Epics_On_RPi Documentation

Release 2014.0317.2

Pete Jemian

Sep 27, 2017

Contents

What is EPICS?

For those who haven't heard, EPICS (http://www.aps.anl.gov/epics) is an open-source control system used worldwide for the routine operation and control of many particle accelerators such as FermiLab and SLAC, for the operation of scientific telescopes such as the Gemini and Keck telescopes, X-ray synchrotrons such as the Advanced Photon Source and the Diamond Light Source, neutron diffraction facilities such as the Spallation Neutron Source, and lots of other neat stuff. The system is scalable and runs on lots of different hardware. Here, we show you how to run EPICS on the **Raspberry Pi!**

Contents

Here is how I installed the Experimental Physics and Industrial Control System software (EPICS)[1] on the Raspberry Pi[2].

The EPICS software is a client/server system. To keep things simple, we will run both the server and a client on the Raspberry Pi. (Clients on other computers on our LAN might be able to interact with our EPICS server as well but we will not discuss that now.)

The EPICS **server** we will use is built in several parts:

- *EPICS Base* provides all the development libraries and a few applications and utilities.
- *synApps* provides additional capabilities that will be useful in real projects. We only use a little of it here, though.

There are many, many possible EPICS **clients**. Since the RPi already has Python, we'll work with that:

- *PyEpics* is an EPICS binding to the Python language, allowing us to build a simple client and interact with our server.

[1] EPICS: http://www.aps.anl.gov/epics
[2] RPi: http://www.raspberrypi.org/

CHAPTER 1

Raspberry Pi Distribution

hardware Raspberry Pi, model B, RASPBRRY-MODB-512M[3]

software 2012-12-16 wheezy-raspbian distribution[4]

Installed wheezy-raspbian distribution on a 16 GB SD card. (It is helpful, but not necessary, to expand the partition to use the full memory of the SD card using `raspi-config` before starting X11):

Filesystem	Size	Used	Avail	Use%	Mounted on
rootfs	15G	2.4G	12G	18%	/
/dev/root	15G	2.4G	12G	18%	/
devtmpfs	220M	0	220M	0%	/dev
tmpfs	44M	252K	44M	1%	/run
tmpfs	5.0M	0	5.0M	0%	/run/lock
tmpfs	88M	68K	88M	1%	/run/shm
/dev/mmcblk0p1	56M	17M	40M	30%	/boot

[3] vendor: http://www.newark.com/jsp/search/productdetail.jsp?SKU=43W5302
[4] wheezy-raspbian: http://downloads.raspberrypi.org/images/raspbian/2012-12-16-wheezy-raspbian/2012-12-16-wheezy-raspbian.zip

CHAPTER 2

Preparing for EPICS

EPICS is flexible about where (which directory path) it is placed. Still, it helps to use standard locations. We'll build it from a directory in the *pi* account, but make a link to that directory called `/usr/local/epics`. You'll need to open a terminal window:

```
cd ~
mkdir -p ~/Apps/epics
sudo su
cd /usr/local
ln -s /home/pi/Apps/epics
exit
cd ~/Apps/epics
```

By making the *epics* directory in *pi* account, we will be able to modify any of our EPICS resources without needing to gain higher privileges.

CHAPTER 3

EPICS Base

EPICS Base is very easy to build. The wheezy-raspbian distribution already has all the tools necessary to build EPICS Base. All that is necessary is to define the host architecture and then build it.

Downloading

The latest stable version of EPICS Base is 3.14.12.3 (3.15 is released but is still not recommended for production use):

```
wget http://www.aps.anl.gov/epics/download/base/baseR3.14.12.3.tar.gz
tar xzf baseR3.14.12.3.tar.gz
ln -s ./base-3.14.12.3 ./base
```

Building

note the backticks

Note the use of backticks in the *export* command. They evaluate the enclosed text as a command and return the result. For more discussion, see the section below titled *Delimiters: Parentheses, Braces, and Back-Quotes*.

EPICS base can be built for many different operating systems and computers. Each build is directed by the `EPICS_HOST_ARCH` environment variable. A command is provided to determine the best choice amongst all the systems for which EPICS currently has definitions. Here is the way to set the environment variable on any UNIX or Linux OS using the *bash* shell (use either of these two commands, they are equivalent in the *bash* shell:

```
export EPICS_HOST_ARCH=`/usr/local/epics/base/startup/EpicsHostArch`
export EPICS_HOST_ARCH=$(/usr/local/epics/base/startup/EpicsHostArch)
```

We can check this value by printing it to the command-line (remember, we are logged in as root):

```
echo $EPICS_HOST_ARCH
linux-arm
```

Good! EPICS base will build for a Linux OS on an ARM architecture. This matches my Raspberry Pi.

Tip: The export command above will be useful for future software development. Add it to the `~/.bash_aliases` file if it exists, otherwise add it to the `~/.bashrc` file with a text editor (such as `nano ~/.bashrc`).

Now, build EPICS base for the first time:

```
cd ~/Apps/epics/base
make
```

This process took about 50 minutes.

Starting

It is possible to start an EPICS IOC at this point, although there is not much added functionality configured. We can prove to ourselves that things will start. Use this linux command:

```
./bin/linux-arm/softIoc
```

and EPICS will start with a basic command line prompt:

```
epics>
```

At this prompt, type:

```
iocInit
```

and lines like these (different time stamp) will be printed:

```
Starting iocInit
############################################################################
## EPICS R3.14.12.3 $Date: Mon 2012-12-17 14:11:47 -0600$
## EPICS Base built Jan 19 2013
############################################################################
iocRun: All initialization complete
epics>
```

Congratulations! EPICS Base has now been built on the Raspberry Pi.

Environment Declarations

To simplify using the tools from EPICS base, consider making these declarations in your environment (`~/.bash_aliases`):

```
export EPICS_ROOT=/usr/local/epics
export EPICS_BASE=${EPICS_ROOT}/base
export EPICS_HOST_ARCH=`${EPICS_BASE}/startup/EpicsHostArch`
export EPICS_BASE_BIN=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}
export EPICS_BASE_LIB=${EPICS_BASE}/lib/${EPICS_HOST_ARCH}
```

```
if [ "" = "${LD_LIBRARY_PATH}" ]; then
    export LD_LIBRARY_PATH=${EPICS_BASE_LIB}
else
    export LD_LIBRARY_PATH=${EPICS_BASE_LIB}:${LD_LIBRARY_PATH}
fi
export PATH=${PATH}:${EPICS_BASE_BIN}
```

Note: We are being a bit cautious here, not to remove any existing definition of **LD_LIBRARY_PATH**. Also the comparison is a *Yoda* condition[5], placing the constant term on the left of the comparison. Yoda conditions can reveal accidental assignments at run time. Perhaps not so much in the *bash* shell, but it's useful in programming languages.

After EPICS base has been built, we see that it has taken ~35 MB of storage:

```
pi@raspberrypi:~$ du -sc base-3.14.12.3
35636  base-3.14.12.3
```

[5] Yoda condition: https://en.wikipedia.org/wiki/Yoda_Conditions

CHAPTER 4

synApps

synApps is a collection of software tools that help to create a control system for beamlines. It contains beamline-control and data-acquisition components for an EPICS based control system.

There are instructions for installing synApps posted online: http://www.aps.anl.gov/bcda/synApps/synApps_5_6.html

Download

The current release of synApps (as this was written in 2013-02) is v5.6. The compressed source archive file is available from the BCDA group at APS. The file should be 149 MB:

```
wget http://www.aps.anl.gov/bcda/synApps/tar/synApps_5_6.tar.gz
tar xzf synApps_5_6.tar.gz
```

Uncompressed and unconfigured, the synApps_5_6 source folder is ~541 MB.

Configuring

All work will be relative to this folder:

```
cd ~/Apps/epics/synApps_5_6/support
```

Follow the instructions in the README file. These are the changes I made to run on the Raspberry Pi.

file	changes
configure/CONFIG_SITE	no changes
configure/RELEASE	`SUPPORT=/usr/local/epics/synApps_5_6/support` `EPICS_BASE=/usr/local/epics/base`

After modifying `configure/RELEASE`, propagate changes to all module RELEASE files by running:

```
cd ~/Apps/epics/synApps_5_6/support
make release
```

Edit `Makefile` and remove support for these modules:

- ALLEN_BRADLEY
- DAC128V
- IP330
- IPUNIDIG
- LOVE
- IP
- VAC
- SOFTGLUE
- QUADEM
- DELAYGEN
- CAMAC
- VME
- AREA_DETECTOR
- DXP

xxx module: reconfigure

The **xxx** module is an example and template EPICS IOC, demonstrating configuration of many synApps modules. APS beam line IOCs are built using *xxx* as a template.

In `xxx-5-6/configure/RELEASE`, place a comment on lines 19 and 32 to remove build support for *areaDetector* in *xxx*:

```
#AREA_DETECTOR=$(SUPPORT)/areaDetector-1-8beta1

#IP=$(SUPPORT)/ip-2-13
```

In `xxx-5-6/xxxApp/src/xxxCommonInclude.dbd`, place a comment on line 34:

```
#include "ipSupport.dbd"
```

Then, in `xxx-5-6/xxxApp/src/Makefile`, comment out all lines that refer to *areaDetector* components, such as *ADsupport*, "NDPlugin*, *simDetector*, and *netCDF*, as well as *dxp* support. Here are the lines I found:

```
#iocxxxWin32_DBD += ADSupport.dbd  NDFileNetCDF.dbd
#xxx_LIBS_WIN32 += ADBase NDPlugin netCDF
#iocxxxCygwin_DBD += ADSupport.dbd  NDFileNetCDF.dbd
#xxx_LIBS_cygwin32 += ADBase NDPlugin netCDF
#iocxxxCygwin_DBD += ADSupport.dbd NDFileNetCDF.dbd
#xxx_LIBS_cygwin32 += ADBase NDPlugin netCDF
#iocxxxLinux_DBD += ADSupport.dbd  NDFileNetCDF.dbd
#xxx_LIBS_Linux += ADBase NDPlugin netCDF
```

```
#iocxxxCygwin_DBD += simDetectorSupport.dbd commonDriverSupport.dbd
#xxx_LIBS_cygwin32 += simDetector
#iocxxxLinux_DBD += simDetectorSupport.dbd commonDriverSupport.dbd
#xxx_LIBS_Linux += simDetector

#xxx_Common_LIBS += ip
```

Install necessary EPICS Extensions

synApps 5.6 requires the *msi* EPICS extension. First, setup the extensions subdirectory

```
cd ~/Apps/epics
wget http://www.aps.anl.gov/epics/download/extensions/extensionsTop_20120904.tar.gz
tar xzf extensionsTop_20120904.tar.gz
```

Now, download *msi*, unpack, build, and install it:

```
wget http://www.aps.anl.gov/epics/download/extensions/msi1-5.tar.gz
cd extensions/src
tar xzf ../../msi1-5.tar.gz
cd msi1-5
make
```

Make these additional declarations in your environment (`~/.bash_aliases`):

```
export EPICS_EXT=${EPICS_ROOT}/extensions
export EPICS_EXT_BIN=${EPICS_EXT}/bin/${EPICS_HOST_ARCH}
export EPICS_EXT_LIB=${EPICS_EXT}/lib/${EPICS_HOST_ARCH}
if [ "" = "${LD_LIBRARY_PATH}" ]; then
    export LD_LIBRARY_PATH=${EPICS_EXT_LIB}
else
    export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${EPICS_BASE_LIB}
fi
export PATH=${PATH}:${EPICS_EXT_BIN}
```

Install other support

The EPICS sequencer needs the *re2c* package (http://re2c.org/). This is available through the standard package installation repositories:

```
sudo apt-get install re2c
```

Building

Now, build the components of synApps selected in the *Makefile*:

```
cd ~/Apps/epics/synApps_5_6/support
make release
make rebuild
```

The `make rebuild` step took about 70 minutes.

CHAPTER 5

PyEpics

It is possible to run the *PyEpics* support from Matt Newville (http://cars.uchicago.edu/software/python/pyepics3/) on the Raspberry Pi!

Preparing Python

To simplify installation, we'll use *easy_install* (from *setuptools*).

Note: The additions to the Python installation will be done as root. Here's how to become root on the default wheezy-raspbian distribution.

```
sudo su
```

First, install the setuptools package from the wheezy repository. (Also, as long as we're here, the *ipython* shell is very helpful.) Let's load them both:

```
sudo apt-get install python-setuptools ipython
```

Next, we want to know which version of Python will be run:

```
# which python
/usr/bin/python
ls -lAFg /usr/bin/python
lrwxrwxrwx 1 root 9 Jun  5  2012 /usr/bin/python -> python2.7*
```

Python 2.7 will be run.

Install PyEpics

With the *setuptools* installed, it becomes simple to install PyEpics (still as root):

```
easy_install -U PyEpics
```

The installation will complain about missing EPICS support libraries (*libca* and *libCom*). Now, we can address that (still as root):

```
cd /usr/local/lib/python2.7/dist-packages/pyepics-3.2.1-py2.7.egg
cp /home/pi/Apps/epics/base-3.14.12.3/lib/linux-arm/libca.so.3.14 ./
cp /home/pi/Apps/epics/base-3.14.12.3/lib/linux-arm/libCom.so.3.14 ./
ln -s libca.so.3.14  libca.so
ln -s libCom.so.3.14  libCom.so
```

Now, exit from *root* back to the *pi* account session:

```
exit
```

Testing PyEpics

First, you might be eager to see that PyEpics will load. Save this code in the file *verify.py* (in whatever folder you wish, we'll use */home/pi*):

```
1 #!/usr/bin/env python
2
3 import epics
4
5 print epics.__version__
6 print epics.__file__
```

Also, remember to make the file executable:

```
chmod +x verify.py
```

Now, run this and hope for the best:

```
./verify.py
3.2.1
/usr/local/lib/python2.7/dist-packages/epics/__init__.pyc
```

This shows that PyEpics was installed but it does not test that EPICS is working.

Testing PyEpics with an IOC

Note: We'll need to use several tools at the same time. It is easiest to create several terminal windows.

To test that EPICS communications are working, we need to do some preparations.

softIoc

The simplest way to do this is to use the *softIoc* support from EPICS base with a simple EPICS database. Save this into a file called *simple.db*:

```
record(bo, "rpi:trigger")
{
        field(DESC, "trigger PV")
        field(ZNAM, "off")
        field(ONAM, "on")
}
record(stringout, "rpi:message")
{
        field(DESC, "message on the RPi")
        field(VAL,  "RPi default message")
}
```

Note: The file *simple.db* defines two EPICS records: *rpi:trigger* and *rpi:message*. The first record can take the value of 0 or 1, which also have the string values of "off" and "on", respectively. The second record is a string.

Now, run the EPICS soft IOC support with this database:

```
pi@raspberrypi:~$ softIoc -d simple.db
Starting iocInit
############################################################################
## EPICS R3.14.12.3 $Date: Mon 2012-12-17 14:11:47 -0600$
## EPICS Base built Jan 19 2013
############################################################################
iocRun: All initialization complete
epics> dbl
rpi:trigger
rpi:message
epics>
```

camonitor

In a separate terminal window, watch the soft IOC for any changes to EPICS PVs we created above:

```
pi@raspberrypi:~$ camonitor rpi:trigger rpi:trigger.DESC rpi:message rpi:message.DESC
rpi:trigger                    <undefined> off UDF INVALID
rpi:trigger.DESC               <undefined> trigger PV UDF INVALID
rpi:message                    <undefined> RPi default message UDF INVALID
rpi:message.DESC               <undefined> message on the RPi UDF INVALID
```

Python code

Now, let's communicate with the PVs of the softIoc. Put this code in file *test.py*:

```
#!/usr/bin/env python

import epics

print epics.caget('rpi:trigger.DESC')
print epics.caget('rpi:trigger')
print epics.caget('rpi:message.DESC')
print epics.caget('rpi:message')

epics.caput('rpi:message', 'setting trigger')
```

```
epics.caput('rpi:trigger', 1)
print epics.caget('rpi:trigger.DESC')
print epics.caget('rpi:trigger')
print epics.caget('rpi:message.DESC')
print epics.caget('rpi:message')

epics.caput('rpi:message', 'clearing trigger')
epics.caput('rpi:trigger', 0)
print epics.caget('rpi:trigger.DESC')
print epics.caget('rpi:trigger')
print epics.caget('rpi:message.DESC')
print epics.caget('rpi:message')
```

Make the file executable and then run it:

```
pi@raspberrypi:~$ chmod +x test.py
pi@raspberrypi:~$ ./test.py
trigger PV
0
message on the RPi
RPi default message
trigger PV
1
message on the RPi
setting trigger
trigger PV
0
message on the RPi
clearing trigger
pi@raspberrypi:~$
```

Note that new messages have also printed on the terminal running *camonitor*:

```
rpi:message     2013-01-21 08:20:28.658746 setting trigger
rpi:trigger     2013-01-21 08:20:28.664845 on
rpi:message     2013-01-21 08:20:28.697210 clearing trigger
rpi:trigger     2013-01-21 08:20:28.702967 off
```

CHAPTER 6

Files

These files, described above, are available for direct download:

file	description
`verify.py`	test that PyEpics is installed
`simple.db`	simple EPICS database to test PyEpics communications with EPICS
`test.py`	Python code to test PyEpics communications with EPICS

CHAPTER 7

Delimiters: Parentheses, Braces, and Back-Quotes

In the code examples above, a combination of parentheses, braces, and back-quotes (a.k.a. accent grave or backtick) are used.

In the */bin/bash* shell, braces, **{** and **}**, are used to delimit the scope of symbol names during shell expansion. In the code examples above, the delimiters are probably unnecessary. Using these delimiters is a cautious practice to adopt. Parentheses are not recognized in this context:

```
~$ echo $EPICS_ROOT
/usr/local/epics
~$ echo ${EPICS_ROOT}
/usr/local/epics
~$ echo $(EPICS_ROOT)
EPICS_ROOT: command not found
```

However, in the various files and commands that configure and command the EPICS components, parentheses, **(** and **)**, are the required delimiters. See these examples from above:

```
#AREA_DETECTOR=$(SUPPORT)/areaDetector-1-8beta1
#IP=$(SUPPORT)/ip-2-13
```

Sometimes, in a shell script, it is necessary to assign a variable with the value obtained from a command line tool. One common way to do that, shared by **bash** and some other shells such as **tcsh**, is to enclose the command line tool with the **‘** back-quote character. See this example:

```
~$ echo $SHELL
/bin/bash
~$ echo `/usr/local/epics/base-3.14.12.3/startup/EpicsHostArch`
linux-x86_64
```

An alternative way to do this assignment in *bash* was pointed out, to use shell expansion with parentheses as the delimiters, such as:

```
~$ echo $(/usr/local/epics/base-3.14.12.3/startup/EpicsHostArch)
linux-x86_64
```