

Raspberry Pi Cluster for EPICS Controlled Table-top Simulations

Project Number: 314

Project Members: M. Henders, E. Sadowski, C. Spooner, B. Xing

Project Sponsor: TRIUMF, C. Charles

School Supervisor: T. Randhawa

Date: 2024-02-26

A project with the end goal of creating an EPICS distributed I/O control system using 20 Raspberry Pis that can read in data from simple sensors (ie: light sensors), and control simple devices. (ie: stepper monitors)



British Columbia Institute of Technology
3700 Willingdon Avenue
Burnaby, BC V5G 3H2

Executive Summary

This document is a guide to setting up an EPICS distributed I/O control system using Raspberry Pis that can read in data from simple sensors (ie: light sensors), and control simple devices. (ie: stepper motors).

This document will only briefly cover set-up of a Raspberry Pi cluster with EPICS and will defer most other steps to YouTube tutorials and/or other tutorials of the same nature.

This document is currently in draft

1 Acknowledgments

Huge thanks to Dr. Charles.

We couldn't have done this without your guidance and support!

Contents

1	Acknowledgments	iv
2	Introduction and Purpose	4
2.1	Purpose and Use Cases	4
2.2	End Goals of the Project	4
2.3	Brief Architecture Overview	4
3	Setup of Headless Raspberry Pi with EPICS	6
3.1	Installing EPICS on a RaspberryPi	6
3.1.1	Pre-Installation	6
3.1.2	Installing EPICS base	6
3.1.3	Setting PATH variables	7
3.2	Setup of Input/Output Controller using Python on a RaspberryPi	8
3.2.1	Creation of an Input/Output Controller	8
3.2.2	Python Code to Get Data from Sensors	8
3.3	Adding Analog Inputs to EPICS	12
4	Setup of Windows Server and GUI	14
4.1	Installing EPICS on Windows	14
4.1.1	Setup of Computer Settings	14
4.1.2	Installing Ubuntu	14
4.1.3	Installing EPICS	14
4.1.4	Installing EDM	16
4.1.5	Setting Environment Variables	16
4.2	Setup of the GUI	17
4.2.1	Installing More Fonts	17
4.2.2	Running XLaunch Server	17
4.2.3	Running the GUI	17
5	Creating Graphical User Interfaces	18
5.1	Creating .edl Files	18
5.2	Linking .edl Views	21
5.3	Conditional Rendering	21
5.4	Using Buttons	22
5.4.1	Shell Command Buttons	23
5.4.2	Using Toggle Buttons	24
5.4.3	Creating Reusable Symbols	24
6	Configuring .db Files	30
6.1	Database Records	30
6.2	Database Record Alarm Severity	31
6.3	Creating Interlocks	32
6.4	Enforcing Interlocks with Pyepics	34
7	Utilizing SSH	37
7.1	Creating User Accounts	37
7.2	Server Configuration	38

7.3	Serving EDM GUI Over SSH	39
7.4	Transferring Files Over SSH	39
7.5	Utilizing Our File Transfer Scripts	40
7.5.1	assign_db.sh	40
7.5.2	start_script.sh	41
8	Testing Latency and Aliasing of EPICS and Raspberry Pi	42
8.1	Latency Testing for Raspberry Pi GPIO Pins	42
8.1.1	Testing Raspberry Pi GPIO Pin Read Speed with C	42
8.1.2	Testing Raspberry Pi GPIO Pin Read Speed with Python	44
8.1.3	Testing Raspberry Pi GPIO Pin Read Speed with EPICS	44
8.2	Aliasing Testing for Raspberry Pi	45
8.2.1	Testing Raspberry Pi Aliasing in Python	45
8.2.2	Testing Raspberry Pi Aliasing in EPICS	46
9	Python Scripts to Control IOCs	48
9.1	Different Types of IOCs	48
9.1.1	Binary Input from Device (Sensor)	48
9.1.2	Binary Output to Device (Controller)	49
9.1.3	Binary Output to Device (Interlocks)	50
9.1.4	Analog Input from Device (Sensor)	51
10	High-Level Application - EPICS Through your Web Browser	54
10.1	HLA Tech Stack and Development	54
10.1.1	Tech Stack	54
10.1.2	Development	54
10.1.2.1	__init__.py	54
10.1.2.2	run.py	54
10.1.2.3	views.py	54
10.1.2.4	templates/index.html	54
10.1.3	Running the HLA	54
10.1.3.1	Installation	54
10.1.3.2	Running the application	55
10.1.3.3	Troubleshooting	55
11	SD Card Images and Flashing	56
11.1	Creating an Image	56
11.2	Flashing an Image to an SD card	56
12	Common Issues	57
12.0.1	Ubuntu For Windows	57
12.0.2	IOCs not Connecting to Server	57
12.0.3	Server Start Script not Running Correctly	57
12.0.4	Unable to Reassign IOCs Using PI Status Page	57
12.0.5	Problems Reassigning IOCs	58
12.0.6	Difficulties Clicking a Specific Object in EDM	58
12.0.7	Unable to pull/push to the Gitlab Repo	58
A	Other Tools and Setup for EPICS	59
A.1	Installing a Raspberry Pi Emulator	59
A.2	Installing Pre-built EPICS tools for Windows	59
A.2.1	Graphing with StripTool	59
A.3	Test Databases in EPICS	60
A.3.1	Setting up a Test Database	60
B	ER-PCC Repo Directory Map	62

B.0.1	er-pcc	62
B.0.2	er-pcc/high-level-application	62
B.0.3	er-pcc/src/database	62
B.0.4	er-pcc/src/scripts	62
B.0.5	er-pcc/src/screens/istf _s <i>creens</i>	62
B.0.6	er-pcc/src/tests	62

List of Figures

List of Tables

2 Introduction and Purpose

2.1 Purpose and Use Cases

Thank you for reading the documentation of the project!

This documentation should thoroughly explain the installation, setup, and reasoning of this project. Also there is detailed instructions on how you can set up a similar EPICS environment for lab testing, teaching/training of the EPICS environment, and/or collecting data using the tools EPICS provides.

Source code used in this project is available at <https://github.com/MattiasHenders/epics-gui-triumf>

2.2 End Goals of the Project

The main points we focused on as we developed this project were as follows:

1. Create an EPICS system that is cheaper than the one currently at TRIUMF
2. Create an EPICS system that is easy to configure for new users
3. Create an EPICS system that can be used for smaller, tabletop experiments
4. Create a GUI that uses the StripTool Interface like Isaacs

2.3 Brief Architecture Overview

There are a few main components of this project, each has their own configuration instructions in their own individual chapters.

1. Raspberry Pi Input Output Controller (IOC)
2. Server running on a Windows computer, serving an EPICS GUI and a High Level Application
3. SSH scripts to facilitate file transfer from server to IOC
4. Python scripts to control functions

The IOC tower (Chapter 3) is connected to a LAN through a 24 port switch. The server (laptop) (Chapter 4) is also on this LAN, and communicates with the IOCs to get and serve Process Variables (PVs) to sensors attached to the GPIO pins of the IOCs (Chapter 9). This communication is handled through EPICS. Python scripts utilizing the pyepics package allow us to do this (Chapter 9).

- Server -> EPICS -> IOC -> python scripts -> GPIO pins / Sensors

The Server has 2 GUIs, an EDM GUI to replicate the GUIs used at TRIUMF to control physics experiments (Chapter 4, 5), and a Flask application to serve PV information over HTTP (Chapter 11).

In the EDM GUI, each IOC has been pre-loaded with a startup script that tries to re-register the same PVs as it had last time it ran (Chapter 7.5.2). If those PVs are occupied then the IOC will appear in the EDM GUI as an open device (Chapter 7.5). Users can manually assign this IOC to have a new purpose through a couple of clicks in the GUI. Once the IOC has been assigned SSH scripts will transfer the necessary python scripts to the IOC to begin reading and writing the desired PVs (Chapter 7.5.2, 9).

IOCs communicate through databases (Chapter 6), by pushing values to the field associated with the PV name. Think of it more like a hash map / dictionary than a traditional database. Database values can be retrieved, updated, or subscribed to.

If you need to register more PIs there is a SD card image that can be used to flash the IOCs (Chapter 11). You can also make your own images.

The Flask application is a non-core feature that serves PV information to the GUI (Chapter 10). It is commonly referred to as a "High Level Application" (HLA). It is currently just a proof of concept and is open for expansion.

The final product should consist of a number of Raspberry Pis running EPICS operating as IOCs (Input/Output Controller) connected to a central EPICS server hosted on a Windows computer. There is an operational GUI on

the server capable of controlling and monitoring from each sensor on the Raspberry Pis. An additional High-Level Application (HLA) was also created which allowed for updating and reading values from the EPICS server through a web application.

3 Setup of Headless Raspberry Pi with EPICS

Note: A prebuild .img file with all of the following is configured and can easily be flashed onto a microSD card for immediate use.

Please contact your system admin for a copy of triumph.img

Otherwise continue on with the tutorial to create your own EPICS install from scratch.

Flash the Raspbian Lite 32-bit OS to your SD card and plug it into the Pi.

WARNING: THIS GUIDE DOES NOT COVER STEPS NEEDED TO CONNECT SENSORS TO THE RASPBERRY PI. LOOK AT THE SENSORS INSTALLATION GUIDE FOR FURTHER DETAILS

The hardware guide and sensors I got were purchased from Freenove, and can be found by clicking this link to Amazon.ca

Additional information can be found in the EPICS control documentation: <https://docs.epics-controls.org/projects/how-tos/en/latest/index.html>

3.1 Installing EPICS on a RaspberryPi

3.1.1 Pre-Installation

Create a user for the RPi, then enter the following command in the console:

```
sudo raspi-config
```

Navigate to "Localisation Options > WLAN Country" and select your country

Navigate to "Interface Options > SSH" and enable SSH on the Pi.

Optionally navigate to "System Options > Wireless LAN" and enter the WiFi password if applicable.

Optionally navigate to "Interface Option > SPI" and enable if you want to read analog inputs.

Either SSH into the Pi from a computer connected to the Internet or ensure that the Pi is connected to the Internet and run the following commands:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get -y install python3-pip
sudo apt-get install git
sudo apt-get install screen
sudo apt-get install sshpass
pip install psutil
pip install pyepics
pip install smbus
pip install adafruit-blinka
adafruit-circuitpython-mcp3xxx
```

3.1.2 Installing EPICS base

Once logged in and a command prompt is showing enter the following commands to make folders to store the epics base:

```
cd /opt
mkdir epics
cd ./epics
git clone -b 3.15 https://git.launchpad.net/epics-base
cd epics-base
make
```

EPICS is now installing, it may take up to an hour for the installation to finish.

3.1.3 Setting PATH variables

Now we need to set up the PATH variables so we can run EPICS from anywhere in the terminal.

```
nano ~/.bashrc
```

To the bottom of the file, add the following text and save the file.

```
export EPICS_BASE=/opt/epics/epics-base
export EPICS_HOST_ARCH=linux-arm
export PATH=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}:$PATH
```

Repeat this step for the root user:

```
sudo -i
nano .bashrc
```

To the bottom of the file, add the following text and save the file.

```
export EPICS_BASE=/opt/epics/epics-base
export EPICS_HOST_ARCH=linux-arm
export PATH=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}:$PATH
```

You should now be able to start an EPICS IOC by typing the following command in your console:

```
softIoc
```

The following text should be displayed in your terminal:

```
epics>
```

Congratulations! epics-base was installed correctly.

Then exit by typing:

```
exit
```

3.2 Setup of Input/Output Controller using Python on a RaspberryPi

3.2.1 Creation of an Input/Output Controller

An Input / Output Controller (IOC) allows EPICS to gather, store and transmit data over a network between devices that are running EPICS.

An EPICS IOC database is made up of one or more records which keep track of values called Process Variables (PVs). Process Variables are points of data transmitted through EPICS.

The following instructions will allow you to create an example database (.db) file.

```
cd $HOME/EPICS;  
mkdir IOC;  
cd IOC  
nano epics.db
```

For this example we will making a sample using a distance sensor.
You can you any sensor you want. Just replace the filenames as needed.

Edit the epics.db file so it reads:

```
record(ai, "sensor:distance"){  
    field(DESC, "Distance Sensor in centimeters")  
}  
record(stringin, "output:led"){  
    field(DESC, "Status of the LED FALSE or TRUE")  
}
```

NOTE: More information about how to customize and create database files can be found in chapter 5.

You can now run a softIoc with your newly created .db file by typing:

```
softIoc -d epics.db
```

3.2.2 Python Code to Get Data from Sensors

Now we will make a Python file to send the data to EPICS. The code will need to be altered based on what sensor you have and what pins you connect the device to.

Please make sure you follow the correct guides when installing as you can *DESTROY* your Pi and sensors if the wrong voltage is applied.

```
nano distance.py
```

```
import RPi.GPIO as GPIO  
import time  
import epics  
  
trigPin = 16  
echoPin = 18  
MAX_DISTANCE = 220      # define the maximum measuring distance, unit: cm  
timeOut = MAX_DISTANCE*60 # calculate timeout according to the maximum measuring distance  
  
def pulseIn(pin,level,timeOut): # obtain pulse time of a pin under timeOut
```

```

t0 = time.time()
while(GPIO.input(pin) != level):
    if((time.time() - t0) > timeOut*0.000001):
        return 0;
t0 = time.time()
while(GPIO.input(pin) == level):
    if((time.time() - t0) > timeOut*0.000001):
        return 0;
pulseTime = (time.time() - t0)*1000000
return pulseTime

def getSonar():    # get the measurement results of ultrasonic module,with unit: cm
    GPIO.output(trigPin,GPIO.HIGH)    # make trigPin output 10us HIGH level
    time.sleep(0.00001)    # 10us
    GPIO.output(trigPin,GPIO.LOW) # make trigPin output LOW level
    pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut) # read plus time of echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0    # calculate distance with sound speed 340m/s
    return distance

def setup():
    GPIO.setmode(GPIO.BOARD)    # use PHYSICAL GPIO Numbering
    GPIO.setup(trigPin, GPIO.OUT) # set trigPin to OUTPUT mode
    GPIO.setup(echoPin, GPIO.IN)  # set echoPin to INPUT mode

def loop():
    while(True):
        distance = getSonar() # get distance
        epics.caput('sensor:distance', distance) #Put the distance in EPICS
        print ("The_distance_is_%.2f_cm"%(distance))
        time.sleep(1)

if __name__ == '__main__':    # Program entrance
    print ('Program_is_starting...')
    setup()
    try:
        loop()

```

nano led.py

```

import epics
import RPi.GPIO as GPIO
import time

ledPin = 11    # define ledPin

def setup():
    GPIO.setmode(GPIO.BOARD)    # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPin, GPIO.OUT) # set the ledPin to OUTPUT mode
    GPIO.output(ledPin, GPIO.LOW) # make ledPin output LOW level
    print ('using_pin%d'%ledPin)

def loop():

    prevLED = False

```

```

while True:

    #get the status from EPICS
    boolLED = epics.caget('output:led') == "True"

    if (prevLED != boolLED): #check for changes

        if (boolLED):
            GPIO.output(ledPin, GPIO.HIGH) # make ledPin output HIGH level to turn on led
            print ('led_turned_on_>>>') # print information on terminal

        else:
            GPIO.output(ledPin, GPIO.LOW) # make ledPin output LOW level to turn off led
            print ('led_turned_off_<<<')

    prevLED = boolLED
    time.sleep(1) # Wait for 1 second

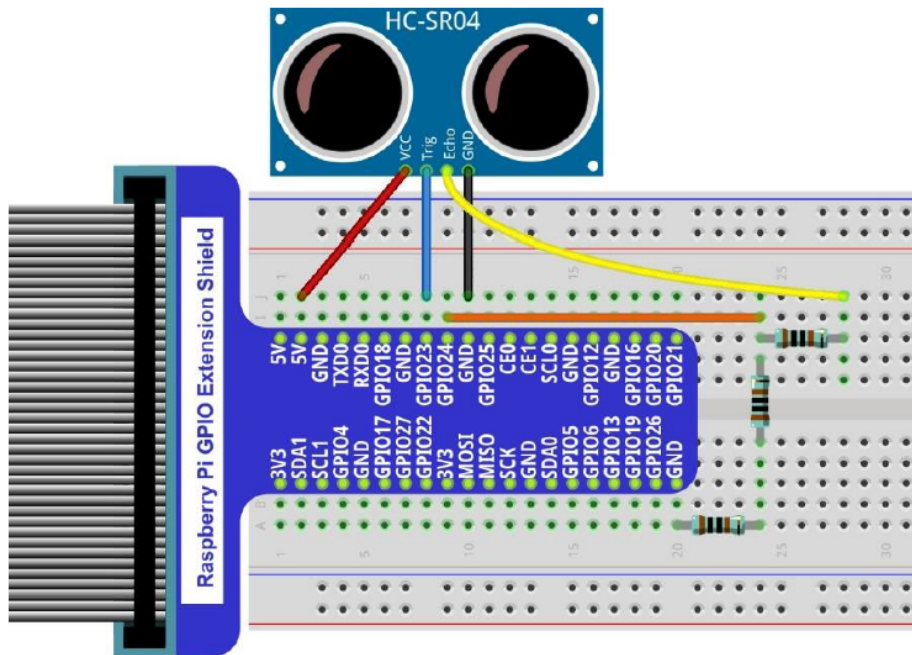
def destroy():
    GPIO.cleanup() # Release all GPIO

if __name__ == '__main__': # Program entrance
    print ('Program_is_starting_...\n')
    setup()
    try:
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        GPIO.output(ledPin, GPIO.LOW) # make ledPin output LOW level
        destroy()

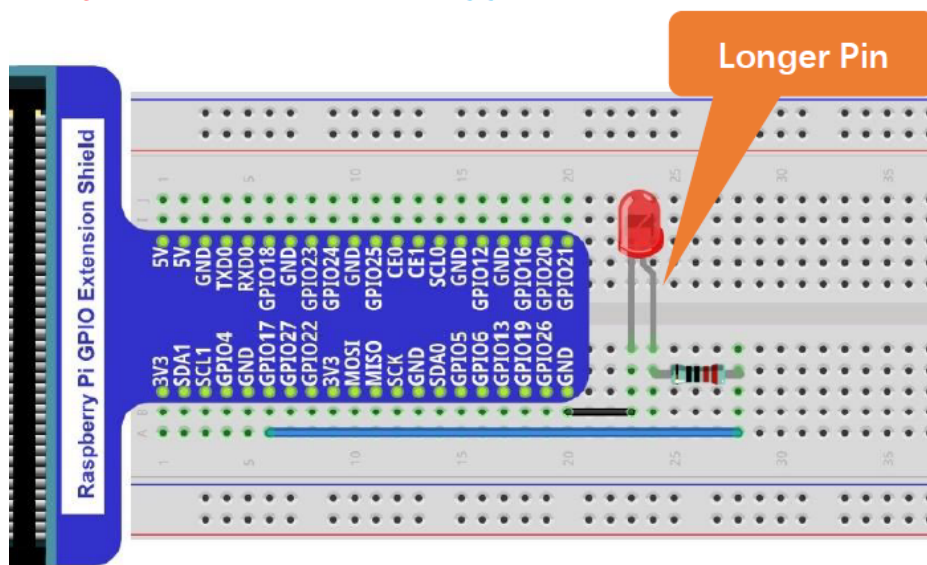
WARNING: YOU SHOULD HAVE PLUGGED YOUR NEEDED SENSOR HARDWARE INTO THE RASPBERRY PI
AT THIS POINT

```

I followed the hardware guide listed at the top of Chapter 2.
The final install of the distance sensor for input looked like so:



The final install of the LED bulb for output looked like so:



Save the files, and turn on the database using:

```
softIoc -d epics.db
```

Then run this command in a separate window to start writing to the EPICS Database.

```
python distance.py
```

You can also run this command in a separate window to start reading from the EPICS Database.

```
python led.py
```

You now should have EPICS installed where you are able to broadcast data to the network!

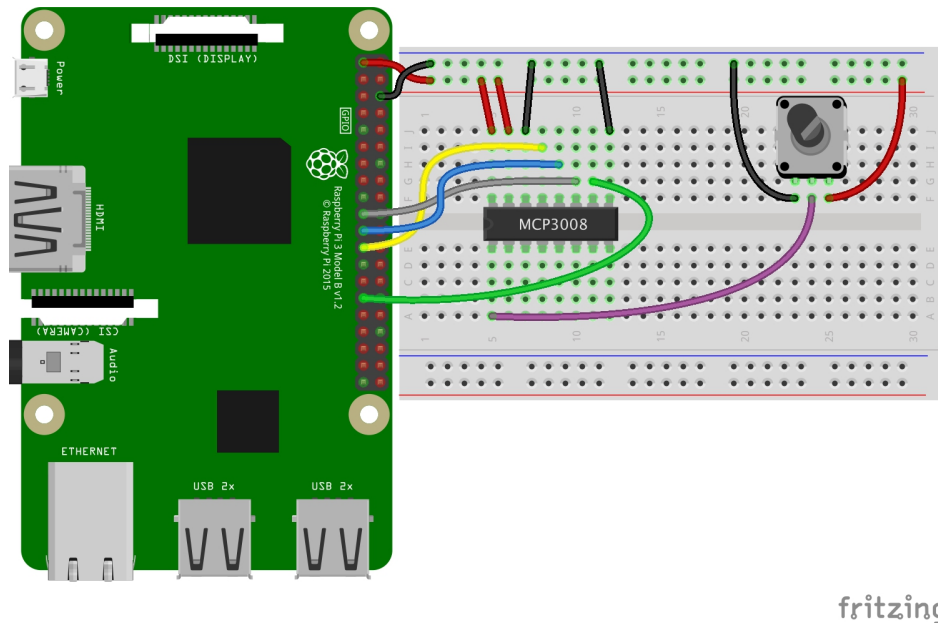
3.3 Adding Analog Inputs to EPICS

Raspberry Pi by default can only read in digital inputs. An Analog-to-Digital-Converter(ADC) is required to read and write analog values to EPICS

This example installation is based on the methods at: <https://learn.adafruit.com/mcp3008-spi-adc/python-circuitpython>

We are using a MCP3008 ADC to connect to our Raspberry Pi.

The wiring diagram to connect the ADC to the Pi is as follows:



fritzing

Once wired, we will need to enable the SPI interface to be able to read from the ADC.

1. Use `sudo raspi-config` to enter into settings
2. Go to **Interfacing Options**.
3. Then select **SPI**.
4. When asked to enable, select **YES**.
5. SPI is now enabled and you can exit raspi-config.

You will also need to enable I2C

1. Use `sudo raspi-config` to enter into settings
2. Go to **Interfacing Options**.
3. Then select **I2C**.
4. When asked to enable, select **YES**.
5. I2C is now enabled and you can exit raspi-config.

To be able to run the code to read and write voltages:

1. Ensure that pip3 is installed.

```
sudo apt-get install python3-pip
```

2. Next install the following packages:

```
sudo pip install adafruit-blinka
```

```
sudo pip install adafruit-circuitpython-mcp3xxx
```

```
sudo pip install pyepics
```

3. create an analogTest.db file containing the following record to store values through EPICS

```
record(ai, 'voltage'){ }
```

4. go to the directory with the analogTest.db file and start running EPICS

```
softloc -d analogTest.db
```

5. While this is running, open another terminal to execute the test code below.

Command to run: `sudo python3 ./ADCTest.py`

6. You should now be storing the voltage readings into EPICS

Test code available at <https://github.com/MattiasHenders/epics-gui-triumf/blob/main/src/tests/ADCTest.py>

SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries

SPDX-License-Identifier: MIT

```
import time
import busio
import digitalio
import board
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
from epics import PV

# create the spi bus
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)

# create the cs (chip select)
cs = digitalio.DigitalInOut(board.D5)

# create the mcp object
mcp = MCP.MCP3008(spi, cs)

# create an analog input channel on pin 0
chan = AnalogIn(mcp, MCP.P0)

voltagePV = PV('voltage')
while True:
    print("Raw_ADC_Value: ", chan.value)
    print("ADC_Voltage: " + str(chan.voltage) + "V")
    voltagePV.put(chan.voltage)
    time.sleep(0.5)
```


4 Setup of Windows Server and GUI

4.1 Installing EPICS on Windows

4.1.1 Setup of Computer Settings

1. Open Windows Powershell in Administrator Mode
2. Run the following 2 commands
 - A. `dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart`
 - B. `dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart`
3. Close Powershell
4. Restart your Computer
5. Run the following .msi file: [CLICK HERE TO DOWNLOAD](#).
6. Follow the installation wizard and is asked anything just use the default selection.
7. Restart your Computer

4.1.2 Installing Ubuntu

1. Install Ubuntu 18.04 LTS from Microsoft Store
2. Open Ubuntu 18.04 LTS
3. Set username to pi and password of your choice
4. When the terminal opens fully, run the following commands
 - A. `sudo apt update`
 - B. `sudo apt upgrade`
5. Close the Ubuntu Terminal
6. Open Windows Powershell in Administrator Mode
7. Run the following command
 - A. `wsl --set-version Ubuntu-18.04 1`
8. Close Powershell
9. Restart your Computer

4.1.3 Installing EPICS

1. Open Ubuntu and run the following command
 - A. `sudo -i`
2. Enter your password from before
3. Run the following 2 commands
 - A. `cd ~`
 - B. `nano .bashrc`
4. Scroll to the bottom and paste in the following block of text:

```
export PATH=/opt/epics/epics-base/bin/linux-x86_64:$PATH
export EPICS_BASE=/opt/epics/epics-base
export EPICS_HOST_ARCH=linux-x86_64
```

```

export EPICS_CA_AUTO_ADDR_LIST=YES
export EPICS_EXTENSIONS=/opt/epics/extensions
export PATH=$EPICS_EXTENSIONS/bin/$EPICS_HOST_ARCH:$PATH
export EDMPROJECTS=$EPICS_EXTENSIONS/src/edm/setup
export EDMOBJECTS=$EPICS_EXTENSIONS/src/edm/setup
export EDMHELFILES=$EPICS_EXTENSIONS/src/edm/helpFiles
export EDMFILES=$EPICS_EXTENSIONS/src/edm/edmMain
export EDMLIBS=$EPICS_EXTENSIONS/lib/$EPICS_HOST_ARCH
export LD_LIBRARY_PATH=$EDMLIBS:$EPICS_BASE/lib/$EPICS_HOST_ARCH

```

5. Close the Ubuntu Terminal

6. Open Ubuntu and run the following command AGAIN

A. `sudo -i`

7. Enter your password from before

8. Run the following commands

```

cd /
cd opt
mkdir epics
cd epics
apt-get update
apt --fix-broken install
apt-get install build-essential git iperf3 nmap openssh-server vim libreadline-gplv2-dev libgif-dev libmotif
-dev libxmu-dev libxmu-headers libxt-dev libxtst-dev xfonts-100dpi xfonts-75dpi x11proto-print-dev autoconf
libtool sshpass
apt-get upgrade

```

9. Close the Ubuntu Terminal

10. Open Ubuntu and run the following command AGAIN

A. `sudo -i`

11. Enter your password from before

12. Run the following commands

```

cd /opt/epics
git clone --branch 3.15 https://github.com/epics-base/epics-base.git
wget https://epics.anl.gov/download/extensions/extensionsTop_20120904.tar.gz
tar xvf extensionsTop_20120904.tar.gz
rm extensionsTop_20120904.tar.gz
wget http://ftp.lyx.org/pub/linux/distributions/Ubuntu/pool/main/libx/libxp/libxp-dev_1.0.2-1ubuntu1_amd
64.deb
wget http://mirror.ufscar.br/ubuntu/pool/main/libx/libxp/libxp6_1.0.2-1ubuntu1_amd64.deb
dpkg -i libxp6_1.0.2-1ubuntu1_amd64.deb libxp-dev_1.0.2-1ubuntu1_amd64.deb
rm libxp6_1.0.2-1ubuntu1_amd64.deb libxp-dev_1.0.2-1ubuntu1_amd64.deb
cd epics-base
make

```

4.1.4 Installing EDM

1. From the same terminal window as before run the following commands.

```
cd /opt/epics/
sed -i -e '21cEPICS_BASE=/opt/epics/epics-base' -e '25s/^/#/' extensions/configure/RELEASE
sed -i -e '14cX11_LIB=/usr/lib/x86_64-linux-gnu' -e '18cMOTIF_LIB=/usr/lib/x86_64-linux-gnu'
extensions/configure/os/CONFIG_SITE.linux-x86_64.linux-x86_64
cd /opt/epics/extensions/src
git clone https://github.com/epicsdeb/edm.git
cd /opt/epics/extensions/src
sed -i -e '15s$/ -DGIFLIB_MAJOR=5 -DGIFLIB_MINOR=1/' edm/giflib/Makefile
sed -i -e 's/ungifllg' edm/giflib/Makefile*
cd edm
make clean
make
cd setup
sed -i -e '53cfor libdir in baselib lib epicsPv locPv calcPv util choiceButton pnglib diamondlib giflib
videowidget' setup.sh
sed -i -e '79d' setup.sh
sed -i -e '81i\\ \\ $EDM -add $EDMBASE/pnglib/O.$ODIR/lib57d79238-2924-420b-ba67-dfbecdf03
fcd.so' setup.sh
sed -i -e '82i\\ \\ $EDM -add $EDMBASE/diamondlib/O.$ODIR/libEdmDiamond.so' setup.sh
sed -i -e '83i\\ \\ $EDM -add $EDMBASE/giflib/O.$ODIR/libcf322683-513e-4570-a44b-7cdd7cae0de5.
so' setup.sh
sed -i -e '84i\\ \\ $EDM -add $EDMBASE/videowidget/O.$ODIR/libTwoDProfileMonitor.so' setup.sh
HOST_ARCH=linux-x86_64 sh setup.sh
cd ../..
```

2. Close the Ubuntu Terminal

4.1.5 Setting Environment Variables

1. Open Ubuntu and run the following command

A. nano /.bashrc

2. Scroll to the bottom and paste in the following block of text:

```
export PATH=/opt/epics/epics-base/bin/linux-x86_64:$PATH
export EPICS_BASE=/opt/epics/epics-base
export EPICS_HOST_ARCH=linux-x86_64
export EPICS_CA_AUTO_ADDR_LIST=YES
export EPICS_EXTENSIONS=/opt/epics/extensions
export PATH=$EPICS_EXTENSIONS/bin/$EPICS_HOST_ARCH:$PATH
export EDMPVOBJECTS=$EPICS_EXTENSIONS/src/edm/setup
export EDMOBJECTS=$EPICS_EXTENSIONS/src/edm/setup
export EDMHELPPFILES=$EPICS_EXTENSIONS/src/edm/helpFiles
export EDMFILES=$EPICS_EXTENSIONS/src/edm/edmMain
export EDMLIBS=$EPICS_EXTENSIONS/lib/$EPICS_HOST_ARCH
export LD_LIBRARY_PATH=$EDMLIBS:$EPICS_BASE/lib/$EPICS_HOST_ARCH
```

3. Close the Ubuntu Terminal

4.2 Setup of the GUI

4.2.1 Installing More Fonts

1. Run the following .exe files in order:
2. Run this .exe file and just use all the defaults given. [CLICK HERE TO DOWNLOAD 1.](#)
3. Run this .exe file and when prompted **CLICK ALL FONTS CHECKBOXES LISTED** in the menu. [CLICK HERE TO DOWNLOAD 2.](#)

4.2.2 Running XLaunch Server

1. Run xLaunch, and click:
 - A. Multiple Windows
 - B. Next
 - C. Start no client
 - D. Next
 - E. Check the 'No Access Control' box
 - F. Next
 - G. Finish

4.2.3 Running the GUI

1. Open Ubuntu and run the following 3 commands
 - A. `git clone https://github.com/MattiasHenders/epics-gui-triumf.git`
 - B. `cd epics-gui-triumf/src/screens`
 - C. `edm`
2. Click file > Open by Path and then click on your .edl file under the "Files" column
3. Once the file opens in a new window, middle click the screen and click on "Execute" to start the program

The EDM GUI should open

5 Creating Graphical User Interfaces

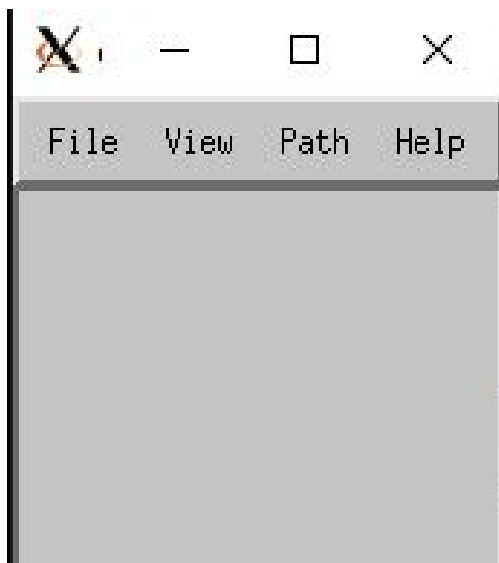
In order to create graphical user interfaces that can interact with EPICS' channel access tools, the use of a display manager is required. For this document we will be using EPICS' Extensible Display Manager (EDM). However, there are several alternative display managers available, some of which are addressed in the appendices of this document.

By using EDM, it is possible to create and execute interfaces consisting of multiple .edl files, each file representing a view/window inside your control system.

This guide will provide some instruction how to get started building your own control system interface.

5.1 Creating .edl Files

Ensure you are running your PC X server, and start EDM by opening your Ubuntu terminal and typing "edm". You should see the following window appear:



Select File > New to create a new .edl view.

Re select your main EDM window and you should now be able to see the beginnings of your new .edl view.

To customize the settings of your .edl view, use your mouse's middle click by clicking down on your mouse scroller (if it has that capability). Otherwise you may need to set a custom key-bind for your computer's mouse middle click in your system settings if you do not have a mouse that can do this. You should see the following menu appear:

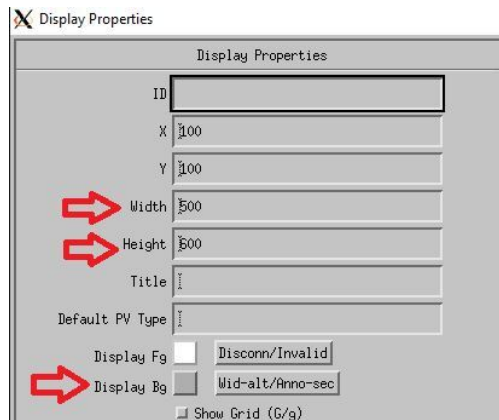


Using this menu you can perform a number of actions, most importantly it will allow you to save your page, change display properties and execute your view.

For now lets begin by changing the dimensions of your .edl view. Select Display Properties from the drop down menu and inside the Width and Height field enter the size of the dimensions you want for your page.

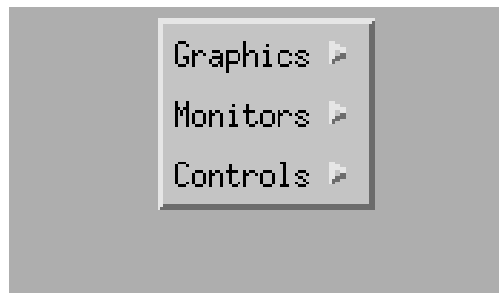
Ensure that when you are changing the size of your screen you use the Display Properties menu, as dragging the corners of your .edl view will not properly increase or decrease the dimensions of your page.

You can also change the background color of your view by clicking the square to the right of the Display Bg field.



After making a change, use the middle click menu and click save or save as to save your changes.

To add content to your view, left click, drag your mouse and release your click to be prompted with the following menu:



Panning over any of the three menu options should open another drop down menu from which you can select and add content to your view (widgets, images, text boxes, etc). For an example, enter the Monitor menu and select Text Monitor.

Double click the newly created box to open its properties.

Inside the PV field, enter "sensor:distance" to tell your newly created text monitor to monitor the value of the PV we created in the EPICS database demo portion of Chapter 2. Click Apply, then OK to update your widget.

After, middle click anywhere on the background of your .edl view to open the settings menu, then select execute to run your view.

To test that the Text Monitor in your view will react to a change in the sensor:distance PV you will need to open two additional Ubuntu terminals.

In the first window, navigate to the directory where you previously saved the epics.db file from Chapter 2 and serve the .db file by typing the following command:

```
softIoc -d epics.db
```

Next, using the second Ubuntu terminal enter the following command:

```
caput sensor:distance 20
```

If you navigate back to the .edl view you previously executed you should now be able to see the text view displays the number 20, which is the value currently stored by the sensor:distance PV.

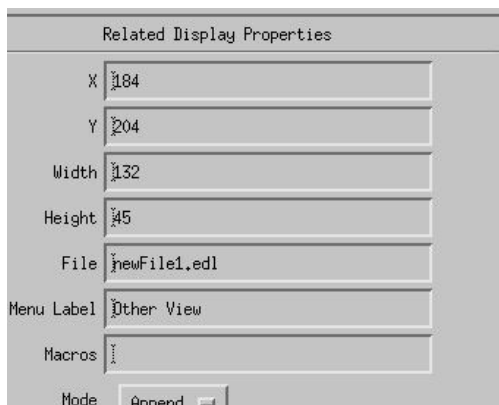
5.2 Linking .edl Views

Navigate back to your main EDM window and save the .edl file you were working on from the previous subsections, choose whatever name you like.

Close your saved .edl view and create a new one. Select Controls from the left click menu and select Related Display . Double click on it to open its properties.

Inside the Related Display Properties, you can add a link to another .edl display by entering the path to the .edl file relative to the .edl file you are currently editing.

For linking to the .edl file we made in the previous subsection, you can simply enter <name_of_the_file>.edl assuming that they are both in the same directory. In the example below you can see we linked the previous view by entering newFile1.edl.



Related Display Properties	
X	184
Y	204
Width	132
Height	45
File	newFile1.edl
Menu Label	Other View
Macros	[Icon]
Mode	Append

Executing your new view and clicking the related display button you just created should open a window to our previous view which contains our Text Monitor widget.

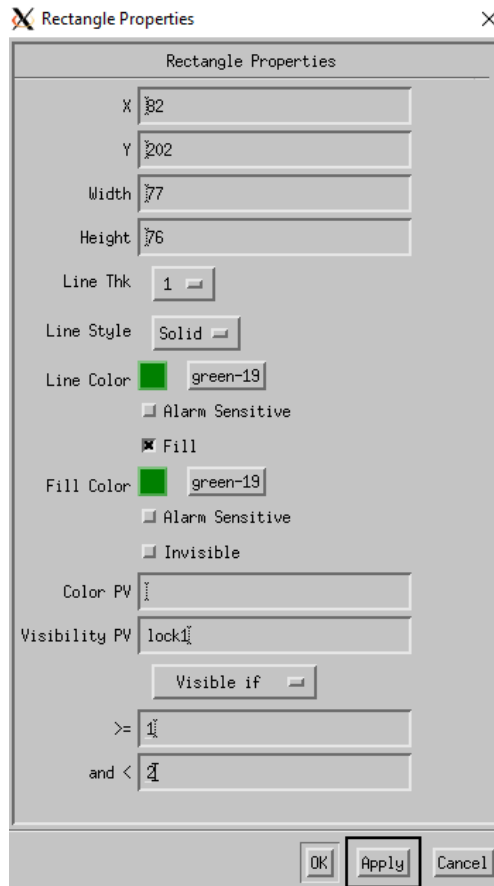
You can also enter a label for your navigation button by utilizing the Menu Label field, or if you want to create a button that opens a drop down menu which links to any number of views, click the Additional Displays button and simply repeat the same process as before by using the File and Label fields.

By linking multiple .edl files together you can make your interface easier to use and interpret by separating different parts of your control system into separate views.

5.3 Conditional Rendering

An important tool that can be used towards creating more dynamic interfaces is to conditionally render widgets/icons based on the presence or absence of a PV value.

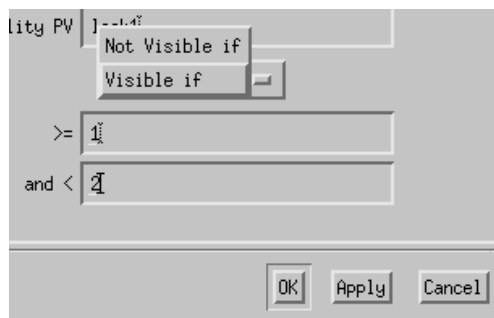
To do this, double left click any component to open its properties and enter the value of the PV you want to use to determine if the component should be displayed or not inside the Visibility PV field.



Here we are using the PV "lock1" which is a binary value of 0 or 1 to determine if our component should be visible inside the EDM interface.

For reference, you can find an example of how we declared the lock1 PV in Chapter 5.1.

To determine when we want our component to display we also need to set a condition inside the component properties. Below you can see we selected "Visible If" lock1 is greater than or equal to 1 and less than 2. By doing this, our component will only display if the value of lock1 is 1 since lock1 can only be set to 0 or 1 because it is a binary input record.



5.4 Using Buttons

Buttons can be used throughout EDM interfaces to run shell scripts and interact with PVs. This can provide your interface with a higher level of functionality if you wanted to be able to run some external code with a shell script or change a PV value on button click.

5.4.1 Shell Command Buttons

For an example of how to use a Shell Command button, start by left clicking, dragging and releasing to open the Control component menu. Select Shell Command and then click OK.

Next we will need to create a shell script to run with our button. Open a new Ubuntu terminal and navigate to the directory where you usually start EDM from. Enter the following commands to create and open an empty shell script :

```
touch testscript.sh
```

```
nano testscript.sh
```

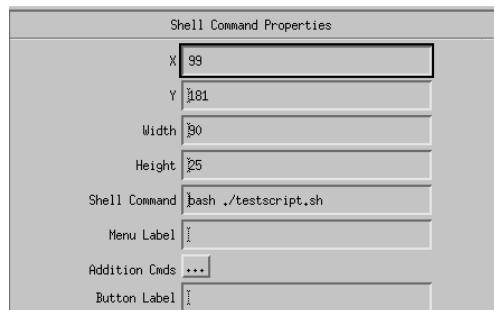
You should now be able to see an empty file open inside your Linux terminal.

In this file you can input your shell code that you want to execute when your button is clicked inside your interface.

For this example we will simply print "Hello World" to our EDM console, copy and paste the following code inside your new shell script:

```
echo "Hello World"
```

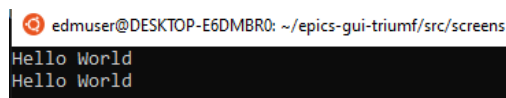
Hit Ctrl + S then Ctrl + X to save your script, then navigate back to your EDM interface. Double left click your Shell Command button to edit its properties, and inside the Shell Command field enter "bash" followed by the the correct path to your shell script. For example:



It is also possible to add more shell scripts to your button by changing your button into a button which opens a drop down menu on click. You can add additional shell commands by opening the "Addition Cnds" menu inside your Shell Command button properties, and following the same process by entering a menu label with a path to the shell script you want to be executed.

NOTE: The path to your shell script must be relative to the file directory which you executed EDM from otherwise your interface will not be able to find the script. Using a path of "/" indicates that the script your are attaching to your interface is located in the same directory which the EDM command was executed. For this reason it is advisable to always execute EDM from the same file location to prevent pathing issues.

Click Apply, OK, and then execute your .edl view. Click your newly made Shell Command button and then navigate to the Ubuntu window you used to run EDM. You should be able to see "Hello World" logged into the Ubuntu console to verify that your shell script executed successfully.



Using shell scripts in this way allows you to use whatever external code you need for your EDM implementation. You can use your shell script to execute any code you would otherwise execute via command line, allowing you to integrate C, Java, Python or whatever other code you need into your interface.

5.4.2 Using Toggle Buttons

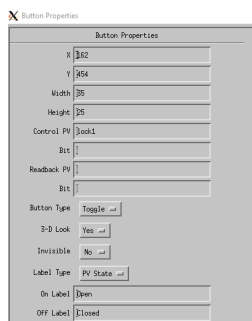
To use a standard toggle Button, left click and drag on your EDM background to open the Controls menu and create a Button.

A toggle Button can be used in combination with a binary input PV to switch its value between 0 or 1 (ON/OFF, OPEN/CLOSED). This is a good addition to any interface because it allows your GUI to interact with EPICS PVs without having to use a separate Linux terminal.

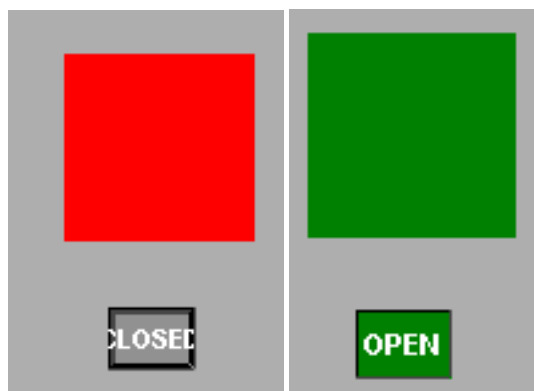
Double left click your new button to open its properties, and enter the name of the binary input PV you want to toggle inside the Control PV field. You can also make your button display different labels based on its state by using the On Label and Off Label fields.

For this example enter "lock1" inside the Control PV field, then click Apply and OK.

If you are not already, ensure that you serve a binary input PV named "lock1", for an example of how to do this refer to Chapter 5.1



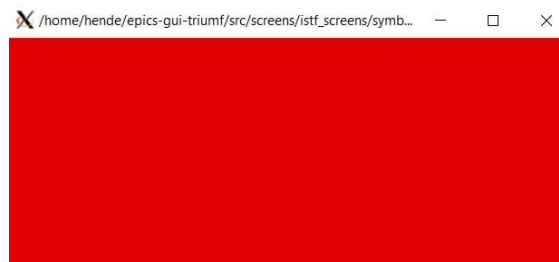
Execute your EDM view and click the button you created to see that it can toggle the lock1 PV between opened and closed. Also notice that if you applied the lock1 PV to a shape and set the shape to be Alarm Sensitive the color of the shape will toggle between red and green when you click the button.



5.4.3 Creating Reusable Symbols

This subsection will cover how to make reusable symbols for your User Interfaces that can dynamically change based on the state of a linked PV.

Start by creating a new .edl file, it will have the default background like so:



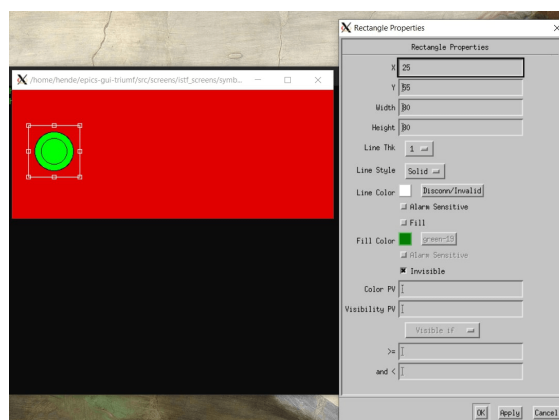
Next, use the graphic tools to draw whatever shape you need your symbol to be.



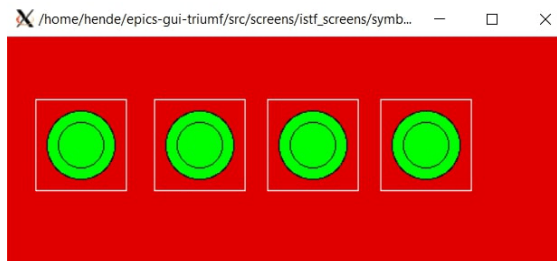
Next draw a rectangle around your shape



Then open the properties, select the 'Invisible' option.



Now copy and paste this shape AND the invisible rectangle for as many states as you need.




When all your states are complete, middle click, and select the Auto Make Symbol button.
Make sure you SAVE YOUR FILE



Finally, you can add the symbol in any other files.
Monitors > Symbol



In the properties menu type in the path to the symbol file.

 Symbol Properties
 ✕

Symbol Properties

X 530

Y 470

Symbol File symbols/turbo_pump.edi

Color PV

PV Names

AND

0

h

XOR

0

h

Shift

0

AND

0

h

XOR

0

h

Shift

0

AND

0

h

XOR

0

h

Shift

0

AND

0

h

XOR

0

h

Shift

0

AND

0

h

XOR

0

h

Shift

0

☐ Binary Truth Table

Orientation

Original

☐ Preserve Original Size

☒ Preserve Original Colors

Fg/Line Color

Disconn/Invalid

Bg/Fill Color

green-19

Number of Items

2

Item Number

1

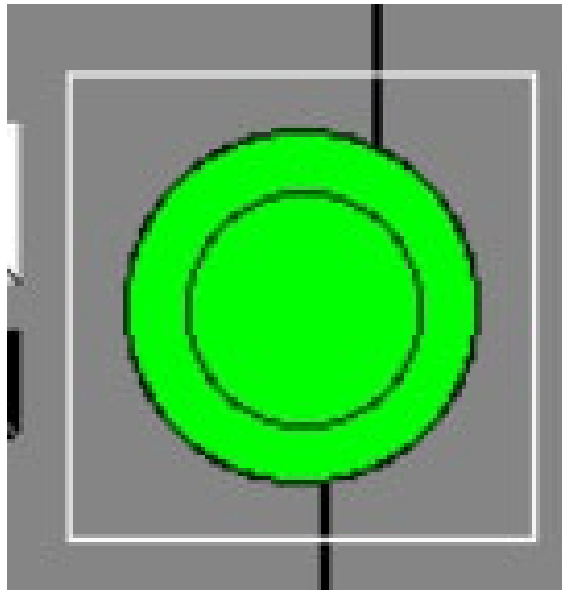
>=

0

<

0

You now have a reusable symbol!



6 Configuring .db Files

EPICS database files are one of the main components which allows EPICS to store and transmit data between IOCs.

This chapter will provide an overview of some of the different types of database records and field properties that can be utilized to perform more complex logic with the values stored by PVs.

6.1 Database Records

Each database file contains one or more records, with each record representing a Process Variable (PV).

The format of declaring a new database record is as follows:

```
record(<Record Type>, "<PV Identifier>"){  
    field(<Field Type>, "<Field Value>")  
    <...>  
}
```

There are over 30 unique types of records that can be added to an EPICS database file. Each record has field properties that are entirely unique to that record type, as well as some field properties that are common amongst multiple record types.

The EPICS developers have created some great documentation of each record type which would be a good resource to look at in order to learn more. You can find that documentation here:

<https://epics.anl.gov/base/R7-0/4-docs/RecordReference.html>

When a database file is served by an IOC, the IOC transmits the state of each of the PVs contained in its .db file (given that the .db file is served successfully) to each of the other IOCs that are connected to the same network.

Consider the contents of the following .db file example:

```
record(bi, "lock1"){  
    field(INP, "0")  
    field(ZNAM, "CLOSED")  
    field(ONAM, "OPEN")  
    field(ZSV, "MAJOR")  
    field(OSV, "NO_ALARM")  
    field(SCAN, "1 second")  
}
```

Create a new file called test.db and paste the above contents inside of it.

Here we have created a PV named lock1 with a type of "bi" which represents a binary input 0 or 1.

Regarding the field parameters of this record:

INP (Input) represents a default input value for the binary input record, here we have set it to 0 by default.

ZNAM (Zero Name) declares a string reference to a value of 0 which has been set to CLOSED in this instance. This means that if you were to interact with this PV using a caput command, you can give it an argument of either 0 or CLOSED.

ONAM (One Name) declares a string reference to a value of 1 which has been set to OPEN in this instance. Similar to ZNAM, if you were to interact with this PV using a reference to OPEN, the record would be able to associate this reference to a value of 1.

Declaring names for binary input values can be useful in the way that it provides the user or designer of the PV with a better idea of the intended purpose of the record. In this case, because we are simulating a lock mechanism with this record, CLOSED and OPEN provide the user with a better idea of the record's purpose than 0 or 1.

ZSV (Zero Severity) declares the level of alarm severity indicated by a value of zero.

OSV (One Severity) declares the level of alarm severity indicated by a value of one.

The following subsection will discuss in more detail what the purpose of alarm severity is.

SCAN declares the frequency at which the alarm severity of this record will be updated. In this instance because we have set the alarm severity to 1 second, the associated severity will be checked and updated every second.

6.2 Database Record Alarm Severity

Alarm severity in EPICS can be used to both create dynamic displays in EDM as well as perform channel access operations in response to an abnormal severity state using the EPICS Alarm Handler.

More information about the EPICS Alarm Handler can be found here:

http://www-linac.kek.jp/cont/kek/IHEP-2001/Noboru_Yamamoto/alhUserGuide.pdf

For the purposes of this document we will be using alarm severity to dynamically update our EDM interface.

There are four different alarm states we can use in our EPICS database records, and each alarm state has an associated coloration when used within an EDM interface.

The four states are:

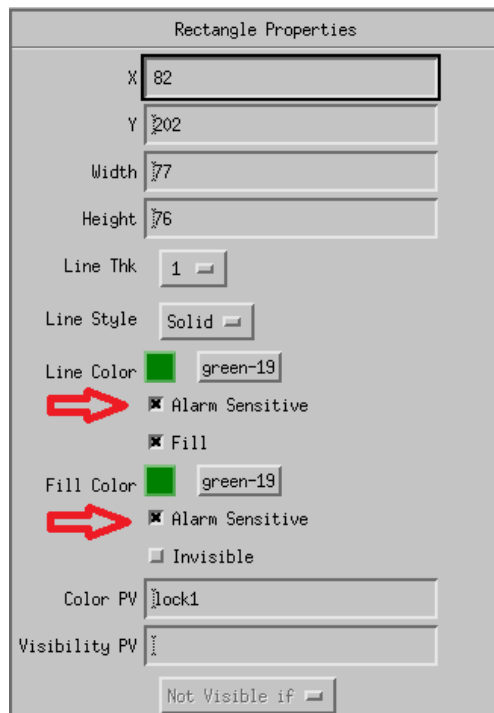
INVALID - White

NO_ALARM - Green

MINOR - Yellow

MAJOR - Red

For an example of how to utilize alarms in EDM, create a new edm view and add any kind of graphic to it you like. Double click the graphic to open its properties and check both of the boxes labelled "Alarm Sensitive" and enter "lock1" in the Color PV field to make the color of the border and inside of the shape respond to the alarm state of our PV "lock1".



Next, open a new Ubuntu terminal and navigate to the location of test.db and serve the .db file by typing the following command in your terminal:

```
softIoc -d test.db
```

Click apply > OK, and then use the middle click menu to execute your view. Notice that the object you just edited will now appear red, regardless of what Line Color and Fill Color you applied to it.

Because we have applied the Alarm Sensitive property to our shape and stated in our lock1 database record that the severity of a value of zero is "MAJOR" and since we specified the default starting value of lock1 is 0 we see a red coloration.

6.3 Creating Interlocks

For our EDM implementation we needed to create interlocks that prevent the user from performing certain interactions if a set of conditions are not met. In the previous chapter we learned how to conditionally render components if a PV does not have a certain value, here we will be using a combination of conditional rendering and calc database records to create an interlock.

To further enforce your desired interlocking conditions it would be advisable to also utilize callback functions through pyepics or another EPICs channel access library. More information on how to do this can be found in the section following this.

Modify your test.db file to match the following example:

```
record(bi, "lock1"){
  field(INP, "0")
  field(ZNAM, "CLOSED")
  field(ONAM, "OPEN")
}
```

```

        field(ZSV, "MAJOR")
        field(OSV, "NO_ALARM")
        field(SCAN, ".1 second")
    }
    record(bi, "lock2") {
        field(INP, "0")
        field(ZNAM, "CLOSED")
        field(ONAM, "OPEN")
        field(ZSV, "MAJOR")
        field(OSV, "NO_ALARM")
        field(SCAN, ".1 second")
    }
    record(calc, "lock1-2_link"){
        field(SCAN, ".1 second")
        field(INPA, "lock1")
        field(INPB, "lock2")
        field(CALC, "A + B")
        field(HIGH, "2")
        field(LOW, "1")
        field(HSV, "NO_ALARM")
        field(LSV, "MAJOR")
    }
}

```

With the above modifications we have added two additional records to our .db file.

lock2 is an exact copy of lock1, a binary input record which can contain 0 or 1.

lock1-2_link is a calculated input record which we will be using to check if both locks 1 and 2 are OPEN and if that condition is true then render a button.

A calc record is a record which determines its value based on a set of inputs. In this example we will be using the values of lock 1 and lock 2 as A and B in the calculated value of the lock1-2_link record.

Regarding fields of the lock1-2_link record:

SCAN is used to determine the rate at which lock1-2_link checks the values of both lock 1 and 2 to calculate its own value.

INPA (Input A) is used to represent an input as a variable A in order to perform a calculation which will determine the value of the lock1-2_link record. In this case we have set Input A's value to the value of the PV lock1.

INPB (Input B) is used to represent an input as a variable B in order to perform a calculation which will determine the value of lock1-2_link record. In this case we have set Input B's value to the value of the PV lock2.

Note that it is possible to use more than two inputs in a calc record.

CALC is used to perform a calculation with each of the Input fields, and will set the value of this record to the result of this calculation. In this case we have set the value of calc to "A + B" because the purpose of this calc record is to check that both lock1 and lock2 are OPEN. Because the OPEN state of lock1 and lock2 are represented by a 1, we know that if the result of A + B is 2 that both locks are OPEN.

HIGH is used to determine at what value a 'HIGH' level of severity has been reached. Here because HIGH has been passed a value of "2", we have declared that any value of 2 or greater yielded by this calc record will be considered HIGH severity.

LOW is used to determine at what value a 'LOW' level of severity has been reached. In this example we have set LOW to "1", which will declare any value of 1 or lower as LOW severity.

Note that HIGH and LOW are used to declare the upper and lower bounds of a range of values and should not be considered literal representations of HIGH and LOW severity. Literal representations of the level of severity occurring in each severity range are specified by HSV and LSV.

HSV (High Severity) is used to represent the alarm state of a value that has reached a threshold of HIGH severity. Because we are using our calc record to check if both lock 1 and 2 are OPEN, and an OPEN lock state is represented

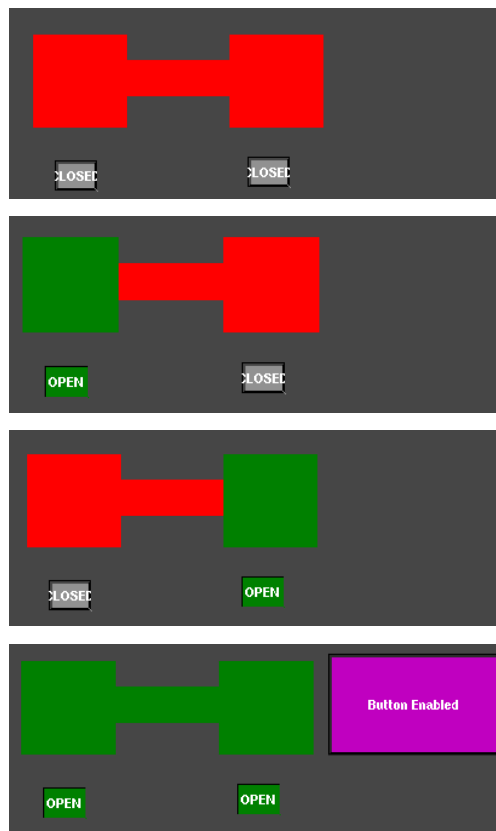
by a value of 1 we want our upper bound to display a green color. Therefore, our HSV is set to NO_ALARM since our upper bound of both locks being open is an acceptable state.

LSV (Low Severity) is used to represent the alarm state of a value that has reached a threshold of LOW severity. Because our lower bound of anything less than 2 is an unacceptable state we have set LSV to HIGH which will display a red color for any resulting calculation of $A + B$ that is less than 2.

As a hypothetical example, say that in our EDM interface we want to have a button which performs an action, but we only want to allow our user to be able to interact with that button if a set of conditions are met. For this example we will ensure that both the lock1 and lock2 PVs are open before allowing this button to be utilized.

Create two buttons and set their Control PVs to lock1 and lock2 respectively. Next create a button and set its Visibility PV to lock1-2_link and change it to be Visible If ≥ 2 and < 3 . If you are interested in seeing the coloration of the lock and lock_alarm state colors you can also create some shapes, apply each respective PV to them and enable their Alarm Sensitivity properties.

Your setup when executed should appear something like this when executed; notice the link between the shapes representing lock1 and lock2 has its PV set to lock1-2_link which will cause it to have a red coloration if both locks are not open, and green if they are both open. In addition to this, we can see that the button on the right hand side will only be visible if both locks are open.



6.4 Enforcing Interlocks with Pyepics

By using Pyepics we can ensure that the integrity of our interlocking conditions are maintained without the use of the EDM interface. This will ensure that whatever conditions we need to enforce on our system are enforced even if a value is manually changed through a caput operation.

To do this, create a long running python script named test.py by typing:

```
nano test.py
```

Inside the python file, create a main method as follows:

```
from epics import ca, caput, caget, PV
import time
def main():
    pv = PV("examplePV")
    pv.add_callback(testCallback)
    while True:
        time.sleep(1)
```

Within the main method above, we have used the PV module from the pyepics library to get our pv 'examplePV' and have declared that we want to add to it a callback function called 'testCallback'

A callback function is a piece of code that will execute every time the value of a PV changes, which will allow us to monitor a PV and execute some code if it is in an undesirable state.

In the above code snippet we have also created a while loop causing our program to sleep. What this is doing is preventing our program from completing, which will ensure that any callback functions we set will continue to run until we close the script by killing the process or using Ctrl + C to sigkill the script.

The next step for enforcing a condition on this PV is to create our callback function. Modify your test.py script like so:

```
from epics import ca, caput, caget, PV
import time
def testCallback(pvname, value, host, **kws):
    if value > 400:
        pv = PV("powerSwitch")
        pv.put(0)
def main():
    pv = PV("examplePV")
    pv.add_callback(testCallback)
    while True:
        time.sleep(1)
```

With the above modification we have defined our callback function 'testCallback'. To use a Pyepics callback you need to create a function that has the required parameters which in this case are pvname, value, host and **kws.

More information about how to define callback functions can be found here:

<https://cars9.uchicago.edu/software/python/pyepics3/pv.html>

Within our 'testCallback' function we are using the PV module again to get a mock example of another PV called 'powerSwitch' that we want to control based on the current value of 'examplePV'. We are declaring that if the value of 'examplePV' is greater than 400, we get our 'powerSwitch' PV and save it as 'pv' then call 'pv.put(0)' to change its value to 0. What this can do is if powerSwitch is a binary input record of 0/1 representing on/off, this could be used to turn the power off on the 'powerSwitch' device, thus preventing it to continue functioning if 'examplePV' is above the value 400.

This type of logic can be used to enforce specific conditions on our control system and prevent all kinds of unintended interactions.

For more information on how to enforce conditions like this, also have a look at how to use alarms here:
<https://cars9.uchicago.edu/software/python/pyepics3/alarm.html>

7 Utilizing SSH

For the requirements of our project we wanted to both be able to load our GUI on a remote computer using SSH, as well as transfer files from our server to our RPIs using SSH.

This chapter will cover the details of how we accomplished each of these design goals.

For reference, it is important to note that the way we implemented our ssh-based functionality was by using password authentication rather than ssh key authentication. Because our implementation uses a Raspberry PI image that is the same across all of our IOC devices, each device will have the same username and password and is connected to a private local network. Therefore, for our implementation requirements it made the most sense to not manually/dynamically generate ssh keys for each individual RPI and maintain a list of authorized ssh keys on our server. If for your implementation requirements you would like to use ssh keys, have a look at this site for more information:

<https://www.ssh.com/academy/ssh/keygen>

7.1 Creating User Accounts

For our implementation we wanted to be able to restrict access to our control system over ssh to authorized users. In order to implement this some Linux setup is required.

In previous sections we have been using `sudo -i` to log in as a super user to both install and use EPICS/EDM. However, if we want to be able to prevent users who connect to our server via SSH from performing harmful or unintended actions, we can create our own user accounts and restrict access via SSH to only allow connections through those accounts.

In order to create a new user, first log in as a super user by typing `sudo -i` and entering your credentials.

Add a user by typing the following command, name the user whatever you like:

```
sudo adduser testuser
```

You will then be prompted to enter and confirm your password and optionally you can enter some user information like first name, last name, address, etc.

For the purposes of SSHing into our server we will use this user that was just created as a generic entry point. If for your implementation you require that some users have more permissions than others, you can always add more users and change their permissions however you like.

To allow your newly created user to interact with EPICS/EDM we will also need to set their environment variables inside of their `.bashrc` file.

Open the newly created user's `.bashrc` file by typing `nano ~/.bashrc` and paste in the following exports at the bottom of the file:

```
export PATH=/opt/epics/epics-base/bin/linux-x86_64:$PATH
export EPICS_BASE=/opt/epics/epics-base
export EPICS_HOST_ARCH=linux-x86_64
export EPICS_CA_AUTO_ADDR_LIST=YES
export EPICS_EXTENSIONS=/opt/epics/extensions
export PATH=$EPICS_EXTENSIONS/bin/$EPICS_HOST_ARCH:$PATH
export EDMPVOBJECTS=$EPICS_EXTENSIONS/src/edm/setup
export EDMOBJECTS=$EPICS_EXTENSIONS/src/edm/setup
export EDMHELPPFILES=$EPICS_EXTENSIONS/src/edm/helpFiles
export EDMFILES=$EPICS_EXTENSIONS/src/edm/edmMain
export EDMLIBS=$EPICS_EXTENSIONS/lib/$EPICS_HOST_ARCH
export LD_LIBRARY_PATH=$EDMLIBS:$EPICS_BASE/lib/$EPICS_HOST_ARCH
```



```
export DISPLAY=$(echo ${SSH_CLIENT} | awk '{print $1}')
trap logout SIGINT
trap logout INT
cd /home/testuser/epics-gui-triumf/src/screens/istf_screens/
edm -x -noedit istf_main_menu.edl
logout
```

This will allow your newly created user to be able to access EPICS over SSH. Note that if you attempt to log in to this user on the machine running the EPICS server to use EDM, you will not be able to render the display because the X11 interface is being forwarded to the connected SSH_CLIENT. For running an EDM instance on the server's machine, use the root user.

Furthermore, the commands appended after the DISPLAY export will force any user that logs in as testuser to run EDM, and once EDM is exited or the process is killed from inside the terminal the SSH connection will be ended. By doing this we prevent this user from having access to the server's command terminal.

If changes need to be made to testuser's .bashrc file, you can make changes as the root user by typing the command:

```
nano /home/testuser/.bashrc
```

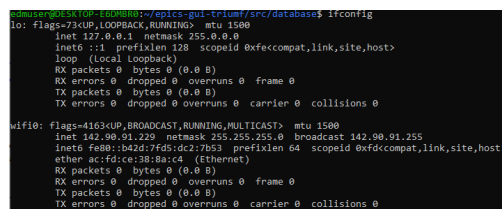
7.2 Server Configuration

In order to allow our EPICS server to serve our GUI over SSH there were several modifications we needed to make via Linux.

Begin by logging in as the root user using `sudo -i` and type the following command in your terminal:

```
ifconfig
```

You should see some text similar to this appear in your terminal:



```
root@epics-gui-triumf:/epics-gui-triumf/src/databases# ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 1500
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0xfeccompact,link,site,host>
    loop (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 142.90.91.229 netmask 255.255.255.0 broadcast 142.90.91.255
    inet6 fe80::b42d:7f5d:dc2:7b53 prefixlen 64 scopeid 0xefdcompact,link,site,host>
    ether ac:fd:ce:38:8a:c4 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Make note of inet address, this is the IP address of your server.

NOTE: use the inet address that does not begin with 127.0

Proceed by editing your SSH config by typing the command: `nano /etc/ssh/sshd_config`

You should now be looking at your Linux system's SSH config.

Be very careful to not make any unnecessary changes as they may be difficult to fix later.

Throughout the file you will see various properties, prefixed with a # character and appearing in blue text. Properties in this file that are prefixed with a # will not be applied to the system's SSH config.

Remove the # character next to the following properties to enable them and set their value as follows:

- Port 22
- ListenAddress 0.0
- PermitRootLogin no

- AllowUsers exampleuser
- MaxAuthTries 6
- MaxSessions 10
- UsePrivilegeSeparation no
- PasswordAuthentication yes
- ChallengeResponseAuthentication no
- UsePam yes
- X11Forwarding yes
- PrintMotd no

If your setup requires more than one user account to allow specific permissions to some users, you can add multiple user accounts by simply separating them with a space, declaring like so: "AllowUsers exampleuser anotheruser thirduser"

If you do not see one of the properties above listed in your sshd_config file, go ahead and add it to your file, the location of where you add it should not matter.

7.3 Serving EDM GUI Over SSH

Now that we have set up the SSH configuration for our server, we should be able to serve our EDM interface to other devices that have access to the EPICS server's IP address.

If the system acting as the EPICS server is running a Linux subsystem via a Windows or Mac OS, ensure that a Ubuntu terminal is open on that system so that the SSH configurations are loaded to allow connections to the subsystem.

On the system acting as your EPICS server, run the following command to ensure your ssh service and its configurations have been started so that it can accept connections:

```
sudo service ssh start
```

On another system in the same network, run your Xming PC X server and enter the following command in Ubuntu:

```
sudo ssh -X testuser@<Your server's IP address>
```

Alternatively, you can also use an SSH client such as PuTTY to create this connection. Ensure that if you use PuTTY you enable X11 Forwarding in your Connection > SSH > X11 settings and set the X display location to :0.0

After successfully connecting to the EPICS server, your Xming client should be served with an EDM client from the server you just connected to with SSH.

7.4 Transferring Files Over SSH

For our EPICS control system implementation we wanted to design a system which can be easily configured without having to manually edit or copy files within each of the RPIs. In order to do this we utilized scp (Secure Copy Protocol) file transfer.

Simply put, scp allows you to securely transfer files from one machine to another. This is possible using a command such as:

```
scp ./file.txt pi@123.456.789.101:
```

The above command could securely copy file.txt to the user 'pi' at the IP address '123.456.789.101' to their directory at ' ' which would likely be '/home/pi'.

Note that the './' before file.txt declares that the file we want to transfer is in the current directory our linux terminal is located in. You must give the correct relative path to the file that you want to transfer to the scp command.

7.5 Utilizing Our File Transfer Scripts

For our EDM GUI design we implemented a system where we can re-assign IOCs at the click of a button. In order to do this we needed to utilize several apt packages and scp/ssh commands in our shell script. The shell scripts that enable this functionality are our assign_db.sh script which is executed by the EPICS server, and our start_script.sh script which is provided to each Raspberry PI IOC.

We incorporated this script inside of our EDM interface's piStatus page by using shell script buttons. One of the main reasons why we wanted to do this was to allow us to easily re-configure or replace an IOC if perhaps one of our IOCs gets damaged while operating in the high voltage cage of ISTF.

7.5.1 assign_db.sh

The way the script works is you pass it two PV names as its parameters, the PV that you want to re-assign, and the PV that you want to re-assign the targeted PV to.

For example you could call this script like so:

```
bash ./assign_db.sh ISTF:FC0 ISTF:FC1
```

This command would re-assign the IOC currently serving the ISTF:FC0 PVs to now serve the ISTF:FC1 PVs. In addition to providing the PVs required by each of these IOCs, the script will transfer and run the correct python scripts to provide GPIO functionality to the RPI.

At a high level, our assign_db.sh script first checks to see that the IOC we want to reassign our IOC to is not already being served on our local network, and if it is not then it transfers the appropriate script and .db files to the targeted Raspberry PI, then killing any softIoc currently running on that PI and re-starting it using the newly transferred files.

A few apt packages to make note of which are used throughout the scripts used to implement this functionality are:

sshpass - A package which allows us to provide a password to an ssh or scp command which prevents our user from having to type anything since we want our interface to be able to transfer files just by clicking a button.

screen - A package that lets us create and run commands in an entirely independent linux session. Because EPICS softIocs do not seem to have the ability to persist as a background task in a linux terminal, we have made thorough use of screen to allow us to start IOCs and their scripts in their own independent linux sessions so that we can start these scripts remotely through ssh, and ensure our IOCs do not terminate unexpectedly.

In order to allow this script to support additional PVs and/or .db files, you will need to add the correct key-value pair to the IOC_DEVICE_ASSOCIATIONS.txt file in the format: <PV>=<DB File> using the exact name of the PV and .db file.

Say for example you wanted to add an IOC named IOC1 to this setup, and you want it to serve a database file named database.db. You would add this pair of values to the IOC_DEVICE_ASSOCIATIONS.txt like so:

```
IOC1=database.db
```

Each entry in this file must exist on its own line of the file.

In order for the script to be able to find the corresponding .db and script files, ensure that they are located in the /src/database/IOC and src/scripts/IOC directories respectively. Corresponding python GPIO scripts should have the same name as their .db file counterpart.

To add to our previous example, if you wanted to associate a python script for IOC1 to execute when it is started, or when another IOC is re-assigned to IOC1; you will need to create a file in the src/scripts/IOC directory named database.py.

7.5.2 start_script.sh

Our start_script.sh script is used by each IOC to start up. It will first use the PV identifier contained in /env/IOC_CONFIG as its PV identifier, and then check that no other IOCs on the network are currently serving that PV identifier. If the PV identifier is already taken, the IOC will start as a 'No Device' ND_IOC, serving an IOC with the database ND_IOC.db. Otherwise, it will use the ID contained in /env/IOC_CONFIG as its PV identifier and start its IOC with the dbconfig.db file as its database.

A 'No Device' IOC is an IOC which was not able to start because the PV it had the last time it was started has been taken by another IOC on the network. Rather than automatically assigning the IOC to a PV, the solution we came up with was to assign what are essentially placeholder PVs that allow our EPICS GUI to be able to find IOCs which were not initialized correctly, and allow the user to choose which device they would like to re-assign the 'No Device' IOCs to.

NOTE: Always start the EPICS server before running any of the IOCs using this script. The start script will not allow IOCs to start if they cannot connect to the server. If you want to use an IOC independently from the server you can still ssh into the RPI and manually start an IOC.

8 Testing Latency and Aliasing of EPICS and Raspberry Pi

8.1 Latency Testing for Raspberry Pi GPIO Pins

Latency testing was completed in order to determine the limits of the RaspberryPi as an IOC device. There could be cases in which faster hardware would be required for certain projects (e.g. fast data acquisition)

Each script should be run across all of the Raspberry Pis in your stack. In our case, we tested across 2 towers with 12 Pis and 10 Pis respectively.

Full source code can be found at <https://github.com/MattiasHenders/epics-gui-triumf/tree/main/src/tests>

8.1.1 Testing Raspberry Pi GPIO Pin Read Speed with C

To test the speed of the GPIO Pin read speed, we attached a pin directly connected to a function generator outputting a square wave at a certain frequency. This script was written in C, as the lower level of the language can execute read commands faster for more accurate representation of a theoretical max read rate.

The script runs 20,000 loops reading from a GPIO pin and calculates the number of reads per second. This is then run 10 times to calculate the average read speed for a GPIO pin on the Raspberry Pi.

To run our script:

First install the lgpio package:

```
wget http://abyz.me.uk/lg/lg.zip
unzip lg.zip
cd lg
make
sudo make install
```

The following code can then be used to test the average read speed off GPIO pin 23 (can be changed). A .txt file will be created with the average reads per second and the data readings will be written to .csv files in the *c-no-epics* folder.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <unistd.h>
#include <time.h>

#include <lgpio.h>

#define LFLAGS 0
#define PIN 23
#define SigOUT 12
#define LOOPS 20000
#define CYCLES 10

int main(int argc, char *argv[]) {
    int h;
    int i;
    int j;
    double t0, t1;
    int status;
    int arr[20000];
    int results[CYCLES];
```

```

int result;
double sum;

FILE * latencyInformation;
latencyInformation = fopen ("/c-no-epics/GpioOutC_2022-05-04_CS-BX_x.txt", "w+");

// Runs test for number of cycles
for (j = 0; j < CYCLES; j++) {

    //int result;
    h = lgGpiochipOpen(0);

    if (h >= 0) {
        if ((status = lgGpioClaimOutput(h, LFLAGS, SigOUT, 0)) == LG_OKAY) {
            t0 = lguTime();

            for (i = 0; i < LOOPS; ++i) {
                //result = lgGpioRead(h, PIN);
                arr[i] = lgGpioRead(h, PIN);
                //printf("%d", result);
            }

            t1 = lguTime();

            FILE *f;
            char name[] = { "/c-no-epics/c_no_epics_x.csv" };
            name[24] = j + '0';
            f = fopen(name, "w+");
            for(i = 0; i < LOOPS; i++) {
                fprintf(f, "%d\n", arr[i]);
            }
            fclose(f);
            result = (1.0 * LOOPS) / (t1 - t0);
            results[j] = result;
            fprintf(latencyInformation, "lgpio_%%d\n", result);
        } else {
            printf("lgGpioClaimSigOUTput_FAILED_on_Pin_%%d\n", SigOUT);
            lgLineInfo_t lInfo;

            status = lgGpioGetLineInfo(h, SigOUT, &lInfo);

            if (status == LG_OKAY) {
                if (lInfo.lFlags & 1) {
                    printf("GPIO_in_use_by_the_kernel_");
                    printf("name=%s_s_user=%s\n", lInfo.name, lInfo.user);
                }
            }
        }
        lgGpioFree(h, SigOUT);
        lgGpiochipClose(h);
    } else
        printf("Unable_to_open_gpiochip_0\n");
}

```

```
// Calculates the average reads per second
for (int i = 0; i < CYCLES; ++i) {
    sum += results[i];
}
fprintf(latencyInformation, "Average_reads_per_second_%.0f\n", sum/CYCLES);
fclose(latencyInformation);
}
```

8.1.2 Testing Raspberry Pi GPIO Pin Read Speed with Python

As the code for running EPICS uses Python, testing GPIO pin speed within Python is necessary for relevancy to observe any noticeable or significant differences.

Ensure that pip3 is installed on your Raspberry Pi.

Install psutil:

```
pip install psutil
```

This package will allow us to get system information about the hardware and network used.

The following script below will execute in the same manner as the C code outputting a text file with system information and average read speeds as well as .csv files containing the input reading to the *python-no-epics* folder.

Code snippet line 129 - 153:

```
for i in range(CYCLES):
    f = open(r"./python-no-epics/python_no_epics_" + str(i + 1) + ".csv", "w")
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(pin, GPIO.IN)

    t0 = time.time()

    for j in range(LOOPS):
        singleTest.append(GPIO.input(pin))
    t1 = time.time()
    print("Done_a_loop")
    for k in range(LOOPS):
        f.write(str(singleTest[k]) + ",\n")
    result = (1.0 * LOOPS) / (t1 - t0)
    results.append(result)
    latencyInfo.write("RPI.GPIO\t{:>10.0f} ".format(result) + "\n")
    f.close()
    GPIO.cleanup()
    latencyInfo.write("Average_number_of_reads_per_second:_" + str(round(sum(results)/len(results))) + "\n")
    latencyInfo.close()
    print("All_done")
```

8.1.3 Testing Raspberry Pi GPIO Pin Read Speed with EPICS

Testing is also done to see the speed at which the Raspberry Pi can read from the GPIO pin and write into an EPICS db using caput.

Ensure pyepics is installed in addition to psutil (from previous test):

```
pip install pyepics
```

The script will execute the same GPIO pin read commands but with an additional write to EPICS. Results will also be saved to a text file with system information and read inputs will be written to .csv files in the *python-with-epics* folder.

Code snippet line 131 - 153:

```
for i in range(CYCLES):
    f = open(r"./python-with-epics/python_with_epics_" + str(i + 1) + ".csv", "w")
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(pin, GPIO.IN)

    t0 = time.time()

    for j in range(LOOPS):
        singleTest.append(GPIO.input(pin))
        epics.caput('rpi:message', '0')
    t1 = time.time()
    print("Done_a_loop")
    for k in range(LOOPS):
        f.write(str(singleTest[k]) + "\n")
    result = (1.0 * LOOPS) / (t1 - t0)
    results.append(result)
    latencyInfo.write("RPI.GPIO\t{:>10.0f}" format(result) + "\n")
    f.close()
    GPIO.cleanup()
latencyInfo.write("Average_number_of_reads_per_second:_ " + str(round(sum(results)/len(results)))+ "\n")
latencyInfo.close()
print("All_done")
```

8.2 Aliasing Testing for Raspberry Pi

Additional testing can be done on the Raspberry Pis to find the aliasing limit for data speed. Adjusting the function wave generator frequency, Pis can be tested with a square wave input between 1Hz in increments of 100Hz up to 1000 Hz and then increasing in increments of 1000 up to 40,000Hz.

Example: Our testing procedure ran the testing code across all the above frequencies for two Raspberry Pi 3s and two Raspberry Pi 2s.

Each data read from the GPIO pin (1 or 0) will be paired with a second value indicating number of nanoseconds since the test began. This provides a "timestamp" to plot out the results for visualization of the signal to be compared against the source.

8.2.1 Testing Raspberry Pi Aliasing in Python

The data read from the pins is written to .csv files and graphing this data will provide a visualization of the input signal being read and allow for determination of when the read signal begins to deviate from the source.

Code snippet line 130 - 155:

```
for i in range(CYCLES):
    f = open(r"./python-no-epics/python_no_epics_" + str(i + 1) + ".csv", "w")
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(pin, GPIO.IN)
```



```

t0 = time.time()
startTime = time.time_ns()
for j in range(LOOPS):
    singleTest.append(GPIO.input(pin))
    singleTestTime.append(time.time_ns() - startTime)
t1 = time.time()
print("Done_a_loop")
for k in range(LOOPS):
    f.write(str(singleTest[k]) + "," + str(singleTestTime[k]) + ",\n")
result = (1.0 * LOOPS) / (t1 - t0)
results.append(result)
latencyInfo.write("RPi.GPIO\t{:>10.0f}".format(result) + "\n")
f.close()
GPIO.cleanup()
latencyInfo.write("Average_number_of_reads_per_second:" + str(round(sum(results)/len(results)))+ "\n")
latencyInfo.close()
print("All_done")

```

8.2.2 Testing Raspberry Pi Aliasing in EPICS

As EPICS is expected to result in lower read speeds, additional aliasing testing can be performed with writing to EPICS to observe the aliasing limit for data acquisition in EPICS.

Our testing procedure followed the same as without EPICS, however as average read speed was observed to be significantly slower from our latency test, our frequencies only ranged from 1Hz to 1000Hz in increments of 100.

The code follows the same as section 3.4 but with writes to EPICS.

Code snippet line 130 - 157:

```

for i in range(CYCLES):
    f = open(r"./python-with-epics/python_with_epics_" + str(i + 1) + ".csv", "w")
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(pin, GPIO.IN)

    singleTestTime = []
    singleTest = []

    t0 = time.time()
    startTime = time.time_ns()
    for j in range(LOOPS):
        singleTest.append(GPIO.input(pin))
        epics.caput('rpi:message', '0')
        singleTestTime.append(time.time_ns() - startTime)
    t1 = time.time()
    print("Done_a_loop")
    for k in range(LOOPS):
        f.write(str(singleTest[k]) + "," + str(singleTestTime[k]) + ",\n")
    result = (1.0 * LOOPS) / (t1 - t0)
    results.append(result)
    latencyInfo.write("RPi.GPIO\t{:>10.0f}".format(result) + "\n")
    f.close()
    GPIO.cleanup()
    latencyInfo.write("Average_number_of_reads_per_second:" + str(round(sum(results)/len(results)))+ "\n")
    latencyInfo.close()

```

```
print("All_done")
```

9 Python Scripts to Control IOCs

In order to control the RaspberryPi's to act as IOC devices, we need to give them some code that will allow them to control GPIO pins, take in data, and write/read to EPICS.

We will be using Python to run these scripts on the Pi's. This is because many Physics students already know Python for data science reasons. As well, the RPI.GPIO library is very useful for interfacing with the GPIO pins themselves.

This guide will provide some instruction how to take our scripts, modify them, and make your own controllable IOC.

9.1 Different Types of IOCs

For directly running any of these files from a Pi you can do so by picking the type of PV in

epics-gui-triumf/src/scripts/IOC/

From there the type of PV can be found. You can run these by using the following code. (Note the PV name is specified as an argument when running the file)

```
~/epics-gui-triumf/src/scripts/IOC$ python istf_fc.py ISTF:FC1
```

If you do not enter this argument when running the PV, the default will be [PV]0

```
11 # ID From System args
12 pvID = ""
13 try:
14     pvID = str(sys.argv[1])
15 except:
16     pvID = "ISTF:FC0"
```

Once inside any of the files you can change the PVs (carefully, as some PVs in certain files are specifically coded for certain IOC setups).

```
20 #####
21 # EDIT PVS and GPIO pins HERE
22
23 # PVs Used IN ORDER OF GPIO_LIST
24 pv0 = PV(pvID + ":SOL")
25 pv1 = PV(pvID + ":IN")
26 pv2 = PV(pvID + ":OUT")
27 pv3 = PV(pvID + ":HWR")
28
29 # List of PV names in order for this device
30 pvList= [pv0.pvname, pv1.pvname, pv2.pvname, pv3.pvname]
31 gpioList = [5, 6, 13, 19] # List of GPIO pins for this device
32 gpioOutputList = [True, False, False, True] # False if INPUT / True if Output
```

9.1.1 Binary Input from Device (Sensor)

Binary Input devices, are used to check PVs that measure things like IN or OUT of a Faraday Cup

To start, we create a function that is checked every few seconds to see if the GPIO pins have changed values on whatever sensor they are measuring.

```

72  def checkBinarySensor(pv, index):
73
74      pin = gpioList[index]
75      boolPinOn = (GPIO.input(pin) == GPIO.HIGH)
76
77      if boolPinOn != previousList[index]:
78          pv.put((0, 1)[boolPinOn])

```

We then add this function to the main method at the bottom that loops infinitely and waits a few seconds each loop.

```

129  def loop():
130      try:
131          while True:
132
133              # Check for changes to the binary sensors
134              checkBinarySensor(pv1, 1)
135              checkBinarySensor(pv2, 2)
136
137              # Wait a long amount of secs
138              time.sleep(sleepTimeLong)

```

9.1.2 Binary Output to Device (Controller)

Binary Output devices, are used to output a signal to a device every time a PV in EPICS changes.

To start, we create a function that handles how we handle that PVs logic.

Here is a generic function that is used to turn ON/OFF a device to match the EPICS PV. This has NO interlock logic.

```

35  # Callback Functions for PV/GPIO logic
36  def binaryPVChanged(pvname=None, value=None, char_value=None, **kw):
37
38      boolTurnON = (value == 1)
39      index = pvList.index(pvname)
40
41      if not boolTurnON:
42          print(pvname + ": Change Detected - Setting OFF")
43      else:
44          print(pvname + ": Change Detected - Setting ON")
45
46      controlGPIO(gpioList[index], boolTurnON)
47

```

We then attach this function directly to a PV. This is done at the bottom of the setup function.

```

56 #####
57 # SET the interlock devices and interlocks
58
59 pv0.add_callback(binaryPVChanged)
60 pv1.add_callback(binaryPVChanged)
61 pv2.add_callback(binaryPVChanged)
62
63 # BE CAREFUL EDITING PAST HERE!
64 #####

```

9.1.3 Binary Output to Device (Interlocks)

Another type of Binary Output device is one that will only output a signal on certain conditions. These are defined as interlocked devices.

For example here, the SOL PV can only be turned on when HWR is ON.

If for any reason the HWR is on and SOL is attempted to be turned on, we need to not output a signal and reset the PV to be OFF.

Here are two custom function that handle this logic.

```

47 def SOLPVChanged(pvname=None, value=None, char_value=None, **kw):
48
49     boolTurnON = (value == 1)
50     index = pvList.index(pvname)
51
52     #Check status of HWR to check if flowing
53     boolHWRFlowing = (pv3.get() == 1)
54
55     if not boolHWRFlowing and boolTurnON:
56         print(pvname + ": Change Detected - Water NOT Flowing")
57         print(" > Not turning on SOL, Turn On Water First")
58         controlGPIO(gpioList[index], False)
59         Thread(target=turnOffSOL).start()
60
61
62     elif boolHWRFlowing and boolTurnON:
63         print(pvname, ": Change Detected - Water IS Flowing")
64         print(" > Turning on SOL!")
65         controlGPIO(gpioList[index], True)
66
67     else:
68         print(pvname, ": Change Detected - Turning OFF")
69         controlGPIO(gpioList[index], False)

```

```

71 def HWRPVChanged(pvname=None, value=None, char_value=None, **kw):
72
73     boolTurnON = (value == 1)
74     index = pvList.index(pvname)
75
76     #Check status of SOL to check if on
77     boolSOLOn = (pv0.get() == 1)
78
79     if boolSOLOn and not boolTurnON:
80         print(pvname + ": Change Detected - Water NOT Flowing")
81         print(" > Turning OFF SOL for Safety")
82         Thread(target=turnOffSOL).start()
83
84     elif not boolSOLOn and not boolTurnON:
85         print(pvname + ": Change Detected - Turning OFF")
86
87     else:
88         print(pvname, ": Change Detected - Turning ON")
89
90     controlGPIO(gpioList[index], boolTurnON)

```

Note that Python Threads are used here.

This is because of how PyEpics handles the .put() call, which is a "blocking" call. This can cause a freeze in our program if we try to change a PV from inside a function that responds to a PV changing.

We use this thread to make sure our IOC keeps running and checking other PV's

The function that the thread runs should be simple and handle just one item. Like so:

```

def turnOffSOL():
    pv0.put(0)

```

9.1.4 Analog Input from Device (Sensor)

Reading from an Analog sensor is similar to reading from a binary sensor.

We first need some preliminary code to set up reading from an ADC.

You also will need to run : `sudo apt-get install python-smbus`

```

5  import smbus
6
7  class ADCDevice(object):
8      def __init__(self):
9          self.cmd = 0
10         self.address = 0
11         self.bus=smbus.SMBus(1)
12         # print("ADCDevice init")
13
14     def detectI2C(self,addr):
15         try:
16             self.bus.write_byte(addr,0)
17             print("Found device in address 0x%x"%(addr))
18             return True
19         except:
20             print("Not found device in address 0x%x"%(addr))
21             return False
22
23     def close(self):
24         self.bus.close()
25
26 class PCF8591(ADCDevice):
27     def __init__(self):
28         super(PCF8591, self).__init__()
29         self.cmd = 0x40 # The default command for PCF8591 is 0x40.
30         self.address = 0x48 # 0x48 is the default i2c address for PCF8591 Module.
31
32     def analogRead(self, chn): # PCF8591 has 4 ADC input pins, chn:0,1,2,3
33         value = self.bus.read_byte_data(self.address, self.cmd+chn)
34         value = self.bus.read_byte_data(self.address, self.cmd+chn)
35         return value
36
37     def analogWrite(self,value): # write DAC value
38         self.bus.write_byte_data(self.address, self.cmd, value)
39
40 class ADS7830(ADCDevice):
41     def __init__(self):
42         super(ADS7830, self).__init__()
43         self.cmd = 0x84
44         self.address = 0x4b # 0x4b is the default i2c address for ADS7830 Module.
45
46     def analogRead(self, chn): # ADS7830 has 8 ADC input pins, chn:0,1,2,3,4,5,6,7
47         value = self.bus.read_byte_data(self.address, self.cmd|(((chn<<2 | chn>>1)&0x07)<<4))
48         return value
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74 #####
75 # Callback Functions for PV/GPIO logic
76 def setup():
77
78     global adc
79     if(adc.detectI2C(0x48)): # Detect the pcf8591.
80         adc = PCF8591()
81     elif(adc.detectI2C(0x4b)): # Detect the ads7830
82         adc = ADS7830()
83     else:
84         print("No correct I2C address found, \n"
85             "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
86             "Program Exit. \n");
87         exit(-1)
88

```

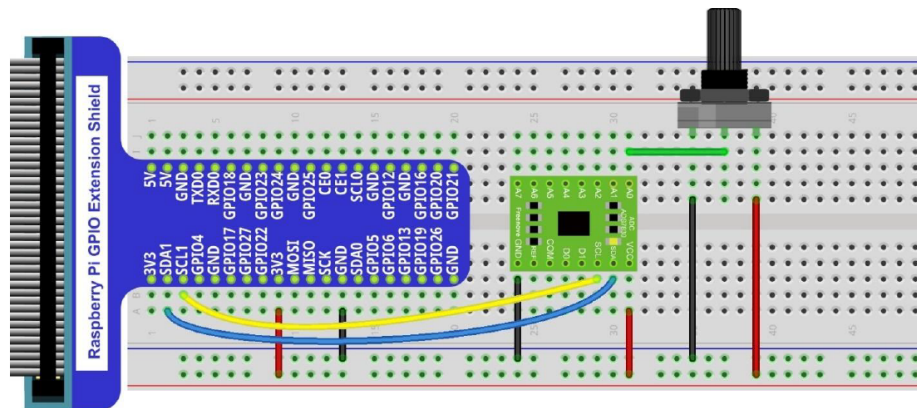
Then in the main loop we just read the channel in which the ADC is listening.

```

107 def checkAnalogSensor(pv):
108
109     value = adc.analogRead(0)    # read the ADC value of channel 0
110     voltage = value / 255.0 * 3.3 # calculate the voltage value
111     pv.put(voltage) # Write voltage to EPICS:
112
113
114 def loop():
115     try:
116         while True:
117
118             # Write the voltage to EPICS
119             checkAnalogSensor(pv0)
120
121             # Wait a short amount of secs
122             time.sleep(sleepTimeShort)
123

```

The ADC Wiring will look something like this, where the ADC is reading in some analog signal to the SDA1 and SCL1 pins



We now have scripts for all of the device types we would need to read/write to.

For the full code for these demos look at:

<https://github.com/MattiasHenders/epics-gui-triumf>

10 High-Level Application - EPICS Through your Web Browser

A High-level application (HLA) has also been developed alongside the RPI EPICS implementation as a proof of concept. This application allows users to monitor PV values of the PIs through their web browsers in real time.

This chapter will describe the steps taken to develop the HLA, how the HLA works, and how to run it.

10.1 HLA Tech Stack and Development

10.1.1 Tech Stack

The tech stack is as follows:

Server - Python, Flask

Client - HTML, JavaScript

The HLA is developed using a Flask server to serve HTML files to a connected client. These HTML files display information about the RPIs in real time. As HTML files are static the real time monitoring has been achieved with a module called flask-socketio, which allows events to be asynchronously sent to the client side JavaScript from the Flask server without re-serving the HTML page.

10.1.2 Development

The resources used to create this Flask application are the first 3 lessons in this Flask tutorial <https://pythonise.com/series/learning-flask>

HTML and JavaScript knowledge is assumed to be pre-existing

I recommend reading the following sections while reference the HLA source code for better understanding.

10.1.2.1 `__init__.py`

`__init__.py` pulls application components together and then serves it as a package `'__init__.py'`

10.1.2.2 `run.py`

`run.py` handles the execution of the Server. Think of this like the the app component in a React application

10.1.2.3 `views.py`

Views contain routes which represent an endpoint of an application. An example endpoint could be : <https://example.com/someEndpoint>

Each route contains logic for serving data to an endpoint and ultimately returns an HTML page (see next subsection) along with a JSON object with data that can be interpreted on the client side.

`views.py` currently only contains a single route alongside quite a bit of application logic. This application logic can (and likely should) be shifted to a new python module, especially if more routes are added.

10.1.2.4 `templates/index.html`

The templates folder contains files that are served by routes. The route for `"/"` in `views.py` renders the `templates/index.html` file in it's return statement. This file contains the actual HTML and JS that power the client side experience over the web.

10.1.3 Running the HLA

10.1.3.1 Installation

Clone the `epicsgui-triumf` repo

- cd into the high-level-application folder by typing: `cd epics-gui-triumf`

```
pip install -r requirements.txt
```

```
pip install flask-socketio
```

```
pip install pyepics
```

10.1.3.2 Running the application

```
flask run --host 0.0.0.0
```

10.1.3.3 Troubleshooting

If pip is not installed run `sudo apt get install python3-pip`

If pyepics is not installed run `pip install pyepics`

If you encountered an issues with Installation make sure that you're using pip3 and python 3 by default. The process to do this depends on your environment but information can be found online.

11 SD Card Images and Flashing

11.1 Creating an Image

The SD cards in the RPI's on the PI tower have all been flashed to use the disk image. This creates a consistent experience across the RPI's.

To export an image from an SD card do the following:

Download and run the Win32DiskImager from: <https://sourceforge.net/projects/win32diskimager/>

For the "Image File" select the directory and name of the image file that you are going to create.

Insert the SD card that you want to copy, then select that drive as the "Device".

Select the "Read Only Allocated Partitions" check box.

Click the "Read" button to create the image.

Once the image file is created, check to make sure that it has a ".img" extension. If it doesn't manually enter the extension on the file name.

11.2 Flashing an Image to an SD card

To flash a disk image to an SD card:

Download and run the Raspberry Pi Imager from: <https://www.raspberrypi.com/software/>

Click on "Operating System > Custom Image" and select the image file that you want to use.

Insert an SD card, click on "Storage", and select the SD card as your target storage device.

Click the "Write" button and wait for the image to write and verify on the SD card. begin

12 Common Issues

12.0.1 Ubuntu For Windows

If edm will not start

- Ensure that Xming is running
- Ensure that you are a super user on the Ubuntu command line (sudo -i)

12.0.2 IOCs not Connecting to Server

In order for your RPIs to connect properly with the EPICS server, ensure that the server is started before the Raspberry PIs by using the start_server.sh script in the root directory of the ER-PCC project. Turn off any RPIs currently running by powering them off.

Run the start script using the following command:

```
bash /er-pcc/start_server.sh
```

Your terminal should then inform you that two screens are now running.

```
STARTING EPICS SERVER
There are screens on:
  115741..DESKTOP-E6DMBR0 (05/24/22 11:13:53) (Detached)
  115741..DESKTOP-E6DMBR0 (05/24/22 11:13:53) (Detached)
2 Sockets in /run/screen/S-esado.
SERVER STARTED
```

In order for the Raspberry PIs to correctly start their IOCs they need to first be able to connect to the server, this is to prevent multiple IOCs from serving the same PVs.

12.0.3 Server Start Script not Running Correctly

If you are trying to run the EPICS server and are seeing an error message stating that you do not have permission to access /run/screen; enter the command:

```
sudo chmod 777 /run/screen
```

12.0.4 Unable to Reassign IOCs Using PI Status Page

If your PI status page is unable to reassign IOCs and you are getting an error message in your EDM terminal that you are unable to authenticate a host; enter the command:

```
nano /etc/ssh/ssh_config
```

Navigate down the page and uncomment the line 'StrictHostKeyChecking' and change its value from 'ask' to 'no'. Save and close this file.

Next restart your ssh service by typing the command:

```
sudo service ssh restart
```

Your server should now be able to correctly transfer files to other devices on the network. Ensure you restart both the server and then your RPIs after doing this.

12.0.5 Problems Reassigning IOCs

If you notice that you are unable to re-assign an IOC, or that when you re-assign an IOC its python script and/or ID are not correctly assigned; check the error log of the Ubuntu window that is running your EDM instance, or the Ubuntu window you used to call assign_db.sh.

If you are seeing error messages in your console suggesting that a file could not be found or a directory does not exist; double check that within the assign_db.sh file the correct path to the files being transferred is given, and that the files you are trying to find are in the right directory. All python scripts need to be in the er-pcc/src/scripts/IOC directory, and all .db files need to be in the er-pcc/src/database/IOC directory.

If you see an error message suggesting that you do not have the correct read or write permissions, ensure that the user you have run the edm interface with has sudo access by first logging in as the root user using 'sudo -i' and then using the command 'sudo adduser <username> sudo'. This will add that user to the linux sudoers group and should give you the correct read and write access to the files you need.

If you are still having the issue where when you re-assign an IOC and its new ID does not persist the next time the IOC is started; try re-starting the Raspberry PI and re-connecting it.

12.0.6 Difficulties Clicking a Specific Object in EDM

If you are experiencing problems selecting an item that is stacked on top of other items in EDM, hold down your left control while clicking on the object stack to be able to select through each of the items consecutively.

If an item appears to be un-selectable, it may be because it is in a group. If you want to individually select items, use your middle click on an item once you have selected it and click 'ungroup' to once again be able to individually select the item. You may want to regroup the items you ungrouped after doing this.

12.0.7 Unable to pull/push to the Gitlab Repo

If you find that you are unable to pull from or push to the Gitlab repo, ensure that you have the correct user permissions. If attempting to do this on the server computer you may have to perform this as the root user. Log in as the root user by typing 'sudo -i' and enter the correct password. After that navigate to /opt/er-pcc and execute your desired git command(s).

A Other Tools and Setup for EPICS

A.1 Installing a Raspberry Pi Emulator

Download <https://sourceforge.net/projects/rpiqemuwindows/>
Unzip the download
Run the run.bat
Follow the README and make sure that SSH is enabled when the menu shows up.

The default login is:
user: pi
password: raspberry

A.2 Installing Pre-built EPICS tools for Windows

There are a number of pre-compiled tools that can be used in combination with EPICS to chart, monitor and output information by using channel access. The list of available tools can be found at <https://epics.anl.gov/distributions/win32/index.php>.

In order to run these extensions you will need to set up a PC X server. Navigate to <https://sourceforge.net/projects/xming/> to download and install Xming using the default settings.

You can later use this Xming server to run EPICS windows tools.

Navigate to <https://epics.anl.gov/distributions/win32/index.php>
Under Downloads click EPICSWindowsTools1.44-x64.msi

When the download finishes unzip it.

Each of the executables provided in the now unzipped EPICSWindowsTools contains one of the aforementioned EPICS extensions and can be run by double clicking on them, provided you are currently running your PC X server.

A.2.1 Graphing with StripTool

StripTool is a tool to PV plot data over time that integrates right into EPICS. It can be downloaded here:

<https://epics.anl.gov/distributions/win32/index.php>

It can be used to plot data made available through channel access.

In order to use this, you will need at least one device connected to your network that is outputting data using caput in order to have data to track.

To run the StripTool, ensure that your PC X server is running and then double click the executable for StripTool.

In the window that opens, enter the process variable that you want to track in the box next to "Plot New Signal". For example "temperature:water".

You can graph multiple process variables by entering several process variables in the "Plot New Signal" text box.

If changes are being made to the values of the process variables you are tracking you should be able to see the plot update to show this.

A.3 Test Databases in EPICS

A.3.1 Setting up a Test Database

We need to test the database so EPICS can update the data that the server reads.

(I will label this terminal for now as terminal-1:)

Do not type in terminal-1:

```
terminal-1: $HOME/EPICS/epics-base/db
```

```
terminal-1: nano test.db
```

Add the following text to this new file:

```
record(ai, "temperature:water")
{
  field(DESC, "Water temperature in the fish tank")
}
```

```
terminal-1: softIoc -d test.db
```

```
terminal-1: epics> dbI
```

You should see:

temperature:water

Now open a new terminal (I will label for now as terminal-2:)

Do not type in terminal-2:

```
terminal-2: cd $HOME/EPICS/epics-base/db
```

```
terminal-2: softIoc -d test.db
```

```
terminal-2: epics> dbI
```

```
terminal-2: epics> exit
```

Now we will test writing and reading to the database.

```
terminal-2: caget temperature:water
```

temperature:water 0

```
terminal-2: caget temperature:water 89
```

Now open a new terminal (I will label for now as terminal-3:)

Do not type in terminal-3:

```
terminal-3: cd $HOME/EPICS/epics-base/db
```

```
terminal-3: softIoc -d test.db
```

```
terminal-3: epics> dbI
```

```
terminal-3: epics> exit
```

```
terminal-3: caget temperature:water
```

temperature:water 89

You can now close terminal-2 and terminal-3

We've now tested the database and shown we can read and write successfully!

B ER-PCC Repo Directory Map

This appendix lists in detail some information about the various directories in the ER-PCC project.

If working from the server the directory containing these files will be located in /opt. Use the command "cd /opt" to navigate there.

B.0.1 *er-pcc*

This is the root of the project, all of the necessary files can be found in the subdirectories of this directory.

B.0.2 *er-pcc/high-level-application*

This directory contains all of the files related to our high level application implementation

B.0.3 *er-pcc/src/database*

This directory contains all of the files pertaining to database configuration. It has two important subdirectories: 'IOC' and 'server'

The er-pcc/src/database/IOC directory contains all of the files used for .db configurations on each of the IOC types. Additionally it contains the IOC_DEVICE_ASSOCIATIONS.txt file where you can declare another IOC device to associate to a set of .db and .py files

The er-pcc/src/database/server directory contains two database files, one which is the 'iocstatus.db' run by the server on startup and allows the server to be able to correctly represent which IOCs are active or inactive.

The 'unittest.db' file in this directory is a file created to be used by one of the test beds in the er-pcc/src/tests directory.

B.0.4 *er-pcc/src/scripts*

This directory contains all of the scripts used by both the IOCs and the server. The python scripts used by each IOC type can be found in the er-pcc/src/scripts/IOC directory.

Within the er-pcc/src/scripts/server directory you can find the assign_db.sh script which is used by the server to be able to re-assign one IOC to another.

B.0.5 *er-pcc/src/screens/istf_screens*

The EDM views we created throughout the project can be found within this directory. If you want to start running the EDM interface directly from the server PC you can do so in this directory by typing the command 'edm' and opening the main_menu.edl using the window that appears afterwards.

B.0.6 *er-pcc/src/tests*

This directory contains the tests that we wrote to verify that our code was working correctly and to test the performance of the Raspberry PIs.

For our IOC reassignment scripts we created a suite of unit tests that can be found in file_transfer_tests.py. You should be able to run those tests by typing the command 'python3 -m unittest file_transfer_tests.py' while inside the tests directory. Ensure your server is running and at least 1 RPI is connected to the network before running them.