

EPICS System Setup Guide — Table of Contents

1. Raspberry Pi OS Setup
 - 1.1. Installing Raspberry Pi Imager
 - 1.2. Flashing 64-bit Raspberry Pi OS
 - 1.3. Enabling SSH, Wi-Fi, and Headless Config
 - 1.4. First Boot and SSH Access
2. System Configuration and Package Setup
 - 2.1. Updating the System (apt, firmware)
 - 2.2. Installing Essential Development Tools
3. EPICS Base Installation
 - 3.1. Downloading and Extracting EPICS Base
 - 3.2. Editing CONFIG_SITE and CONFIG Files
 - 3.3. Building EPICS Base
 - 3.4. Verifying EPICS Commands (caget, caput, etc.)
4. Asyn Driver Setup
 - 4.1. Downloading asyn Module
 - 4.2. Linking to EPICS Base
 - 4.3. Compiling asyn
 - 4.4. Running a Test IOC
5. Protocol Driver Setup (FINS / Modbus)
 - 5.1. Choosing a Communication Protocol
 - 5.2. (For FINS) Setting Up ISIS EPICS-FINS Driver
 - 5.3. Fixing Driver Memory Area Mapping (W Area Access)
 - 5.4. Creating the IOC for FINS Communication
 - 5.5. Testing PLC Reads/Writes
6. IOC Database and Records
 - 6.1. Writing EPICS .db Files for PLC Variables
 - 6.2. Handling Bit-Level Access and Trigger Logic
 - 6.3. Debugging Common IOC/CA Issues
 - 6.4. Using camonitor for Live Monitoring
7. Python Logging Scripts
 - 7.1. Creating Python Scripts to Log PVs
 - 7.2. CSV Output and Timestamping
 - 7.3. Wrapping Scripts with Bash for Automation
8. CS-Studio Setup
 - 8.1. Installing CS-Studio (on Laptop/Desktop)

- 8.2. Connecting to Pi's IOC
- 8.3. Designing a Simple GUI with BOB Files
- 8.4. Adding Dynamic Colors and Triggers
- 8.5. Testing and Deployment


9. Final System Checks

- 9.1. Auto-starting IOC on Boot
- 9.2. Network/IP Configuration Notes
- 9.3. Common Pitfalls and Fixes
- 9.4. Summary of Working Components



1. Raspberry Pi OS Setup

This section walks through preparing a microSD card with the correct 64-bit Raspberry Pi OS for EPICS use, including enabling SSH and network connectivity for a headless setup.

1.1 Installing Raspberry Pi Imager

1. On your laptop or desktop, go to:
 <https://www.raspberrypi.com/software>
2. Download the appropriate installer for your operating system (Windows, macOS, or Linux).
3. Follow the installation prompts to install **Raspberry Pi Imager**.




1.2 Flashing 64-bit Raspberry Pi OS

1. Launch **Raspberry Pi Imager**.
2. Click “**Choose OS**” → select:
 **Raspberry Pi OS (64-bit)** under the “**Raspberry Pi OS (other)**” section.
3. Click “**Choose Storage**” → select the correct microSD card (minimum 16 GB recommended).
4. Before clicking “Write,” click the **gear icon**  (bottom-right) to preconfigure the OS image (see next section).

1.3 Enabling SSH, Wi-Fi, and Headless Config

In the  **Advanced Options** menu:

Enable the following:

-  **Set hostname** → e.g., **kimball-gun**
-  **Enable SSH** → Use password authentication
-  **Set username and password**
 - **Username:** **Hutch**

- Password: #####
- ✓ **Configure Wi-Fi** (SSID, password, country)
- ✓ **Set locale and timezone** (e.g., US / New York)

Click **Save**, then click **Write** to flash the image.

⚠ Warning: This will erase all data on the selected SD card.

1.4 First Boot and SSH Access

1. After the image has been written and verified, eject the SD card.
2. Insert it into the Raspberry Pi and power it on.
3. If Wi-Fi was configured, the Pi will connect automatically.
4. To access the Pi via terminal, open windows powershell and run:

ssh [Hutch@kimball-gun.local](ssh:Hutch@kimball-gun.local)

5. Accept the SSH fingerprint prompt and enter your password.

2. System Configuration and Package Setup

Once the Raspberry Pi has booted and you have SSH access, the next step is to update the system and install required packages for building and running EPICS.

2.1 Updating the System

Start by updating the system's package index and installing the latest packages:

Command:

```
sudo apt update
sudo apt upgrade -y
```

(Optional but recommended):

```
sudo apt install rpi-update -y
sudo rpi-update
```

⚠ Reboot after a firmware update:

```
sudo reboot
```

2.2 Installing Essential Development Tools

Install the build tools and libraries needed for EPICS Base and modules:

Command:

```
sudo apt install -y build-essential git wget curl libreadline-dev
libncurses-dev libpcre3-dev libssl-dev libxml2-dev ca-certificates
pkg-config cmake unzip
```

📌 If on **Debian 12 (Bookworm)**: `libssl-dev` may give version issues when building some EPICS modules (see next section).

3. EPICS Base Installation

This section covers downloading, building, and verifying EPICS Base on the Raspberry Pi.

3.1 Downloading and Extracting EPICS Base

1. Create a directory to store EPICS software (we'll use `/opt/epics` as the standard location):

```
sudo mkdir -p /opt/epics
sudo chown $USER:$USER /opt/epics
cd /opt/epics
```

2. Download EPICS Base (example uses 7.0.8):

```
wget https://epics.anl.gov/download/base/base-7.0.8.tar.gz
tar -xzf base-7.0.8.tar.gz
mv base-7.0.8 base
```

3.2 Editing CONFIG Files

1. Go into the configuration directory:

```
cd /opt/epics/base/configure
```

2. Open `CONFIG_SITE`:

```
nano CONFIG_SITE
```

- Uncomment and set this line:
`EPICS_HOST_ARCH = linux-aarch64`

(You can confirm your architecture later with `./startup/EpicsHostArch` if unsure.)

3. Save and exit (`Ctrl+O`, `Enter`, then `Ctrl+X`)
-

3.3 Building EPICS Base

1. Navigate back to the base directory:

```
cd /opt/epics/base
```

2. Run the build:

```
make -j$(nproc)
```

This may take a few minutes. If successful, binaries like `caget`, `caput`, and `softIoc` will be created under:

```
/opt/epics/base/bin/linux-aarch64
```

3.4 Verifying EPICS Commands

To quickly test if the build was successful:

1. Run:

```
~/opt/epics/base/bin/linux-aarch64/caget -h
```

If the help text appears, EPICS Base was built correctly.

2. Now go back and **add environment variables** (Section 2.3 if skipped earlier):

```
nano ~/.bashrc
```

Add these lines at the bottom:

```
export EPICS_BASE=/opt/epics/base
export EPICS_HOST_ARCH=linux-aarch64
export PATH=$EPICS_BASE/bin/$EPICS_HOST_ARCH:$PATH
```

Then apply them:

```
source ~/.bashrc
```

Try running:

```
caget -h
```

You should now see the help menu from anywhere in the terminal.

At this point:

- EPICS Base is installed and working
- Core EPICS commands are in your PATH
- You're ready to build and use modules like **asyn**

4. Asyn Driver Setup

The **asyn** module is a required EPICS support module for serial, TCP/IP, and driver-layer communication — and it's needed for most protocol drivers like Modbus or FINS.

4.1 Downloading asyn Module

1. Navigate to your EPICS modules directory:

```
cd /opt/epics
mkdir modules
cd modules
```

2. Clone the official asyn repository:

```
git clone https://github.com/epics-modules/asyn.git
cd asyn
```

3. (Optional) Checkout a known stable release (recommended for production):

```
git checkout R4-43
```

4.2 Linking to EPICS Base

Make sure the `EPICS_BASE` environment variable is set (see Section 3.4). You can test:

```
echo $EPICS_BASE
→ Should return /opt/epics/base
```

If not, re-source your `.bashrc`:

```
source ~/.bashrc
```

4.3 Building asyn

1. Go into the `asyn` directory:


```
cd /opt/epics/modules/asyn
```

2. Build the module:

```
make -j$(nproc)
```

If the build succeeds, binaries and libraries will be generated under:

```
/opt/epics/modules/asyn/lib/linux-aarch64 and
```

```
/opt/epics/modules/asyn/bin/linux-aarch64
```

4.4 Running a Test IOC (Optional Sanity Check)

1. Go into the `testIPServer` example:

```
cd /opt/epics/modules/asyn/testIPServer
```

2. Run the example IOC:

```
../../../../bin/linux-aarch64/testIPServer st.cmd
```

If it starts up and prints `iocInit` without errors, the module works properly.

You can exit the IOC with `Ctrl+C`.

At this point:

- The **asyn** module is installed and built
- You're ready to build protocol drivers that depend on it (e.g., FINS, Modbus)
- Optional test IOC confirms functionality

5. FINS Protocol Driver Setup (ISIS EPICS-FINS)

This section walks through installing, configuring, and running the ISIS Computing Group's EPICS-FINS driver to communicate with an Omron PLC over UDP using the FINS protocol.

5.1 Cloning the ISIS EPICS-FINS Driver

1. Navigate to your EPICS modules directory:

```
cd /opt/epics/modules
```

2. Clone the driver repository:

```
git clone https://github.com/ISISComputingGroup/EPICS-FINS.git  
cd EPICS-FINS
```

3. (Optional) Checkout the latest stable version or tag:

```
git checkout main
```

5.2 Configuring Makefiles and Building the Driver

1. Open the top-level Makefile:

```
nano Makefile
```

2. Confirm the paths are set correctly:

```
EPICS_BASE = /opt/epics/base  
ASYN = /opt/epics/modules/asyn
```

3. Save and exit the file.
4. Build the module:

```
make -j$(nproc)
```

After successful compilation, you should see `bin/linux-aarch64` and `lib/linux-aarch64` directories created.

5.3 Enabling W-Memory Area Access (Driver Patch)

By default, the FINS driver does not support the Omron W memory area. You must patch the source code:

1. Open the source file:

```
nano src/finsUDP.cc
```

2. Find the section where memory area types are parsed (look for "D", "CIO", etc.)
3. Add the following condition:

```
else if (type == "W") area = 0xB1;
```

4. Save the file and rebuild:

```
make clean  
make -j$(nproc)
```

This enables support for reading and writing to Work Area (W) memory.

5.4 Creating the IOC and Configuring st.cmd

1. Go to the IOC directory:

```
cd /opt/epics/EPICS-FINS/CS-RF-IOC-01
```

2. Open the startup command script:

```
nano st.cmd
```

3. Confirm or modify it to include the following:

```
< envPaths  
cd "${TOP}"
```

```
dbLoadDatabase "dbd/CS-RF-IOC-01.dbd"  
CS_RF_IOC_01_registerRecordDeviceDriver pdbbase  
finsConfigurePort("LN2WR", "192.168.1.10", 9600, 0, 0, 0, 0)  
dbLoadRecords("db/plc.db")  
iocInit
```

4. Replace `192.168.1.10` with the actual IP address of your PLC.

5.5 Running the IOC and Verifying Communication

1. Navigate to the IOC bin directory:

```
cd /opt/epics/EPICS-FINS/CS-RF-IOC-01
```

2. Run the IOC:

```
./bin/linux-aarch64/CS-RF-IOC-01 st.cmd
```

3. If successful, you will see EPICS initialization messages ending with `iocInit`.
4. You can now test your PLC tags using `caget`, `caput`, or `camonitor`.

✓ At this point:

- The FINS driver is fully installed and supports W memory
- Your IOC is running and communicating with the Omron PLC
- You're ready to configure `.db` records and map them to PLC addresses

6. IOC Database and Records

This section explains how to define `.db` files to read from and write to Omron PLC memory addresses using the FINS protocol. It covers full register access, individual bit control, and chaining logic for reliable operation.

6.1 Writing EPICS .db Records for PLC Registers

To read or write a full 16-bit W register (e.g. W4):

1. Create or open a database file:

```
cd /opt/epics/EPICS-FINS/CS-RF-IOC-01/db
nano plc.db
```

2. Add the following record to read from W4:

```
record(longin, "PLC:W4")
{
  field(DESC, "Read full 16-bit W4 register")
  field(DTYP, "asynInt32")
  field(INP, "@asyn(LN2WR 4 0) FINS_WR_READ")
  field(SCAN, "2 second")
}
```

3. Add the following record to write to W4:

```
record(longout, "PLC:W4_Write")
{
  field(DESC, "Write full 16-bit W4 register")
  field(DTYP, "asynInt32")
  field(OUT, "@asyn(LN2WR 4 0) FINS_WR_WRITE")
  field(PINI, "NO")
}
```

6.2 Reading and Setting Individual Bits

To access specific bits (e.g. Bit 5 of W4):

1. Bit Readback (shows value of bit 5):

```
record(bi, "PLC:W4_BIT5_RBV")
{
```

```

field(DESC, "Read W4 bit 5")
field(DTYP, "asynInt32")
field(INP, "PLC:W4 MS")
field(ZNAM, "OFF")
field(ONAM, "ON")
field(SCAN, "2 second")
}

```

2. Bit Set Trigger (write logic):

```

record(bo, "PLC:W4_BIT5_Set")
{
  field(DESC, "Set W4 bit 5")
  field(DTYP, "asynInt32")
  field(OUT, "@asynMask(LN2WR 4 0 0x20) FINS_WR_SET")
  field(HIGH, "1")
  field(VAL, "0")
}

```

Note: $0x20 = 2^5 = \text{bit } 5$

6.3 Using FLNK and Trigger Logic

To make setting a bit trigger a process chain:

```

record(bo, "PLC:W4_BIT5_Trigger")
{
  field(DESC, "Trigger record chain")
  field(DOL, "1")
  field(OUT, "PLC:W4.PROC")
  field(FLNK, "PLC:W4_BIT5_Proc")
}

record(bo, "PLC:W4_BIT5_Proc")
{
  field(DESC, "Actual bit set")
  field(OUT, "@asynMask(LN2WR 4 0 0x20) FINS_WR_SET")
  field(VAL, "1")
  field(PINI, "NO")
}

```

This structure:

- Forces an update of `W4` from the PLC first
 - Then sets bit 5 via a clean second record
 - Avoids accumulated bit corruption from repeated writes
-

6.4 Debugging with `camonitor` and `caput`

To view live PV changes:

```
camonitor PLC:W4
camonitor PLC:W4_BIT5_RBV
```

To manually test a write:

```
caput PLC:W4_Write 32
caput PLC:W4_BIT5_Set 1
```

At this point:

- You can read and write both full words and individual bits on your Omron PLC
- Bit masking and update logic ensures consistent behavior
- You're ready to move to CS-Studio for visualization

7. Python Logging Scripts

7.1 Creating Python Scripts to Log PVs

Install the `pyepics` Python package using `pip`. Then, create a Python script called `log_epics_csv.py` that imports `caget` from `epics`, uses `datetime` for timestamps, and `csv` to write data. Define a list of PVs such as `PLC:TEMP_2083`, `PLC:TEMP_2084`, `PLC:TEMP_2093`, and `PLC:TEMP_2094`. Inside a loop, fetch their values, prepend a timestamp, and append the data to a `.csv` file every 2 seconds.

7.2 CSV Output and Timestamping

The Python script should write to a file named `plc_temp_log.csv`. Each row includes a timestamp followed by the four temperature values. This allows for easy offline data analysis and graphing.

7.3 Wrapping Scripts with Bash for Automation

Create a shell script called `log_start.sh`. In it, navigate to the correct folder, source your environment (if needed), and run the Python script. Make the script executable. You can now launch logging with a single command.

8. CS-Studio Setup

8.1 Installing CS-Studio (on Laptop/Desktop)

Download CS-Studio (Phoebus version) from the official nightly build page. Extract the ZIP file and run the launcher script to open the IDE.

8.2 Connecting to Pi's IOC

In CS-Studio, open Preferences. Under the EPICS settings, go to Channel Access and add the Raspberry Pi's IP address to the Address List. Apply the settings and restart CS-Studio.

8.3 Designing a Simple GUI with BOB Files

Start a new display using File → New → Display. Use widgets like Text Entry for writing values, Labels for displaying values, LEDs for binary states, and Bar Graphs for analog values. Assign the correct PV names to each widget, such as `PLC:W4` or `PLC:TEMP_2083`.

8.4 Adding Dynamic Colors and Triggers

For widgets that reflect status, use the Rules section. Create rules that change background color or text based on the PV's value. For example, turn red when a temperature exceeds a threshold.

8.5 Testing and Deployment

Start your IOC on the Raspberry Pi. Open your display in CS-Studio. Confirm that values update in real time and that buttons and setpoints function as expected. Save your `.bob` files for reuse.

9. Final System Checks

9.1 Auto-starting IOC on Boot

To auto-start your IOC at boot, either add the launch command to `/etc/rc.local` before the `exit 0` line, or create a `systemd` service that runs your IOC binary and points to your `st.cmd` file.

9.2 Network/IP Configuration Notes

Assign a static IP address to the Raspberry Pi either through your router or in the `/etc/dhcpd.conf` file. Ensure the hostname is set correctly, such as `kimball-gun`, and test network visibility from your laptop.

9.3 Common Pitfalls and Fixes

If EPICS commands like `caget` or `caput` are not recognized, verify your environment variables are properly set in `.bashrc`. If FINS reads return "INVALID", check your PLC IP, ensure the W area patch is applied, and confirm that your IOC is running. If the `.local` hostname doesn't resolve, switch to using the Pi's static IP address. If CS-Studio widgets don't update, make sure the `SCAN` field in your `.db` records is set properly.

9.4 Summary of Working Components

You now have a complete EPICS system with:

- EPICS Base, asyn, and FINS drivers installed and compiled
- An IOC communicating with an Omron PLC using FINS
- CS-Studio GUI fully operational with live read/write PVs
- Python logger saving PV data with timestamps to CSV
- A GitHub repository backing up your entire `/opt/epics` directory at github.com/Nandpatel18/EPICS-Hutch