COSYLAB HZB

EPICS IOC Training

Training Handout

Berlin, August 2025

	Name	Role/Affiliation
Author(s)	Žiga Oven	Cosylab
	Jure Varlec	Cosylab

Contents

Exercise 1: Create an example EPICS Application 3

Exercise 2: Inspect EPICS database 4

Exercise 3: Compile EPICS application 5

Exercise 4: Start an IOC for your application 6

Exercise 5: EPICS Record Basics 7

Exercise 6: Processing Chains 9

Exercise 7: A Chiller Application 10

Exercise 8: Application Based on Stream Device 11

Exercise 9: Control System Studio (CS-Studio or CSS) 12

Exercise 10: EPICS 7 PVAccess and Groups (OPTIONAL) 13

Appendix 14

Abbreviations 14

Links 14

Helpful tools to install on Linux machine 15

EPICS Command Line Tools 15

Exercise 1: Create an example EPICS Application

1. Enter your home directory in terminal:

```
cd
```

2. Create a directory structure where you will create your first EPICS example application:

```
mkdir -p workspace/ex1
```

3. Enter a directory for your first example application:

```
cd workspace/ex1
```

4. Create an EPICS application with the following command:

```
makeBaseApp.pl -t example ex1
```

5. Create an IOC inside an EPICS application with the following command:

```
makeBaseApp.pl -i -t example ex1
```

Note: If you understood what you have to do, but found it difficult to do this exercise, ask the trainer for advice. It may be there are tools that you will find easier to use!

Exercise 2: Inspect EPICS database

Open a text editor such as `gedit` (using the `gedit` command from a terminal), and open file `dbExample1.db`:

```
gedit ex1tApp/Db/dbExample1.db
```

Examine the file and records that it contains.

Exercise 3: Compile EPICS application

Go to the root folder of your EPICS application, `ex1` and compile the entire application by typing:

```
make
```

Check the output of your compilation and make sure there are no compilation errors.

If you had no errors, then your application is ready to run.

Exercise 4: Start an IOC for your application

1. Enter your IOC directory:

```
cd iocBoot/iocex1
```

2. And from there start the IOC using the `st.cmd` file as a parameter to the executable that was built in previous exercise:

```
../../bin/linux-x86_64/ex1 st.cmd
```

3. Observe what happens.
4. Exit the IOC
   a. Type `exit` or press `Ctrl` + `d`
   b. Press `Ctrl` + `c`
   c. Terminate from another window/terminal

```
kill -9 `pgrep ex1`
```

   Explain the differences between these methods. How do they affect the IOC shutdown process?
5. Start an IOC again
6. Check commands inside the IOC shell

   `help`

   `dbl`

   `dbgf`

   `help dbpr dbgf dbpf`

   `dbgf <pv_name>`

   `dbpf <pv_name> <value>`
7. Read from / write to one of the PVs from another window/terminal

```
pvget <pv_name>

pvmonitor <pv_name>

pvput <pv_name> <value>
```

Exercise 5: EPICS Record Basics

1. Create a database file in the correct location inside EPICS application.

   1.1 The new file **does** need to be added to the local `Makefile`. Append the following line in the appropriate place:

   ```
   DB += <name_of_your_new_database>.db
   ```

   1.2 The module needs to be recompiled whenever any source file is added or modified.

   1.3 The new database has to be loaded into the IOC via `dbLoadRecords` command in the `st.cmd` file

2. Create a record of type `ai` with name `$(user):Input_1`; provide a description.

   ```
   record(ai, "$(user):Input_1") {
       field(DESC, "A nice and descriptive description")
   }
   ```

   2.1 Try reading the description and the value using PVAccess.

   2.2 Try setting the value.

3. Set the `TPRO` field of `<user>:Input_1` to nonzero value.

   ```
   pvput <user>:Input_1.TPRO 1
   ```

   3.1 Read the record value from the IOC console.

   3.2 Read the record value via PVAccess while observing the IOC console.

   3.3 Set the value from the IOC console.

   3.4 Set the value via PVAccess while observing the IOC console.

   3.5 Write to the `PROC` field while observing the IOC console.

4. Create a `calc` record named `$(user):Calculation_1` that multiplies the value of `Input_1` by 100.

   ```
   record(calc, "$(user):Calculation_1") {
       field(DESC, "Description must be useful to operators")
       field(INPA, "$(user):Input_1")
       field(CALC, "A * 100")
   }
   ```

   4.1 Read the value after IOC boots. What is the `STAT` of the record?

   4.2 Process the record, then read the value again. What is the `STAT` of `<user>:Input_1`?

   4.3 Change the value of `<user>:Input_1`, then read the value of `<user>:Calculation_1`. Process `<user>:Calculation_1`, then read the value again.

5. Set the `EGU` field to an arbitrary unit and read the value using PVAccess. Notice that `pvget` does not show the unit. What purpose does the `EGU` field serve?

6. Choose alarm thresholds and severities, try setting and reading different values inside and outside of the limits.
7. Monitor the value of `<user>:Input_1` with `pvmonitor`.

   7.1 Try writing the same value to the record several times. How many updates do you see? How many times does the record process?

   7.2 Try writing different values. How many updates do you see?

   7.3 What happens if you set the `MDEL` field to non-zero value?

   7.4 What happens if you set the `MDEL` field to -1?

Exercise 6: Processing Chains

Extend the previous exercise with database scanning. Implement the “push” and “pull” approaches depicted in the image.

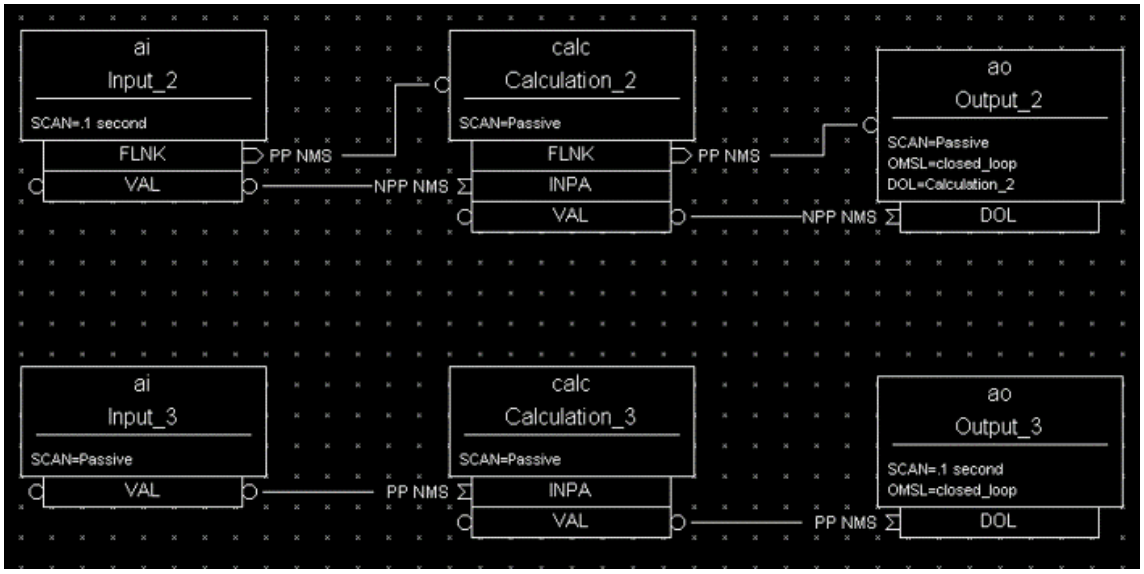

Figure 1: Processing chains

Next, reimplement the “push” chain by using `CP` links.

Exercise 7: A Chiller Application

Goal: Create a new module based on picture below:

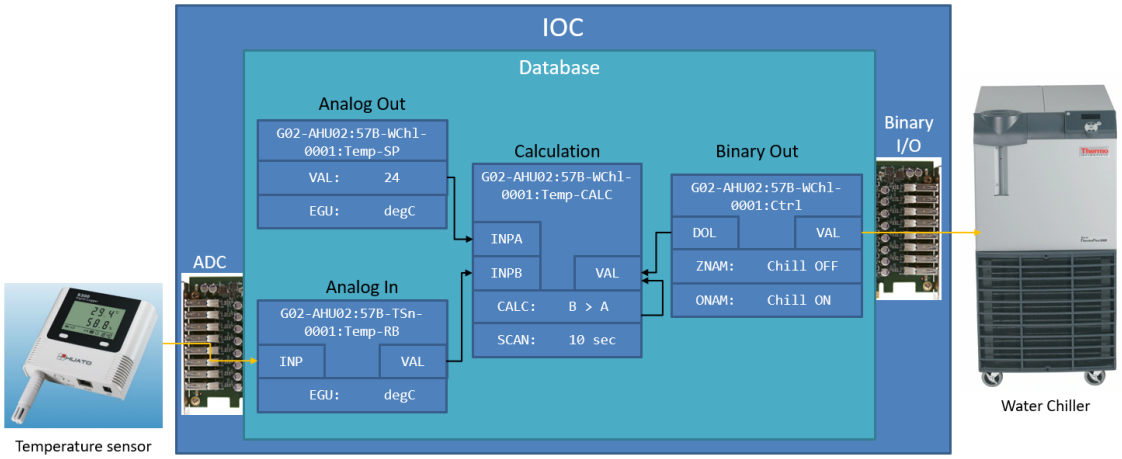

Figure 2: EPICS Database

Steps:

0. Create a new folder inside your `workspace` directory.
1. Create a new EPICS application and IOC with `makeBaseApp.pl` and call it `wch`. This time use the template named `ioc`

```
makeBaseApp.pl -t ioc wch
makeBaseApp.pl -i -t ioc wch
```

2. Create a database file in the correct location and include appropriate instruction into the `Makefile`
3. Load newly created database in the `st.cmd` file
4. Have a setpoint record that simulates readback from temperature sensor
5. Have a setpoint record that determines the threshold for the operation of the water chiller
6. Have a record that simulates control of the water chiller On/Off switch
7. Periodic processing of the records to see if the temperature is OK and turn on the water chiller if necessary
8. Add alarm to notify that the temperature is to high
9. Have a record that shows the status of the section
    - Proposed statuses: `Normal operation`, `Maintenance`
10. Disable record processing based on the status of section
11. Save the setpoint for the threshold through IOC reboots
12. Simulate input to read-back PV to be a sin wave

Exercise 8: Application Based on Stream Device

1. Clone repository from GitLab and inspect to what commands it responds

   1.1. Clone from GitLab.com

   ```
   git clone https://gitlab.com/zoven/stream-simulator.git
   ```

   1.2. Enter the folder and examine the `simulator.py` file for which commands it will accept

2. Start the simulator and test the behavior

   2.1 Start the simulator

   ```
   ./simulator.py
   ```

   2.2 Open another terminal and connect via `telnet`

   ```
   telnet 127.0.0.1 5555
   ```

   You may need to install it first with `sudo apt install telnet`.

3. Create a new EPICS application using `makeBaseApp.pl`.

   3.1. Add `ASYN` and `STREAM` dependencies in `configure/RELEASE`

   3.2. Include `stream-base.dbd`, `asyn.dbd` and `drvAsynIPPort.dbd` into `src/Makefile`

   3.3. Link against `stream` and `asyn` library in `src/Makefile`

   3.4. Create a new protocol file in the same location as your databases

   3.5. Create a database that can talk to the simulator

   3.6. Install the both database and protocol file with `Makefile` definition

   ```
   DB += <your_new_database>.db
   DB += <your_new_protocol>.proto
   ```

4. Create a startup script.

   4.1. Create an asyn port for StreamDevice to communicate through

   ```
   # Connect to simulator through asyn IP Port
   drvAsynIPPortConfigure("DEV1", "127.0.0.1:5555")
   ```

   4.2. Provide path to protocol file

   ```
   # Provide path to the protocol file(s)
   epicsEnvSet("STREAM_PROTOCOL_PATH", "${TOP}/db/")
   ```

   4.3. Load created database

   4.4. Start the IOC and examine the operation

Exercise 9: Control System Studio (CS-Studio or CSS)

1. Open CS-Studio

   ```
   phoebus_start
   ```

2. Create a new display using **Display Editor**

   `Application > Display > New Display`

3. Save the file to desired location

   3.1. `File > Save As ...`

   3.2. Select the path and provide the **`Name:`**

   3.3. `Save`

4. Start adding widgets

   4.1. Create a display for the records from the **Exercise 7**

5. Open display file in runtime mode

   5.1. *Right click on display file*

   5.2._a_ `Execute Display`

   or

   5.2._b_ Click the green play button in the top right corner

6. Observe the results, test the inputs

7. Extend display for **Exercise 7** with:

   7.1. Widgets for setpoint and readback parameters

   7.2. Plot for water chiller control and temperature readback PVs

   7.3. Widgets for showing the alarm status

Exercise 10: EPICS 7 PVAccess and Groups (OPTIONAL)

1. In startup script of first exercise include `circle.db` database from the module

   4.1. Examine what is inside the database before including into startup script

2. Issue following commands in the IOC shell:

   ```
   > help dbl
   > dbl
   > help pval
   > pval
   ```

   Carefully examine the output of both listings and note the differences.

3. Issue following commands on a terminal:

   ```
   pvlist -h

   pvlist

   pvlist <GUID or ipaddr:port>

   pvinfo <pv>

   pvget -h

   pvget -v <pv>

   pvget -r 'field()' <pv>

   pvmonitor -vv <pv>
   ```

   Compare the output of `pvlist` to the listings obtained from the IOC shell.

4. Try any (or all) of the commands on the `$(user):circle` or `$(user):line` process variables. In particular, try

   - `dbgf $(user):circle` from the IOC shell,
   - `caget $(user):circle` from the Linux shell,
   - `pvget $(user):circle` from the Linux shell.

   Explain their behavior.

Appendix

Abbreviations

Abbreviation	Description
ADC	Analog to Digital Converter
CA	Channel Access
CAC	Channel Access Client
CAS	Channel Access Server
CSS	Control System Studio
EPICS	Experimental Physics and Industrial Control System
GUI	Graphical User Interface
HLA	High Level Application
IOC	Input Output Controller
LLA	Low Level Application
PV	Process Variable
PVA	PV Access
PVAS	PV Access Server
PVAC	PV Access Client

Links

- EPICS
  - https://epics-controls.org
  - https://epics.anl.gov
  - https://github.com/epics-base/
  - https://docs.epics-controls.org/en/latest/index.html
- Application Developers Guide [base-3.16.2]
  - https://epics.anl.gov/base/R3-16/2-docs/AppDevGuide.pdf
  - https://docs.epics-controls.org/en/latest/appdevguide/AppDevGuide.html
- Component Reference Manual:
  - https://epics.anl.gov/base/R7-0/9-docs/ComponentReference.html
- Collaboration Supported Records:
  - https://epics-controls.org/resources-and-support/modules/soft-support/
- EPICS modules: https://github.com/epics-modules
  - ASYN
    - https://epics.anl.gov/modules/soft/asyn/
    - https://github.com/epics-modules/asyn

    - https://epics-modules.github.io/asyn/
  - autosave: https://github.com/epics-modules/autosave
  - iocStats: https://github.com/epics-modules/iocStats
  - sequencer: https://epics-modules.github.io/sequencer/

- CS-Studio
  - http://controlsystemstudio.org

Helpful tools to install on Linux machine

```
sudo apt install <package>
```

- `htop` - system monitoring tool
- `vim` - terminal text editor
- `gedit` - standalone (GUI) text editor
- `telnet` - a tool for text communication
- `netcat` - alternative to `telnet`

EPICS Command Line Tools

Description	Command
Get value of one or more PVs	`pvget`
Monitor value changes of one or more PVs	`pvmonitor`
Set the value of one PV	`pvput`
Get information about one or more PVs	`pvinfo`
Get a list of IOCs and their PVs	`pvlist`