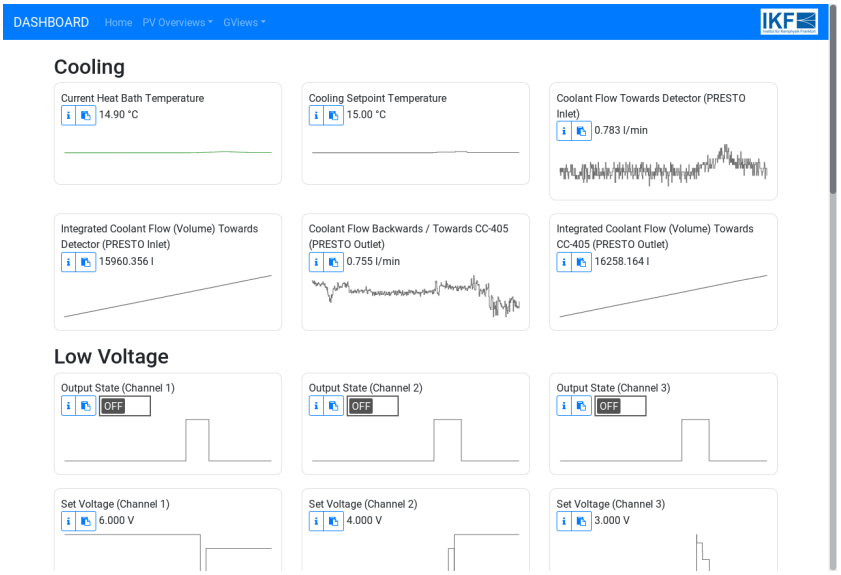


Figure 13: General System Overview page of the PRESTO Web Dashboard GUI (presto.site)

with a 30-minute sparkline visualizing of their evolution [4] and two buttons: information ('information button' marked with the letter 'i') and copy (copy symbol) [22]. Under the cursor the 'i' shows the full PV name like `CBM:MVD:POWER:` ↪ `PRESTO_RECIPIENT:CHAN1:GetOutput` and by clicking 'i' information about the settings in the corresponding records are shown. The button with the copy symbol allows to 'Copy the PV name to the clipboard'.

[4] Currently, the sparklines got their data from web cache which means by every new call the sparklines plot only the actual value. In future, the 30-minute sparklines should fit with the last 30-minute written archiver data to got a full 30-minute sparkline by every web start.

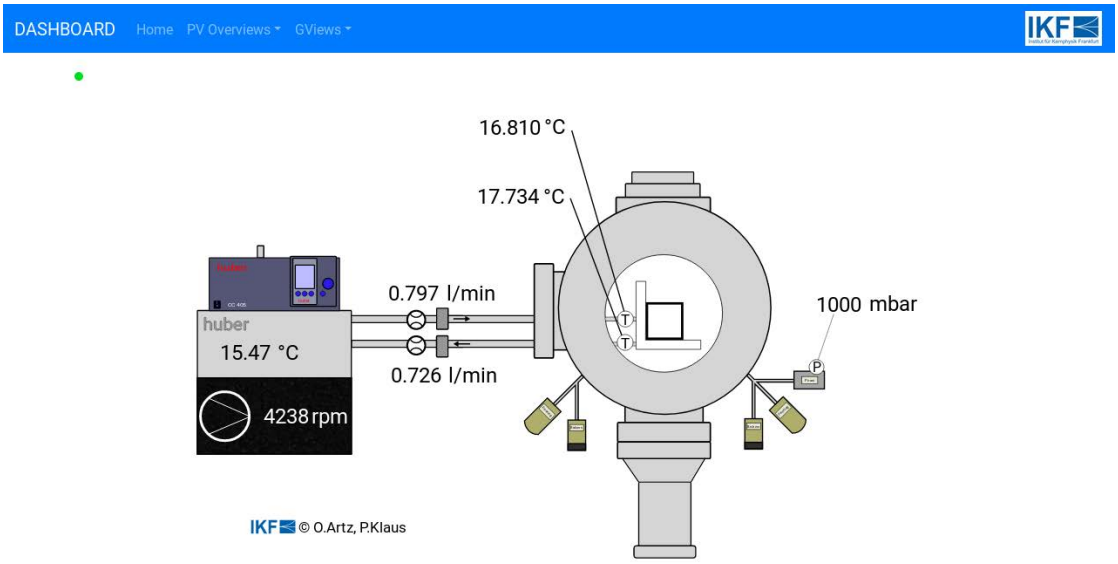


Figure 14: PRESTO Cooling & Vacuum System (pictorial) page of the PRESTO Web Dashboard GUI (https://presto.site)

2.7 Review of the progress achieved within this thesis

Within this thesis different components of the DCS for PRESTO were implemented. In the following figure 15, the implemented IOCs, the Archiver and the Web-GUI are plotted against their date of the first working version.

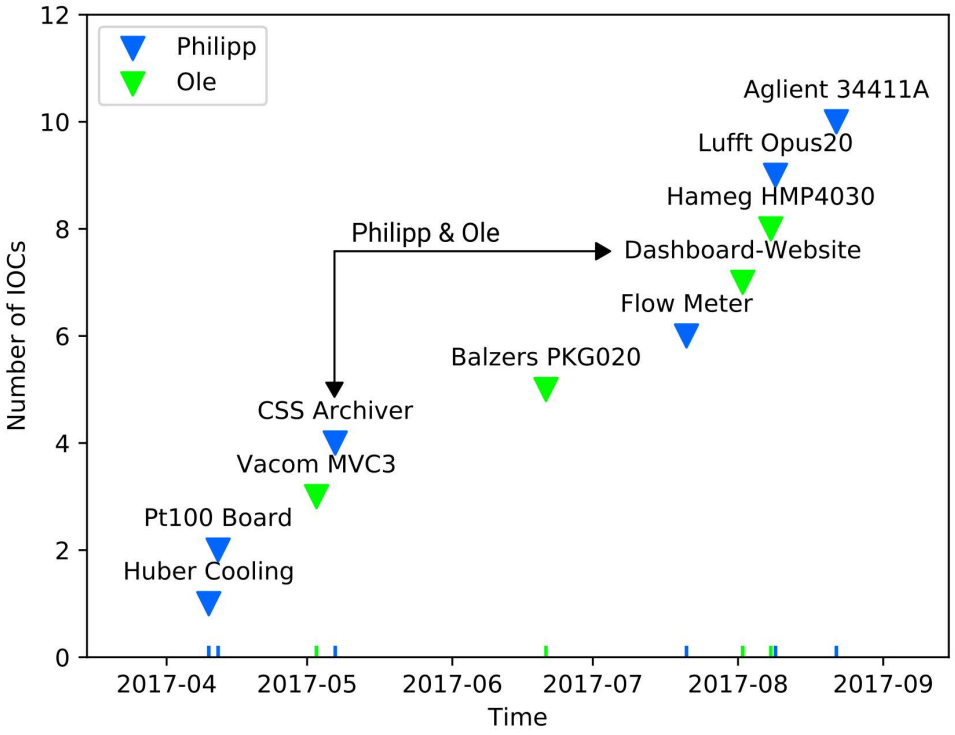


Figure 15: Timeline of implemented IOCs, CSS Archiver and Dashboard-Website. The implementation of the CSS Archiver and Dashboard-Website were the result of a joint effort between Philipp Klaus and Ole Artz.

3 Summary and Outlook

The implementation of running EPICS as our DCS for PRESTO comprises installing and configuring EPICS, thereby, the understanding of the EPICS architecture and also the installation and configuration of Input/Output Controllers for all PRESTO relevant devices and the knowledge of the protocols associated with those devices. Also, installing and configuring EPICS clients like CS-Studio, RDB Archiver and the Web Dashboard. All of these were described in this thesis.

To operate PRESTO in an autonomous mode in the future, the alarm manager called Best Ever Alarm System Toolkit (BEAST), alarms across devices, and the implementation of Finite State Machines (FSMs) need to complement the components described in this thesis. The sparklines of the Web Dashboard could be fed by data stored in the archiver database. Also, adding missing systems (e.g. low voltage) should be added to the graphical views in the web dashboard.

After the implementation of all theses steps, a powerful tool will be at hand to operate the detector autonomously after going through thorough evaluations.

References

[1] FAIR GmbH. http://www.fair-center.eu/for-users/experiments/nuclear-matter-physics.html.

[2] Peter Senger. Cbm school: Cosmic matter on dense baryonic matter. page 13, September 2017.

[3] GSI Helmholtzzentrum für Schwerionenforschung GmbH. https://www.gsi.de/en/researchaccelerators/fair.

[4] Bengt Friman, Claudia Höhne, Jörn Knoll, Stefan Leupold, Jorgen Randrup, Ralf Rapp, and Peter Senger. *The CBM physics book: Compressed baryonic matter in laboratory experiments*, volume 814. Springer, 2011.

[5] INPAC. Nuclear physics under extreme conditions. September 2012.

[6] Pavel Larionov and CBM Collaboration. Overview of the silicon tracking system for the cbm experiment. *Journal of Physics: Conference Series*, 599(1):012025, 2015.

[7] G Agakishiev, O Arnold, A Balanda, D Belver, A Belyaev, JC Berger-Chen, A Blanco, M Böhmer, JL Boyard, P Cabanelas, et al. Statistical model analysis of hadron yields in proton-nucleus and heavy-ion collisions at sis 18 energies. *arXiv preprint arXiv:1512.07070*, 2015.

[8] http://www.fair-center.eu/fileadmin/fair/experiments/CBM/setup/2016.03.14_CBM_Hades_SIS_100.jpg.

[9] P.Klaus. private communication.

[10] Tomáš Balog. Overview of the cbm detector system. In *Journal of Physics: Conference Series*, volume 503, page 012019. IOP Publishing, 2014.

[11] Qiyan Li. Cbm-mvd collaboration meeting. March 2017.

[12] Bengt Friman, Claudia Höhne, Jörn Knoll, Stefan Leupold, Jørgen Randrup, Ralf Rapp, and Peter Senger. General introduction. *The CBM Physics Book*, pages 623–630, 2011.

[13] P Klaus, M Wiebusch, S Amar-Youcef, M Deveaux, M Koziel, J Michel, B Milanovic, C Müntz, T Tischler, and J Stroth. Prototyping the read-out chain of the cbm microvertex detector. *Journal of Instrumentation*, 11(03):C03046, 2016.

[14] Tobias Tischler. *Mechanical Integration of the Micro Vertex Detector for the CBM Experiment*. PhD thesis, Master's thesis, Goethe-Universität Frankfurt, 2015. 19, 2015.

[15] http://www.datatec.de/Hameg-HMP-4030-Stromversorgung.htm.

[16] https://www.huber-online.com/de/product_datasheet.aspx?no=2017.0001.01.

[17] https://www.kobold.com/uploads/files/dpm-gb-flow.pdf.

[18] T.Kühn VACOM. Mvc-3 vacuum gauge controller instruction manual. 2011.

[19] Balzers. Betriebsanweisung pirani-kaltkathoden-vakuummeter pkg 020. 1986.

[20] https://www.lufft.com/products/in-room-measurements-291/opus-20-thip-1983/productAction/outputAsPdf/.

[21] Philipp Klaus. Control systems and protocols or epics for the cbm-mvd & presto. November 2017.

[22] Cbm progress report 2017. March 2018.

[23] https://epics.anl.gov/about.php.

[24] https://epics.anl.gov/neat.php.

[25] L.R. Dalesio, A.J. Kozubal, and M.R. Kraimer. Epics architecture. 1 1991.

[26] https://prjemian.github.io/epicspi/#starting.

[27] http://epics.web.psi.ch/software/streamdevice/doc/stream.pdf.

[28] https://epics.anl.gov/modules/soft/asyn/.

[29] https://www.aps.anl.gov/BCDA/synApps.

[30] Tobias Triffterer. Entwicklung von komponenten für das detektorsteuersystem des panda-kalorimeters und studien zur photoproduktion angeregter η-mesonen mit dem cb/elsa-experiment. 2016.

[31] https://epics.anl.gov/base/R3-14/10-docs/CAref.html.

[32] https://epics.anl.gov./EpicsDocumentation/AppDevManuals/RecordRef/Recordref-1.html.

[33] M.Mitkov P.Zumbruch and P.Klaus. private communication.

[34] https://hades-wiki.gsi.de/foswiki/bin/view/DaqSlowControl/SCSArchiverCSStudio#CS_45Studio_45_RDB_Archiver.

[35] P.Zumbruch and P.Klaus. private communication.

[36] http://css.desy.de/content/index_eng.html.

[37] Kay Kasemir et al. Control system studio applications. *Proceedings of ICALEPCS07, Knoxville, Tennessee, USA*, 2007.

[38] Jan Hatje, Matthias Clausen, Christian Gerke, Markus Moeller, and Helge Rickens. Control system studio (css). 2007.

[39] https://github.com/ControlSystemStudio/cs-studio/wiki/BEAST.

[40] Kay Kasemir, Xihui Chen, and Ekaterina Danilova. The best ever alarm system toolkit. *ICALEPCS09, Kobe Japen*, 2009.

Declaration - Selbstständigkeitserklärung

Erklärung nach § 30 (12) Ordnung für den Bachelor- und dem Masterstudiengang

Hiermit erkläre ich, dass ich die Arbeit selbständig und ohne Benutzung andere als der angegebenen Quellen und Hilfsmittel verfasst habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderen fremden Texten entnommen wurden, sind von mir als solche kenntlich gemacht worden. Ferner erkläre ich, dass die Arbeit nicht - auch nicht auszugsweise - für eine andere Prüfung verwendet wurde.

Ort, Datum	Unterschrift

Acknowledgement - Danksagung

Eine BA-Thesis umfasst nicht nur die Ausarbeitung einer Problemstellung oder die Lösung einer Aufgabe. Vielmehr umfasst die BA-Thesis auch das Erlernen, der dafür benötigten Grundlagen und Kenntnisse. An dieser Stelle möchte ich mich bei allen Bedanken, die dazu beigetragen haben, diese Grundlagen und Kenntnisse zu erlernen und die mich dabei begleitet haben. Insbesondere danke ich Florian Eisenhut für seine ständigen Erklärungen und Bemühungen, die Inhalte im Grundstudium zu verstehen. Und Viviana Macaluso, die mich moralisch unterstützt hat.

Bezogen auf meine BA-Thesis selbst gilt mein Dank an erster Stelle Prof. Dr. Joachim Stroth, der mich in seine Arbeitsgruppe aufgenommen hat und somit ermöglicht hat, meine BA-Thesis bei ihm zu schreiben. Ebenfalls Dr. Christian Müntz, der mir frühe Einblicke in ein laufendes DCS gezeigt hat und direkt damit verbundene Ideen in mir hervor gerufen hat. Schließlich möchte ich mich bei Peter Zumbruch und Martin Mitkov bedanken, die uns bei Hürden beiseite standen und uns geholfen haben, diese zu überwinden. Uns meint Philipp Klaus und mich.

Mein größter Dank gilt Philipp, der Mitauslöser für die Themenwahl meiner BA-Thesis war und der mich schließlich auch betreut hat. Diese Betreuung umfasste eine große Wissensweitergabe und damit verbunden, eine enge und gute Zusammenarbeit.

Von DCS und vor allem Programmierung hatte ich zuvor keine Ahnung, weshalb ich das Thema ausgewählt habe, um viel neues zu erlernen, was ich Dank Philipp erst ermöglicht wurde.

A Appendix

A.1 Table of all PVs of the PRESTO DCS

IOC name	Telnet Port [5]	PV name
iocHUBER_COOLING	20101	CBM:MVD:COOLING:CC405:Verify
		CBM:MVD:COOLING:CC405:Firmware
		CBM:MVD:COOLING:CC405:SetpointMom
		CBM:MVD:COOLING:CC405:BathTemperatureMom
		CBM:MVD:COOLING:CC405:ExternalTemperatureMom
		CBM:MVD:COOLING:CC405:T0:RAW
		CBM:MVD:COOLING:CC405:TC:RAW
		CBM:MVD:COOLING:CC405:T1:RAW
		CBM:MVD:COOLING:CC405:TIntern:RAW
		CBM:MVD:COOLING:CC405:TProcess:RAW
		CBM:MVD:COOLING:CC405:V:RAW
		CBM:MVD:COOLING:CC405:HT:RAW
		CBM:MVD:COOLING:CC405:Heating:RAW
		CBM:MVD:COOLING:CC405:F:RAW
		CBM:MVD:COOLING:CC405:I:RAW
		CBM:MVD:COOLING:CC405:unknown1:RAW
		CBM:MVD:COOLING:CC405:unknown2:RAW
		CBM:MVD:COOLING:CC405:AIF_IN:RAW
		CBM:MVD:COOLING:CC405:TU:RAW
		CBM:MVD:COOLING:CC405:TX1:RAW
		CBM:MVD:COOLING:CC405:TX0:RAW
		CBM:MVD:COOLING:CC405:Pressure:RAW

		CBM:MVD:COOLING:CC405:PumpSpeed:RAW
		CBM:MVD:COOLING:CC405:V
		CBM:MVD:COOLING:CC405:HT
		CBM:MVD:COOLING:CC405:Heating
		CBM:MVD:COOLING:CC405:unknown1
		CBM:MVD:COOLING:CC405:unknown2
		CBM:MVD:COOLING:CC405:F
		CBM:MVD:COOLING:CC405:T0
		CBM:MVD:COOLING:CC405:TC
		CBM:MVD:COOLING:CC405:T1
		CBM:MVD:COOLING:CC405:TIntern
		CBM:MVD:COOLING:CC405:TProcess
		CBM:MVD:COOLING:CC405:TU
		CBM:MVD:COOLING:CC405:TX1
		CBM:MVD:COOLING:CC405:TX0
		CBM:MVD:COOLING:CC405:Pressure
		CBM:MVD:COOLING:CC405:PumpSpeed
		CBM:MVD:COOLING:CC405:I
		CBM:MVD:COOLING:CC405:AIF_IN
		CBM:MVD:COOLING:CC405:State
		CBM:MVD:COOLING:CC405:Setpoint
		CBM:MVD:COOLING:CC405:MinimumValue
		CBM:MVD:COOLING:CC405:MaximumValue
		CBM:MVD:COOLING:CC405:TemperatureControl
iocPT100_BOARD	20102	CBM:MVD:PT100:VAC:00
		CBM:MVD:PT100:VAC:01
		CBM:MVD:PT100:VAC:02
		CBM:MVD:PT100:VAC:03
		CBM:MVD:PT100:VAC:04

		CBM:MVD:PT100:VAC:05
		CBM:MVD:PT100:VAC:06
		CBM:MVD:PT100:VAC:07
		CBM:MVD:PT100:VAC:10
		CBM:MVD:PT100:VAC:11
		CBM:MVD:PT100:VAC:12
		CBM:MVD:PT100:VAC:13
		CBM:MVD:PT100:VAC:14
		CBM:MVD:PT100:VAC:15
		CBM:MVD:PT100:VAC:16
		CBM:MVD:PT100:VAC:17
		CBM:MVD:PT100:VAC:CONV_RATE
		CBM:MVD:PT100:VAC:00:RAW
		CBM:MVD:PT100:VAC:01:RAW
		CBM:MVD:PT100:VAC:02:RAW
		CBM:MVD:PT100:VAC:03:RAW
		CBM:MVD:PT100:VAC:04:RAW
		CBM:MVD:PT100:VAC:05:RAW
		CBM:MVD:PT100:VAC:06:RAW
		CBM:MVD:PT100:VAC:07:RAW
		CBM:MVD:PT100:VAC:10:RAW
		CBM:MVD:PT100:VAC:11:RAW
		CBM:MVD:PT100:VAC:12:RAW
		CBM:MVD:PT100:VAC:13:RAW
		CBM:MVD:PT100:VAC:14:RAW
		CBM:MVD:PT100:VAC:15:RAW
		CBM:MVD:PT100:VAC:16:RAW
		CBM:MVD:PT100:VAC:17:RAW
iocVACOM_MVC3	20103	CBM:MVD:VACUUM:RECIPIENT:1:Pressure

		CCBM:MVD:VACUUM:RECIPIENT:2:Pressure
		CCBM:MVD:VACUUM:RECIPIENT:3:Pressure
		CCBM:MVD:VACUUM:RECIPIENT:1:Status
		CCBM:MVD:VACUUM:RECIPIENT:2:Status
		CBM:MVD:VACUUM:RECIPIENT:3:Status
	iocBALZERS_PKG020	20104 CBM:MVD:VACUUM:PRESTO_RECIPIENT:SIDE_A:TPR2:Voltage
		CBM:MVD:VACUUM:PRESTO_RECIPIENT:SIDE_A:IKR:Voltage
		CBM:MVD:VACUUM:PRESTO_RECIPIENT:SIDE_B:TPR2:Voltage
		CBM:MVD:VACUUM:PRESTO_RECIPIENT:SIDE_B:IKR:Voltage
	iocFLOW_METER	20105 CBM:MVD:COOLING:CC405:FLOW:IN:Flow
		CBM:MVD:COOLING:CC405:FLOW:IN:Total
		CBM:MVD:COOLING:CC405:FLOW:OUT:Flow
		CBM:MVD:COOLING:CC405:FLOW:OUT:Total
	iocHAMEG_HMP4030	20106 CBM:MVD:POWER:PRESTO_RECIPIENT:SetGeneralOutput
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:SetOutput
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:SetFuse
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:ClearVoltageProtection
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:SetOutput
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:SetFuse
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:ClearVoltageProtection
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:SetOutput
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:SetFuse
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:ClearVoltageProtection
		CBM:MVD:POWER:PRESTO_RECIPIENT:GetGeneralOutput
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOutput
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetFuse
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetOutput
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetFuse
		CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetOutput

CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetFuse
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:SetOVPMode
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:SetOVPMode
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:SetOVPMode
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:MeasuredVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetVoltageStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetCurrent
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:MeasuredCurrent
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetCurrentStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOVPVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOVPVoltageMin
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOVPVoltageMax
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:MeasuredVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetVoltageStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetCurrent
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:MeasuredCurrent
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetCurrentStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetOVPVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetOVPVoltageMin
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetOVPVoltageMax
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:MeasuredVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetVoltageStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetCurrent
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:MeasuredCurrent
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetCurrentStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetOVPVoltage

CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetOVPVoltageMin
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetOVPVoltageMax
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:GetOVPMode
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:GetOVPMode
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:GetOVPMode
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:SetVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:SetVoltageStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:SetCurrent
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:SetCurrentStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN1:SetOVPVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:SetVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:SetVoltageStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:SetCurrent
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:SetCurrentStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN2:SetOVPVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:SetVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:SetVoltageStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:SetCurrent
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:SetCurrentStepSize
CBM:MVD:POWER:PRESTO_RECIPIENT:CHAN3:SetOVPVoltage
CBM:MVD:POWER:PRESTO_RECIPIENT:Identification

Table 5: Table of all PRESTO-DCS included PVs sort by their IOCs

[5]'Telnet Port' denotes the port on the computer running the IOC where the procServ command (see on page 34) is offering the shell to the parent IOC via Telnet.

A.2 Archiver setup and config files

pluginCustomization_dcs_engine.ini

```
org.csstudio.archive.rdb/url=jdbc:postgresql://
   ↪ cbmmvd_pgsql_server:5432/archive
org.csstudio.archive.rdb/user=archive
org.csstudio.archive.rdb/schema=
org.csstudio.security/secure_preference_location=Instance
org.csstudio.platform.libs.epics/addr_list=192.168.10.101
   ↪ 192.168.10.102 192.168.10.103 192.168.10.104
   ↪ 192.168.10.105 192.168.10.106 192.168.10.107
   ↪ 192.168.10.108
```

postgres_schema.sql

```
/*
 * These commands create a new PostGres SQL database for
    ↪ archiving.
 *
 * THIS WILL DELETE ANY DATA THAT MIGHT BE IN THERE!
 *
 * The Oracle schema should be similar, except:
 * - Exact data types can vary (length of strings etc.)
 * - Oracle TIMESTAMP already includes the nanosecond detail,
    ↪ so
 *   no separate "nanosecs" column is required
 * - Support for partitioning
 *
 * Test setup:
 * postgres 8.1.20 on a Linux Laptop
 * (1.6 GHz, disk access hampered by pointsec)
 * and INSERTs performed from a Java program
 * on a 1.3 GHz OS X PowerBook G4
 *
 * Result of 10 min test run with MyISAM (no referential
    ↪ integrity check)
 * About 2500 inserts per second,
 * using ~33 bytes per value
 * (overall database size / pure sample count).
 * accl2 (about 20% CPU used by other people) -> ics-srv-
    ↪ softioc4: 1900 / sec
 * fe-ics-opi1 (otherwise ~idle CPU) -> ics-srv-softioc4: 3400
```

```
    ↪ / sec
 * local, all on ics-srv-softioc4: 11000 / sec
 *
 * Result of 10 min test run with InnoDB (check sample.
    ↪ channel_id)
 * About 750 inserts per second.
 * Byte/sample unclear because not all data in a mysql/data
    ↪ subdir?
 * accl2 -> ics-srv-softioc4: 1200 / sec
 * local, all on ics-srv-softioc4: 3500 / sec
 */

-- Create 'archive' user who can remotely access the 'archive'
   ↪ tables,
-- but only change the table layout locally
--
-- Assume you are connected as the 'postgres' super user

/*
CREATE USER archive WITH PASSWORD '$archive';
ALTER USER archive WITH PASSWORD '$archive';

CREATE USER report WITH PASSWORD '$report';

SELECT * FROM pg_user;

-- The following would have to be executed _after_ creating the
   ↪  tables:
GRANT SELECT, INSERT, UPDATE, DELETE
  ON smpl_eng, retent, smpl_mode, chan_grp, channel, status,
     ↪ severity, sample, array_val, num_metadata,
     ↪ enum_metadata
  TO archive;

GRANT SELECT
  ON smpl_eng, retent, smpl_mode, chan_grp, channel, status,
     ↪ severity, sample, array_val, num_metadata,
     ↪ enum_metadata
  TO report;

-- Might have to check with \d which sequences were
-- created by Postgres to handle the SERIAL columns:
GRANT USAGE ON SEQUENCE
```

```
  chan_grp_grpid_seq, channel_chid, retent_retentid_seq,
  severity_sevid, smpl_eng_engid_seq, status_statid
  TO archive;
*/

------------------------



DROP DATABASE IF EXISTS archive;

CREATE DATABASE archive;

\connect archive

DROP SCHEMA archive CASCADE;
CREATE SCHEMA archive;
set search_path to 'archive';

------------------------
-- Sample engine
CREATE SEQUENCE smpl_eng_engid_seq;

DROP TABLE IF EXISTS archive_schema;
CREATE TABLE archive_schema(
   id SERIAL PRIMARY KEY,
   version VARCHAR(10)
);

DROP TABLE IF EXISTS smpl_eng;
CREATE TABLE  smpl_eng
(
   eng_id BIGINT NOT NULL PRIMARY KEY DEFAULT nextval('
      ↪ smpl_eng_engid_seq'),
   name VARCHAR(100) NOT NULL,
   descr VARCHAR(100) NOT NULL,
   url VARCHAR(100) NOT NULL
);
INSERT INTO smpl_eng VALUES (1, 'Demo', 'Demo␣Engine', 'http://
   ↪ localhost:4812');
SELECT * FROM smpl_eng;

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

```
-- Retention
-- Not used at this time
CREATE SEQUENCE retent_retentid_seq;

DROP TABLE IF EXISTS retent;
CREATE TABLE  retent
(
   retent_id BIGINT NOT NULL PRIMARY KEY DEFAULT nextval('
      ↪ retent_retentid_seq'),
   descr VARCHAR(255) NOT NULL
);
INSERT INTO retent VALUES (30, 'Months');
INSERT INTO retent VALUES (9999, 'Forever');
SELECT * FROM retent;

------------------------
-- Channel Group

CREATE SEQUENCE chan_grp_grpid_seq;

DROP TABLE IF EXISTS chan_grp;
CREATE TABLE  chan_grp
(
   grp_id BIGINT NOT NULL PRIMARY KEY DEFAULT nextval('
      ↪ chan_grp_grpid_seq'),
   name VARCHAR(100) NOT NULL,
   eng_id BIGINT NOT NULL,
   descr VARCHAR(100) NULL,
   enabling_chan_id BIGINT NULL
);
INSERT INTO chan_grp VALUES (1, 'Demo', 1, 'Demo␣Group', NULL);
SELECT * FROM chan_grp;

------------------------
-- Sample modes
DROP TABLE IF EXISTS smpl_mode;
CREATE TABLE  smpl_mode
(
   smpl_mode_id BIGINT NOT NULL PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   descr VARCHAR(100) NOT NULL
);
INSERT INTO smpl_mode VALUES (1, 'Monitor', 'Store␣every␣
```

```
   ↪ received␣update');
INSERT INTO smpl_mode VALUES (2, 'Scan', 'Periodic␣scan');
SELECT * FROM smpl_mode;

------------------------
-- Channel: ID and name
CREATE SEQUENCE channel_chid;

DROP TABLE IF EXISTS channel;
CREATE TABLE  channel
(
   channel_id BIGINT NOT NULL PRIMARY KEY DEFAULT nextval('
      ↪ channel_chid'),
   name VARCHAR(100) NOT NULL,
   descr VARCHAR(100) NULL,
   grp_id BIGINT NULL,
   smpl_mode_id BIGINT NULL,
   smpl_val double precision NULL,
   smpl_per double precision NULL,
   retent_id BIGINT NULL,
   retent_val DOUBLE precision NULL
);
INSERT INTO channel(channel_id, name) VALUES (1, 'sim://sine(0,
   ↪ ␣10,␣50,␣0.1)');
INSERT INTO channel(channel_id, name) VALUES (2, 'sim://
   ↪ noiseWaveform(0,10,100,10)');
INSERT INTO channel(channel_id, name) VALUES (3, 'freddy');
INSERT INTO channel(channel_id, name) VALUES (4, 'jane');
UPDATE channel SET retent_val=9999 WHERE channel_id < 4;
UPDATE channel SET grp_id=1 WHERE channel_id < 4;
UPDATE channel SET smpl_val=1 WHERE channel_id = 1;
SELECT * FROM channel;

------------------------
-- Severity mapping of severity ID to string
CREATE SEQUENCE severity_sevid;

DROP TABLE IF EXISTS severity;
CREATE TABLE severity
(
   severity_id BIGINT NOT NULL PRIMARY KEY default nextval('
      ↪ severity_sevid'),
   name VARCHAR(100) NOT NULL
```

```
);
INSERT INTO severity VALUES (1, 'OK'), (2, 'MINOR'), (3, 'MAJOR
   ↪ '), (4, 'INVALID');
SELECT * FROM severity;

------------------------
-- Status mapping of status ID to string
create sequence status_statid;
DROP TABLE IF EXISTS status;
CREATE TABLE  status
(
   status_id BIGINT PRIMARY KEY default nextval('status_statid'
      ↪ ),
   name VARCHAR(100) NOT NULL UNIQUE
);
INSERT INTO status (name) VALUES ('OK'), ('disconnected');
SELECT * FROM status;

------------------------
-- Samples of a channel
-- Either the numeric, floating point or string value should be
   ↪  set,
-- not all of them.
--
-- See array_encoding.txt for handling of array data.
DROP TABLE IF EXISTS sample;
CREATE TABLE sample
(
   channel_id BIGINT NOT NULL,
   smpl_time TIMESTAMP NOT NULL,
   nanosecs BIGINT  NOT NULL,
   severity_id BIGINT NOT NULL,
   status_id BIGINT  NOT NULL,
   num_val INT NULL,
   float_val double precision NULL,
   str_val VARCHAR(120) NULL,
   datatype CHAR(1) NULL DEFAULT '␣',
   array_val BYTEA  NULL,

   -- Note that these foreign keys are good for data
      ↪ consistency,
   -- but bad for performance.
   -- Writing to the table will be almost twice as fast without
```

```
    -- the following constraints
    FOREIGN KEY (channel_id) REFERENCES channel (channel_id) ON
       ↪ DELETE CASCADE,
    FOREIGN KEY (severity_id) REFERENCES severity (severity_id)
       ↪ ON DELETE CASCADE,
    FOREIGN KEY (status_id) REFERENCES status (status_id) ON
       ↪ DELETE CASCADE
);

-- Need index on channel_id and smpl_time?
CREATE INDEX sample_id_time ON sample ( channel_id, smpl_time,
    ↪ nanosecs );

-- These inserts are in reverse time order to check retrieval
INSERT INTO sample (channel_id, smpl_time,  nanosecs,
    ↪ severity_id, status_id, float_val)
    VALUES (1, '2004-01-10␣13:01:17', 1,  3, 2, 3.16),
           (1, '2004-01-10␣13:01:11', 2,  1, 1, 3.16),
           (1, '2004-01-10␣13:01:10', 3, 1, 2, 3.15),
           (1, '2004-01-10␣13:01:10', 4, 1, 2, 3.14);

------------------------
-- *** OLD Array element table. Replaced by array_val BLOB in
    ↪ sample table ***
-- See sample table: Array elements 1, 2, 3, ... beyond the
    ↪ element 0
-- that's in the sample table
DROP TABLE IF EXISTS array_val;
CREATE TABLE  array_val
(
    channel_id BIGINT  NOT NULL,
    smpl_time TIMESTAMP NOT NULL,
    nanosecs BIGINT  NOT NULL,
    seq_nbr BIGINT  NOT NULL,
    float_val double precision NULL,
    FOREIGN KEY (channel_id) REFERENCES channel (channel_id) ON
       ↪ DELETE CASCADE
);

CREATE INDEX array_val_id_time ON array_val ( channel_id,
    ↪ smpl_time, nanosecs );
```

```
------------------------
-- Channel Meta data: Units etc. for numeric channels
DROP TABLE IF EXISTS num_metadata;
CREATE TABLE  num_metadata
(
   channel_id BIGINT  NOT NULL PRIMARY KEY,
   low_disp_rng double precision NULL,
   high_disp_rng double precision NULL,
   low_warn_lmt double precision NULL,
   high_warn_lmt double precision NULL,
   low_alarm_lmt double precision NULL,
   high_alarm_lmt double precision NULL,
   prec INT NULL,
   unit VARCHAR(100) NOT NULL
);
INSERT INTO num_metadata VALUES (1, 0, 10, 2, 8, 1, 9, 2, 'mA')
   ↪ ;
SELECT * FROM num_metadata;

------------------------
-- Enumerated channels have a sample.num_val that can also be
   ↪ interpreted
-- as an enumeration string via this table
DROP TABLE IF EXISTS enum_metadata;
CREATE TABLE enum_metadata
(
   channel_id BIGINT  NOT NULL,
   enum_nbr INT NULL,
   enum_val VARCHAR(120) NULL,
   FOREIGN KEY (channel_id) REFERENCES channel (channel_id) ON
      ↪ DELETE CASCADE
);

------------------------
------------------------
------------------------
-- Dump all values for all channels
SELECT channel.name, smpl_time, severity.name, status.name,
   ↪ float_val
   FROM channel, severity, status, sample
   WHERE channel.channel_id = sample.channel_id AND
         severity.severity_id = sample.severity_id AND
         status.status_id = sample.status_id
```

```
   ORDER BY channel.name, smpl_time
   LIMIT 50;

-- Same with INNER JOIN
SELECT channel.name AS channel,
       smpl_time,
       severity.name AS severity,
       status.name AS status,
       float_val
   FROM sample INNER JOIN channel ON sample.channel_id =
      ↪ channel.channel_id INNER JOIN severity ON sample.
      ↪ severity_id = severity.severity_id INNER JOIN status
   ON sample.status_id = status.status_id
   ORDER BY smpl_time
   LIMIT 50;



CREATE USER archive WITH PASSWORD '$archive';
ALTER USER archive WITH PASSWORD '$archive';

CREATE USER report WITH PASSWORD '$report';

SELECT * FROM pg_user;

-- The following would have to be executed _after_ creating the
   ↪  tables:
GRANT SELECT, INSERT, UPDATE, DELETE
  ON smpl_eng, retent, smpl_mode, chan_grp, channel, status,
     ↪ severity, sample, array_val, num_metadata,
     ↪ enum_metadata
  TO archive;

GRANT SELECT
  ON smpl_eng, retent, smpl_mode, chan_grp, channel, status,
     ↪ severity, sample, array_val, num_metadata,
     ↪ enum_metadata
  TO report;
  GRANT SELECT ON ALL TABLES IN SCHEMA archive TO report;
-- Might have to check with \d which sequences were
-- created by Postgres to handle the SERIAL columns:
GRANT USAGE ON SEQUENCE
```

```
  chan_grp_grpid_seq, channel_chid, retent_retentid_seq,
  severity_sevid, smpl_eng_engid_seq, status_statid
  TO archive;



ALTER TABLE archive_schema OWNER TO archive;
ALTER TABLE array_val OWNER TO archive;
ALTER TABLE chan_grp OWNER TO archive;
ALTER TABLE channel OWNER TO archive;
ALTER TABLE enum_metadata OWNER TO archive;
ALTER TABLE num_metadata OWNER TO archive;
ALTER TABLE retent OWNER TO archive;
ALTER TABLE sample OWNER TO archive;
ALTER TABLE severity OWNER TO archive;
ALTER TABLE smpl_eng OWNER TO archive;
ALTER TABLE smpl_mode OWNER TO archive;
ALTER TABLE status OWNER TO archive;


alter user postgres set search_path to 'archive';
alter user archive set search_path to 'archive';
alter user report set search_path to 'archive';

GRANT ALL ON SCHEMA archive TO archive;
GRANT USAGE ON SCHEMA archive TO report;

 commit;
```

postgres_partitioning.sql

```
                                                -- Provided by Martin
                                                   ↪  Konrad, TU
                                                   ↪ Darmstadt

-- The sample table can grow very fast if you archive a lot of
   ↪ data and with the
-- table size the index size also grows. This results in poor
   ↪ performance if the
-- index doesn't fit into main memory anymore. One way out of
   ↪ this partitioning
-- of the sample table. PostgreSQL supports partitioning but it
   ↪ 's not as easy to
```

```
-- setup as on an Oracle DB.
-- Use the SQL statements in this file to setup partitioning on
   ↪ the sample
-- table. The function provided below automatically creates the
   ↪ necessary
-- sub-tables. Run this function regularly to generate new
   ↪ tables as time moves
-- on.
-- Note: Make sure
-- SHOW constraint_exclusion;
-- returns "partition" otherwise your DB server will waste time
   ↪ scanning
-- sub-tables that do not contain relevant data. Constraint
   ↪ exclusion is enabled
-- for partitioned tables by default in PostgreSQL 8.4 and
   ↪ later.

-- We only create the master table explicitly. The sub-tables
   ↪ are created by the
-- function below as they are needed.
DROP TABLE IF EXISTS archive.sample;
CREATE TABLE archive.sample
(
   channel_id BIGINT NOT NULL,
   smpl_time TIMESTAMP NOT NULL,
   nanosecs BIGINT  NOT NULL,
   severity_id BIGINT NOT NULL,
   status_id BIGINT  NOT NULL,
   num_val INT NULL,
   float_val REAL NULL,
   str_val VARCHAR(120) NULL,
   datatype CHAR(1) NULL DEFAULT '␣',
   array_val BYTEA  NULL
);

-- This maintenance function automatically creates partitions
   ↪ according to the
-- specified interval (e.g. weekly or monthly). The first
   ↪ partition starts at
-- <begin_time> and ends a day/week/month/year later. This
   ↪ function has to be
-- called regularly (e.g. daily by cron):
--   0 * *   *   *   postgres psql -d mydb -c "SELECT public.
```

```
   ↪ update_partitions('2012-06-01'::timestamp, 'archive', '
   ↪ table_owner', 'week');"
-- This function is based on a more generic version by Nicholas
   ↪  Whittier
-- (http://imperialwicket.com/postgresql-automating-monthly-
   ↪ table-partitions).
CREATE OR REPLACE FUNCTION archive.sample_update_partitions(
   ↪ begin_time timestamp without time zone, schema_name text,
   ↪  table_owner text, plan text)
 RETURNS integer
 LANGUAGE plpgsql
AS $function$
declare startTime timestamp;
declare endTime timestamp;
declare intervalTime timestamp;
declare createStmts text;
declare createTrigger text;
declare fullTablename text;
declare triggerName text;
declare createdTables integer;
declare dateFormat text;
declare planInterval interval;

BEGIN
dateFormat:=CASE WHEN plan='month' THEN 'YYYYMM'
                 WHEN plan='week' THEN 'IYYYIW'
                 WHEN plan='day' THEN 'YYYYDDD'
                 WHEN plan='year' THEN 'YYYY'
                 ELSE 'error'
            END;
IF dateFormat='error' THEN
  RAISE EXCEPTION 'Invalid plan --> %', plan;
END IF;
-- Store the incoming begin_time, and set the endTime to one
   ↪ month/week/day in the future
-- (this allows use of a cronjob at any time during the month/
   ↪ week/day to generate next month/week/day's table)
startTime:=(date_trunc(plan,begin_time));
planInterval:=('1 '||plan)::interval;
endTime:=(date_trunc(plan,(current_timestamp + planInterval)));
createdTables:=0;

-- Begin creating the trigger function, we're going to generate
```

```
   ↪  it backwards.
createTrigger:='
ELSE
RAISE␣EXCEPTION␣''Error␣in␣'||schema_name||'.
   ↪ sample_insert_trigger_function():␣smpl_time␣out␣of␣range'
   ↪ ';
END␣IF;
RETURN␣NULL;
END;
$$
LANGUAGE␣plpgsql;';

while (startTime <= endTime) loop

   fullTablename:='sample_'||to_char(startTime, dateFormat);
   intervalTime:= startTime + planInterval;

   -- The table creation sql statement
   if not exists(select * from information_schema.tables where
      ↪ table_schema = schema_name AND table_name =
      ↪ fullTablename) then
     createStmts:='CREATE␣TABLE␣'||schema_name||'.'||
        ↪ fullTablename||'␣(CHECK␣(smpl_time␣>=␣'''||startTime
        ↪ ||'''␣AND␣smpl_time␣<␣'''||intervalTime||''')),␣
        ↪ INHERITS␣('||schema_name||'.sample)';

     -- Run the table creation
     EXECUTE createStmts;

     -- Set the table owner
     createStmts :='ALTER␣TABLE␣'||schema_name||'.'||
        ↪ fullTablename||'␣OWNER␣TO␣"'||table_owner||'";';
     EXECUTE createStmts;

     -- Create an index on the timestamp
     createStmts:='CREATE␣INDEX␣'||fullTablename||'
        ↪ _channel_time_pkey␣ON␣'||schema_name||'.'||
        ↪ fullTablename||'␣(channel_id,␣smpl_time,␣nanosecs);'
        ↪ ;
     EXECUTE createStmts;

     -- Create foreign key on column channel_id
     createStmts:='ALTER␣TABLE␣'||schema_name||'.'||
```

```
        ↪ fullTablename||' ADD constraint
        ↪ sample_channel_id_fkey FOREIGN KEY (channel_id)
        ↪ REFERENCES '||schema_name||'.channel(channel_id) ON
        ↪ DELETE CASCADE;';
      EXECUTE createStmts;

      -- Create foreign key on column severity
      createStmts:='ALTER TABLE '||schema_name||'.'||
        ↪ fullTablename||' ADD constraint sample_severity_fkey
        ↪ FOREIGN KEY (severity_id) REFERENCES '||schema_name
        ↪ ||'.severity(severity_id) ON DELETE CASCADE;';
      EXECUTE createStmts;

      -- Create foreign key on column status
      createStmts:='ALTER TABLE '||schema_name||'.'||
        ↪ fullTablename||' ADD constraint
        ↪ sample_status_id_fkey FOREIGN KEY (status_id)
        ↪ REFERENCES '||schema_name||'.status(status_id) ON
        ↪ DELETE CASCADE;';
      EXECUTE createStmts;

      -- Track how many tables we are creating (should likely be
        ↪  1, except for initial run and backfilling efforts).
      createdTables:=createdTables+1;
    end if;

    -- Add case for this table to trigger creation sql statement
      ↪ .
    createTrigger:=' ( NEW.smpl_time >= TIMESTAMP '''||startTime
      ↪ ||''' AND NEW.smpl_time < TIMESTAMP '''||intervalTime
      ↪ ||''' ) THEN INSERT INTO '||schema_name||'.'||
      ↪ fullTablename||' VALUES (NEW.*); '||createTrigger;

    startTime:=intervalTime;

    if (startTime <= endTime)
    then
      createTrigger:='
ELSEIF '||createTrigger;
    end if;

end loop;
```

```
-- Finish creating the trigger function (at the beginning).
createTrigger:='CREATE␣OR␣REPLACE␣FUNCTION␣'||schema_name||'.
   ↪ sample_insert_trigger_function()
RETURNS␣TRIGGER␣AS␣$$
BEGIN
IF␣'||createTrigger;

-- Run the trigger replacement;
EXECUTE createTrigger;

-- Create the trigger that uses the trigger function, if it isn
   ↪ 't already created
triggerName:='sample_insert_trigger';
if not exists(select * from information_schema.triggers where
   ↪ trigger_name = triggerName) then
  createTrigger:='CREATE␣TRIGGER␣sample_insert_trigger␣BEFORE␣
     ↪ INSERT␣ON␣'||schema_name||'.sample␣FOR␣EACH␣ROW␣EXECUTE
     ↪ ␣PROCEDURE␣'||schema_name||'.
     ↪ sample_insert_trigger_function();';
  EXECUTE createTrigger;
END if;
return createdTables;
END;
$function$;

ALTER FUNCTION archive.sample_update_partitions(timestamp
   ↪ without time zone, text, text, text) OWNER TO archive;

commit;
```