
EPICS の備忘録

西田 賢人

2022 年 08 月 26 日

Contents:

第 1 章	EPICS とは	1
1.1	EPICS の概要	1
1.2	EPICS の特徴	1
第 2 章	EPICS 適用	3
2.1	EPICS サーバの導入について	3
2.2	EPICS の用語集	5
2.3	基本動作例 1 : Google に telnet 接続する EPICS 制御	6
2.4	基本動作例 2 : Arduino の AD 変換端子からの入力読取り	12
2.5	基本動作例 3 : EPICS-Arduino による L チカ (ON/OFF) 制御	19
2.6	基本動作例 4 : EPICS-Arduino による LED-PWM 調光制御	25
第 3 章	References	33

第 1 章

EPICS とは

1.1 EPICS の概要

- EPICS は、PC から複数の制御機器を統括的に取り扱えるようにした制御用ミドルウェア、サーバ、命令群である。
- ユーザ PC のソフトウェアとハードウェアをシームレスに接続できる環境を提供しうる。
- 開発元は、アルゴンヌ国立研究所。Open License (EPICS Open License) である。
- 公式の説明は以下の通り。

Note: "EPICS is a set of Open Source software tools, libraries and applications developed collaboratively and used worldwide to create distributed soft real-time control systems for scientific instruments such as a particle accelerators, telescopes and other large scientific experiments."

1.2 EPICS の特徴

以下のようなシステムの制御を得意とする。

- 多数の機器がぶらさがった大規模システムを統括的に制御できる。
- 高速・リアルタイムな通信が可能
- 柔軟かつスケーラブルで、拡張が容易である。

第 2 章

EPICS 適用

2.1 EPICS サーバの導入について

ここでは、EPICS サーバの初期設定、及び、基本動作テストについて記載する。

2.1.1 環境

以下の環境を想定する。

- IOC として使用する PC : **Raspberry Pi 2 Model B**
- OS : Raspbian
- user 名 : epics
- EPICS 導入ディレクトリ (環境変数\${EPICS_BASE}) : /home/epics/epics/epics-base/

2.1.2 EPICS(epics-base) のインストール

epcis-base のダウンロード及びインストール

- base の本家ウェブサイト (<https://docs.epics-controls.org/projects/how-tos/en/latest/getting-started/installation.html>) に従う。

```
$ git clone --recursive https://github.com/epics-base/epics-base.git
$ cd epics-base
$ make
```

環境変数の設定

- 以下を `${HOME}/.zshrc` や `${HOME}/.bash_profile` 等へ書き込む

```
export EPICS_BASE=${HOME}/epics/epics-base
export EPICS_HOST_ARCH=$( ${EPICS_BASE}/startup/EpicsHostArch )
export PATH=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}:${PATH}
```

epics-base のテスト

- 以下の、データベースファイル (`.db` ファイル, `example.db`) を作成する.

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

- 以下、コマンドにて実行する.

```
$ softIoc -d example.db
epics>
( EPICS コンソールに入る. 新規コンソールを立ち上げて、以下を実行. )
$ caput temperature:water 22
$ caget temperature:water
```

2.1.3 その他、モジュールのインストール

- EPICS サーバを使用する際に、ハードウェア機器に応じて、拡張モジュールを使用する事が多い.
- 導入するモジュールは例えば、以下.
 - ASYN (Asynchronous (非同期通信用モジュール) `re2c` が必要)
 - Stream Devices (バイトストリーム処理用モジュール: 例えば 1 バイト文字列を利用した制御 RS-232C 等)
 - seq (シーケンサー, C 言語ライクな状態記述言語によるプログラム)
 - devGpio (Raspberry Pi の GPIO 端子の制御用モジュール、主はエラーで動作できていない.)
- インストールは各 website に従えば、問題なくインストール可能.

2.2 EPICS の用語集

2.2.1 概念・機器構成編

EPICS 制御の機器構成・概念を理解する上で必要となる用語について、以下に備忘録を記す。

表 1 用語集 (EPICS 概念・機器構成)

用語	略, 別名	説明
EPICS	Experimental Physics and Industrial Control System	制御用ミドルウェア、ツールパッケージの総称
IOC	Input/Output Controller	ハード側の制御器。ハード機器に制御命令を出す。下位層にあたる。
OPI	OPerational Interface	ユーザ側の制御ソフトウェア。上位層。

2.2.2 モジュール・ソフトウェア編

EPICS 制御に用いるモジュール・ソフトウェアの用語について、以下に備忘録を記す。

表 2 用語集 (モジュール・ソフトウェア)

用語	略, 別名	説明
Asyn		非同期通信制御用のモジュール。StreamDevices 内でも使用される。ユーザに近い位置で積極的に使用するの、時代遅れの可能性あり。StreamDevice を使用すべきか。
StreamDevice	stream etc.	バイトストリームを取り扱う制御モジュール。1 バイト文字等を送信するため制御（つまりは RS-232C や USB 含む Serial 制御、等、なんでも制御可能。protocol ファイルと db ファイルの設定のみで制御できるため重宝されている様子）
re2c		字句解析ツール。数式等を理解し、コードを自動ジェネレートするなど使用する。Asyn で使用。apt 管理されている。

2.2.3 ファイル編

EPICS で使用するファイルについて、以下に備忘録を記す。

表 3 用語集 (EPICS 概念・機器構成)

ファイル名/拡張子	意味	説明
.db	database	EPICS で入出力管理するレコード（データ型、タイミングなど）を記述する
.proto	protocol	StreamDevice で使用するプロトコル（データ入出力・受け渡し形式について記載するファイル）
configure/RELEASE		EPICS_BASE やモジュールのインストール位置等、コンパイルに必要な情報を記載するファイル。

2.3 基本動作例 1 : Google に telnet 接続する EPICS 制御

2.3.1 検証目標

- IOC を構築。
 - IOC の動作検証。
 - IOC と OPI 間で通信する。
- (IOC:Input/Output Controller, OPI:OPerational Interface)

2.3.2 前提条件

- IOC として "**RaspberryPi**"、OPI として、手元 PC の macOS を使用する。
- RaspberryPi-mac 間は LAN ケーブルで接続し、RaspberryPi はインターネットへ接続できる。(同一 LAN 内.)
- 接続先は、適当なサーバ : `www.google.com:80` (80 は HTML 通信用のウェルノウンポート) とし、HTTP リクエストする。
- 制御モジュールとして、"**StremDevice**" を使用する。
- IOC(RaspberryPi) に、`epics-base`, `Asyn`, `StreamDevice` はインストール済み
- OPI(macOS) からは、python コンソールから "**pyEpics**" (pip からインストール可能) を利用する。

- EPICS_BASE は、"\$HOME/epics/epics-base", サポートモジュールは、"\$HOME/epics/support/"にインストールされており、App の作成場所は、"\$HOME/epics/app/"とする。

2.3.3 IOC 構築

IOC 構築の手順

- IOC-App(アプリ) 構築の基本手順は、以下である。
 1. ベースアプリ を作成する。
 2. **configure/RELEASE** にコンパイルに必要な情報 (EPICS_BASE のパス/モジュールのパス) を記載する。
 3. "**xxxApp**" ディレクトリに、データベース及び使用するモジュールの情報を記載する。
 4. StreamDevice などモジュールを利用する際は、モジュールに応じた設定ファイル（例えば StreamDevice の場合、**protocols** ディレクトリに **xxx.proto**）を作成する。
 5. **iocBoot** ディレクトリの **ioc_xxx****内にある **IOC** 初期化スクリプト ****"st.cmd"** を編集し、実行可能とする。
 6. アプリのベースディレクトリ（\$(TOP) として Makefile 中に記載されている）にて、\$ make、したのちに、sudo 権限付きで IOC 初期化スクリプト "st.cmd" を実行する

以下、上記手順について詳述する。

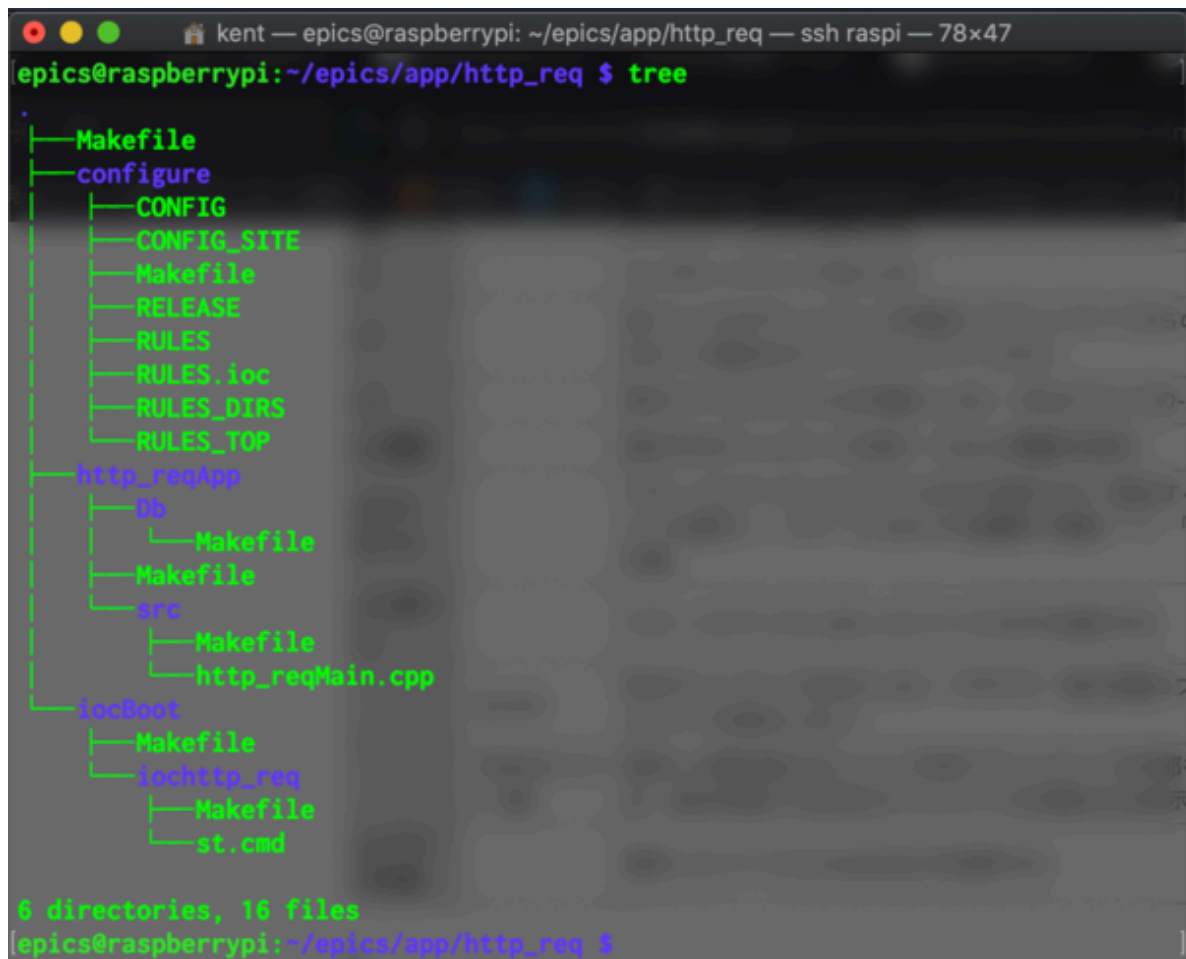
1. ベースアプリの作成

- makeBaseApp.pl を用いたベースアプリの作成

```
$ mkdir -p ~/epics/app/http_req
$ cd ~/epics/app/http_req
$ makeBaseApp.pl -t ioc http_req
$ makeBaseApp.pl -i -t ioc http_req
```

- 1 回目の makeBaseApp.pl で作成されるファイル・ディレクトリは以下。
 - Makefile
 - configure
 - http_reqApp
- 2 回目の makeBaseApp.pl で作成されるファイル・ディレクトリは以下。
 - iocBoot

- ディレクトリツリーは以下.



```
epics@raspberrypi:~/epics/app/http_req $ tree
.
├── Makefile
├── configure
│   ├── CONFIG
│   ├── CONFIG_SITE
│   ├── Makefile
│   ├── RELEASE
│   ├── RULES
│   ├── RULES.ioc
│   ├── RULES_DIRS
│   └── RULES_TOP
├── http_reqApp
│   ├── Db
│   │   └── Makefile
│   ├── Makefile
│   └── src
│       ├── Makefile
│       └── http_reqMain.cpp
└── iocBoot
    ├── Makefile
    ├── iochttp_req
    │   ├── Makefile
    │   └── st.cmd
    └── st.cmd

6 directories, 16 files
epics@raspberrypi:~/epics/app/http_req $
```

2. 共通コンパイル設定事項の編集 (configure/RELEASE)

- configure/RELEASE に、共通のコンパイル設定（モジュールの場所等、）を例えば以下のように記載する.

```
ASYN    = /home/epics/epics/support/asyn
STREAM  = /home/epics/epics/support/StreamDevice
```

3. データベースファイルとコンパイルの準備

- データベース及び使用するモジュールの情報を記載し、~/epics/app/http_req/http_reqApp/Db/http_req.db を作成する.

```
$ nano ~/epics/app/http_req/http_reqApp/Db/http_req.db
( 以下を保存 )
record( stringin, "http:get" )
{
```

(次のページに続く)

(前のページからの続き)

```
field( DESC, "getbitstream" )
field( DTYP, "stream" )
field( INP , "@http_req.proto getVal web")
}
```

- データベースのコンパイル対象として、上記の"http_req.db"を追加.

```
$ nano ~/epics/app/http_req/http_reqApp/Db/Makefile
( 以下を追記 )
DB += http_req.db
```

- その他モジュールを利用する場合は、IOC の通信コードのコンパイルに使用するモジュール情報を、
"http_reqApp/src/Makefile" に記載し、コンパイルできるようにする.

```
$ nano ~/epics/app/http_req/http_reqApp/src/Makefile
( 以下を追記 )
http_req_DBD += stream.dbd
http_req_DBD += asyn.dbd
http_req_DBD += drvAsynIPPort.dbd

http_req_LIBS += stream
http_req_LIBS += asyn
```

4. StreamDevice の設定ファイル ("protocol") の作成

- アプリのベースディレクトリ (~/epics/app/http_req/) にディレクトリ "protocols"を作成し、StreamDevice
の入出力情報を記載する.

```
$ mkdir $HOME/epics/app/http_req/protocols
$ nano http_req.proto

(以下を記入)
Terminator = CR LF;
getVal {
out "GET / HTTP/1.1\nHost: www.google.co.jp\n";
in "%39c";
ExtraInput = Ignore;
}
```

5. IOC 初期化スクリプト "st.cmd" の編集

- IOC 初期化スクリプト (iocBoot/iochttp_req/st.cmd) に、以下の情報を記載する。
 - StreamDevice を使用する場合、protocols ディレクトリの位置を記載する。
 - 初期化時にロードするデータベースファイルを記載する。
 - ethernet ケーブルなどの、ハードウェアを使用する場合は、設定を記載する。
- ここでは、以下を記載する。

```
epicsEnvSet( "STREAM_PROTOCOL_PATH", "....../protocols" )
dbLoadRecords( "db/http_req.db", "user=epics" )
drvAsynIPPortConfigure "web", "www.google.co.jp:80",0,0,0
```

- スクリプトに実行権限を与えておく。

```
$ chmod u+x $HOME/epics/app/http_req/iocBoot/iochttp_req/st.cmd
```

6. make 及び、初期化スクリプト "st.cmd" の実行

- ベースディレクトリにて make する。

```
$ cd $HOME/epics/app/http_req/
$ make distclean
$ make
```

- 初期化スクリプトを実行する。

```
$ cd $HOME/epics/app/http_req/iocBoot/iochttp_req/
$ sudo ./st.cmd
```

2.3.4 IOC の動作状況の確認

EPICS コンソール上での確認

- EPICS コンソールへエラーなく遷移していることを確認。
- 以下を実行。

```
epics> dbpf http:get 0
epics> dbgf http:get
```

- 戻り値は、

```
DBF_STRING:      "HTTP/1.1 200 OK"
```

ローカルからの CA

- epics-base がインストールされている IOC/OPI では、PV(Process Variable) に CA(Channel Access) が可能
- 別コンソールを立ち上げて、以下コマンドを実行

```
epics@raspberrypi: ~ $ caget http:get
http:get  HTTP/1.1 200 OK
```

OPI (手元 macOS) からの CA

- 同一ネットワークに LAN 接続してある OPI(手元 PC:macOS) から CA 可能.
- 以下、IP アドレス/ポート番号の設定を環境変数にセット.

```
$ export EPICS_CA_ADDR_LIST="169.254.202.104:5064"

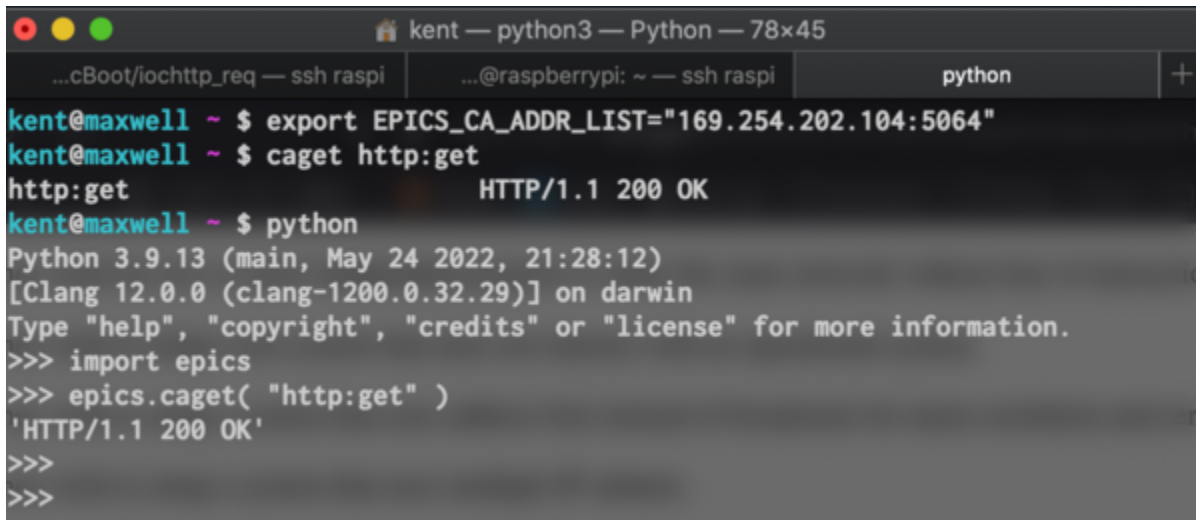
( e.g.1 $ export EPICS_CA_ADDR_LIST="1.2.3.255 8.9.10.255" etc. )
or
( e.g.2 $ export EPICS_CA_ADDR_LIST="1.1.1.1" and,   )
(           $ export EPICS_CA_SERVER_PORT=5064 etc.   )
```

- コンソールから CA.

```
$ caget http:get
```

- pyEpics から CA.

```
$ python3
>>> import epics
>>> epics.caget( "http:get" )
'HTTP/1.1 200 OK'
```



```
kent — python3 — Python — 78x45
...cBoot/iochttp_req — ssh raspi | ...@raspberrypi: ~ — ssh raspi | python | +
kent@maxwell ~ $ export EPICS_CA_ADDR_LIST="169.254.202.104:5064"
kent@maxwell ~ $ caget http:get
http:get HTTP/1.1 200 OK
kent@maxwell ~ $ python
Python 3.9.13 (main, May 24 2022, 21:28:12)
[Clang 12.0.0 (clang-1200.0.32.29)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import epics
>>> epics.caget( "http:get" )
'HTTP/1.1 200 OK'
>>>
>>>
```

- OPI から IOC を介して、制御 (HTTP リクエスト) を実施することができた。

2.3.5 参考 URL

- 参考 ノート : ["https://note.com/dev_associate/n/nfa4605c70f60"](https://note.com/dev_associate/n/nfa4605c70f60), ["https://note.com/dev_associate/n/nd886d700b10a"](https://note.com/dev_associate/n/nd886d700b10a)
- OPI/IOC 通信時のポート番号、IP アドレスの設定 (https://epics.anl.gov/EpicsDocumentation/AppDevManuals/ChannelAccess/cadoc_4.htm)

2.4 基本動作例 2 : Arduino の AD 変換端子からの入力読取り

2.4.1 検証目標

- ハードウェア (Arduino) を遠隔制御する検証。

2.4.2 前提条件

- IOC は "**RaspberryPi**"、OPI は手元 PC の macOS、制御機器は "**Arduino Uno**"。
- Arduino-RaspberryPi 間は USB 接続、RaspberryPi-mac 間は LAN ケーブルで接続する。
- Arduino の 5V-GND 端子間に 1K Ω 抵抗 x2 個、さらに LED を直列に接続し、さらに抵抗の中間地点に Arduino A0 端子を接続する。A0 から、A/D 変換 (ADC) によってデジタル値に変換した値を読み取る。
- 制御モジュールとして、"**StremDevice**" を使用する。
- その他、`asyn`, "**pyEpics**" を適宜使用する。

- 作業ディレクトリ：\${HOME}/epics/app/simpleRead/ (\${HOME}=/home/epics/)

2.4.3 Arduino プログラム (ADC) の転送

- Arduino プログラムは以下である.

リスト 1 A Arduino program to measure the voltage between 2 registers.

```
// use A0-plug to check the voltage. //  
//  
// parameters //  
  
int analogPin = 0;  
int val = 0;  
int sampleRate = 19200;  
int interval_ms = 1000  
  
void setup() {  
    Serial.begin( sampleRate );  
}  
  
void loop() {  
    val = analogRead( analogPin );  
    Serial.println(val);  
    delay( interval_ms );  
}
```

- 上記プログラムは、**Arduino IDE** を使用して Arduino へ転送しておく。(手元 PC からでも勿論、可)

2.4.4 IOC 構築

IOC 構築 / テスト の手順

- アプリ名は、参考 URL (下記) に従い、simpleRead とする.
- また、.db やレコード名などは、全て simpleRead へ統一する.
- IOC-App 構築の手順は、以下である.
 1. ベースアプリ の作成.
 2. **configure/RELEASE** に共有するモジュールのインストールパスを追記.
 3. "**simpleReadApp**" 内の データベース情報、及び、コンパイル用の Makefile へ追記する.
 4. **StreamDevice** 用の プロトコル を、 **protocols/simpleRead.proto** として作成する.

5. IOC 初期化スクリプト (`iocBoot/iocsimpleRead/st.cmd`) を編集し、実行可能とする。

6. `make`、及び、`st.cmd` の実行、`camonitor` により、経時変化を観察する。

以下、上記手順について詳述する。

1. ベースアプリの作成

- `makeBaseApp.pl` を用いたベースアプリの作成

```
$ mkdir -p ~/epics/app/simpleRead
$ cd ~/epics/app/simpleRead
$ makeBaseApp.pl -t ioc simpleRead
$ makeBaseApp.pl -i -t ioc simpleRead
```

2. 共通コンパイル設定事項の編集 (`configure/RELEASE`)

- `configure/RELEASE` に、共通のコンパイル設定（モジュールの場所等、）を例えば以下のように記載する。

```
ASYN      = /home/epics/epics/support/asyn
STREAM    = /home/epics/epics/support/StreamDevice
```

3. データベースファイルとコンパイルの準備

- データベース及び使用するモジュールの情報を記載し、`${HOME}/epics/app/simpleRead/simpleReadApp/Db/simpleRead.db` を作成する。

リスト 2 simpleRead.db

```
1 record(longin, "epics:simpleRead")
2 {
3     field(DESC, "A/D convertor raw input signal" )
4     field(DTYP, "stream")
5     field(INP, "@simpleRead.proto getval PS1")
6     field(SCAN, "I/O Intr")
7 }
```

- データベースのコンパイル対象として、上記の"simpleRead.db"を追加。

```
@ simpleReadApp/Db/Makefile

DB += simpleRead.db
```

- その他モジュールを利用する場合は、IOC の通信コードのコンパイルに使用するモジュール情報を、`"simpleReadApp/src/Makefile"` に記載し、コンパイルできるようにする。

```
@ simpleReadApp/src/Makefile

simpleRead_DBD += stream.dbd
simpleRead_DBD += asyn.dbd
simpleRead_DBD += drvAsynSerialPort.dbd

simpleRead_LIBS += stream
simpleRead_LIBS += asyn
```

4. StreamDevice の設定ファイル ("protocol") の作成

- ディレクトリ "protocols" を作成し、StreamDevice の入出力情報を記載する。

```
$ mkdir $HOME/epics/app/simpleRead/protocols
```

リスト 3 simpleRead.proto

```
1 Terminator = CR LF;
2
3 getVal{
4     in "%d";
5 }
```

5. IOC 初期化スクリプト "st.cmd" の編集

- IOC 初期化スクリプト (iocBoot/iocsimpleRead/st.cmd) に、以下の情報を記載する。

リスト 4 st.cmd

```
#!/../../bin/linux-arm/simpleRead

#- You may have to change simpleRead to something else
#- everywhere it appears in this file

< envPaths

# -- n.k. added -- #
epicsEnvSet("STREAM_PROTOCOL_PATH", "../../protocols")

cd "${TOP}"

## Register all support components
dbLoadDatabase "dbd/simpleRead.dbd"
simpleRead_registerRecordDeviceDriver pdbbase

## Load record instances
```

(次のページに続く)

(前のページからの続き)

```
#dbLoadRecords ("db/xxx.db", "user=epics")
# -- n.k. added -- #
dbLoadRecords ("db/simpleRead.db")

# drvGenericSerialConfigure( "PS1", "/dev/ttyACM0" )
# asynSetPortOption( "PS1", "baud", "19200" )
drvAsynSerialPortConfigure ("PS1", "/dev/ttyACM0")
asynSetOption ("PS1", 0, "baud", "19200")
# -- n.k. added -- #

cd "${TOP}/iocBoot/${IOC}"
iocInit

## Start any sequence programs
#seq sncxxx, "user=epics"
```

Warning: (隘路事項) dbLoadRecord, dbLoadDatabase の順番が逆になったりすると、うまく動作しない。しかも、".db"ファイルの1行目がおかしいというエラーメッセージがでるので、ミスリーディングである。st.cmd 前後の状態も確認すべきである。

- スクリプトに実行権限を与えておく。

```
$ chmod u+x $HOME/epics/app/simpleRead/iocBoot/iocsimpleRead/st.cmd
```

6. make 及び、初期化スクリプト "st.cmd" の実行

- ベースディレクトリにて make する。

```
$ cd $HOME/epics/app/simpleRead/
$ make distclean
$ make
```

- 初期化スクリプトを実行する。

```
$ cd $HOME/epics/app/simpleRead/iocBoot/iocsimpleRead/
$ sudo ./st.cmd
```

- make 完了後の最終的なディレクトリツリーは以下。

リスト 5 Directory tree after compilation.

```

epics@raspberrypi:~/epics/app/simpleRead $ tree
.
├── Makefile
├── bin
│   ├── ^c2^a0^c2^a0 └── linux-arm
│   └── ^c2^a0^c2^a0 └── simpleRead
├── configure
│   ├── ^c2^a0^c2^a0 └── CONFIG
│   ├── ^c2^a0^c2^a0 └── CONFIG_SITE
│   ├── ^c2^a0^c2^a0 └── Makefile
│   ├── ^c2^a0^c2^a0 └── O.Common
│   ├── ^c2^a0^c2^a0 └── O.linux-arm
│   ├── ^c2^a0^c2^a0 | ^c2^a0^c2^a0 └── Makefile
│   ├── ^c2^a0^c2^a0 └── RELEASE
│   ├── ^c2^a0^c2^a0 └── RULES
│   ├── ^c2^a0^c2^a0 └── RULES.ioc
│   ├── ^c2^a0^c2^a0 └── RULES_DIRS
│   └── ^c2^a0^c2^a0 └── RULES_TOP
├── db
│   └── ^c2^a0^c2^a0 └── simpleRead.db
├── dbd
│   └── ^c2^a0^c2^a0 └── simpleRead.dbd
├── iocBoot
│   ├── ^c2^a0^c2^a0 └── Makefile
│   └── ^c2^a0^c2^a0 └── iocsimpleRead
│       ├── ^c2^a0^c2^a0 └── Makefile
│       ├── ^c2^a0^c2^a0 └── envPaths
│       └── ^c2^a0^c2^a0 └── st.cmd
├── lib
│   └── ^c2^a0^c2^a0 └── linux-arm
├── protocols
│   └── ^c2^a0^c2^a0 └── simpleRead.proto
└── simpleReadApp
    ├── Db
    │   ├── ^c2^a0^c2^a0 └── Makefile
    │   ├── ^c2^a0^c2^a0 └── O.Common
    │   ├── ^c2^a0^c2^a0 └── O.linux-arm
    │   ├── ^c2^a0^c2^a0 | ^c2^a0^c2^a0 └── Makefile
    │   └── ^c2^a0^c2^a0 └── simpleRead.db
    ├── Makefile
    └── src
        ├── Makefile
        ├── O.Common
        │   └── ^c2^a0^c2^a0 └── simpleRead.dbd
        ├── O.linux-arm
        └── ^c2^a0^c2^a0 └── Makefile

```

(次のページに続く)

(前のページからの続き)

```
| ^^c2^^a0^^c2^^a0 |—— simpleRead
| ^^c2^^a0^^c2^^a0 |—— simpleRead.dbd.d
| ^^c2^^a0^^c2^^a0 |—— simpleReadMain.d
| ^^c2^^a0^^c2^^a0 |—— simpleReadMain.o
| ^^c2^^a0^^c2^^a0 |—— simpleRead_registerRecordDeviceDriver.cpp
| ^^c2^^a0^^c2^^a0 |—— simpleRead_registerRecordDeviceDriver.d
| ^^c2^^a0^^c2^^a0 |—— simpleRead_registerRecordDeviceDriver.o
|—— simpleReadMain.cpp
```

```
19 directories, 33 files
```

```
epics@raspberrypi:~/epics/app/simpleRead $
```

2.4.5 ADC の動作状況の確認

ローカルからの **camonitor**

- 別コンソールを立ち上げて、以下コマンドを実行

```
epics@raspberrypi: ~ $ camonitor epics:simpleRead
```

OPI (手元 macOS) からの CA

- pyEpics から CA.

```
$ python3
>>> import epics
>>> epics.caget      ( "epics:simpleRead" )
>>> epics.camonitor( "epics:simpleRead" )
```

- OPI から IOC を介して、制御 (電圧モニタ) を実施することができた.

2.4.6 参考 URL

- Arduino-EPICS サンプル (KEK-EPICS Users JP, <https://cerldev.kek.jp/trac/EpicsUsersJP/wiki/epics/arduino/simpleRead>)

2.5 基本動作例 3 : EPICS-Arduino による L チカ (ON/OFF) 制御

2.5.1 検証目標

- ハードウェア (Arduino) の出力状態を遠隔制御する検証。

2.5.2 前提条件

- IOC は "**RaspberryPi**"、OPI は手元 PC の macOS、制御機器は "**Arduino Uno**"。
- Arduino-RaspberryPi 間は USB 接続、RaspberryPi-mac 間は LAN ケーブルで接続する。
- Arduino の 2 番端子-GND 端子間に 1K Ω 抵抗 x2 個、LED を直列に接続する。2 番端子の出力によって LED を点灯させる。
- 制御モジュールとして、"**StremDevice**" を使用し、その他、**asyn**、"**pyEpics**" を適宜使用する。
- 作業ディレクトリ : `${HOME}/epics/app/lightupLED/` (`${HOME}=/home/epics/`)

2.5.3 Arduino プログラム 1 : (L チカ : ON/OFF 制御) の転送

- シリアル制御 (USB) から、ASCII 文字を受け取り、文字に応じて以下の動作をする。+ 文字が "**H**" (ASCII:72 (10 進数)) であった場合 LED を 点灯 させる。+ 文字が "**L**" (ASCII:76 (10 進数)) であった場合 LED を 消灯 させる。
- Arduino プログラムは以下である。
- 下記プログラムは、**Arduino IDE** を使用して Arduino へ転送しておく。(手元 PC からでも勿論、可)

リスト 6 arduino_LEDcontrol01.ino

```
// program to control LED's luminocity by PWM.

// parameters
int pin_LED      = 2;
int powerFactor = 0.0;
int serial_bps   = 19200;
```

(次のページに続く)

(前のページからの続き)

```
int ASCII_H      = 72;
int ASCII_L      = 76;

void setup() {
    // put your setup code here, to run once:
    pinMode( pin_LED, OUTPUT );
    Serial.begin( serial_bps );
}

void loop() {
    // put your main code here, to run repeatedly:

    if ( Serial.available() > 0 ){
        char cRecv = Serial.read();

        if ( cRecv == ASCII_H ){
            digitalWrite( pin_LED, HIGH );
        }
        else if ( cRecv == ASCII_L ){
            digitalWrite( pin_LED, LOW );
        }
    }
}
```

- python からの制御テストコードは以下である。

リスト 7 test_lightupLED.py

```
# -*- coding: utf-8 -*-
import serial

com_num    = "/dev/cu.usbmodem142201"
baud_rate  = 19200

def main():

    ser = serial.Serial( com_num, baud_rate, timeout=1)
    while True:

        print( " input control command, ( 'H':on, 'L':off, 'E':quit ) >>>", end="" )
        str_cmd    = input()
        byte_cmd    = bytes( str_cmd, encoding="ascii" )

        if ( str_cmd == "E" ): break
        ser.write( byte_cmd )

    ser.close()
```

(次のページに続く)

(前のページからの続き)

```
if __name__ == '__main__':  
    main()
```

2.5.4 IOC 構築

IOC 構築 / テスト の手順

- アプリ名は、参考 URL（下記）に従い、lightupLED とする。
- また、.db やレコード名などは、全て lightupLED へ統一する。
- IOC-App 構築の手順は、以下である。
 1. ベースアプリ の作成。
 2. **configure/RELEASE** に共有するモジュールのインストールパスを追記。
 3. **"lightupLEDApp"** 内の データベース情報、及び、コンパイル用の Makefile へ追記する。
 4. **StreamDevice** 用の プロトコル を、 **protocols/lightupLED.proto** として作成する。
 5. IOC 初期化スクリプト (**iocBoot/ioclightupLED/st.cmd**) を編集し、実行可能とする。
 6. make 、 及び、 st.cmd の実行、 camonitor により、経時変化を観察する。

以下、上記手順について詳述する。

1. ベースアプリの作成

- makeBaseApp.pl を用いたベースアプリの作成

```
$ mkdir -p ~/epics/app/lightupLED  
$ cd ~/epics/app/lightupLED  
$ makeBaseApp.pl -t ioc lightupLED  
$ makeBaseApp.pl -i -t ioc lightupLED
```

2. 共通コンパイル設定事項の編集 (**configure/RELEASE**)

- **configure/RELEASE** に、共通のコンパイル設定（モジュールの場所等、）を例えば以下のように記載する。

```
ASYN      = /home/epics/epics/support/asyn  
STREAM    = /home/epics/epics/support/StreamDevice
```

3. データベースファイルとコンパイルの準備

- データベース及び使用するモジュールの情報を記載し、`${HOME}/epics/app/lightupLED/lightupLEDAp/Db/lightupLED.db` を作成する。

リスト 8 lightupLED.db

```
1 record( stringout, "epics:lightupLED" )
2 {
3     field( DESC, "var to light up LEDs" )
4     field( DTYP, "stream" )
5     field( OUT , "@lightupLED.proto putval PS1" )
6 }
```

- データベースのコンパイル対象として、上記の"lightupLED.db"を追加。

```
@ lightupLEDAp/Db/Makefile

DB += lightupLED.db
```

- その他モジュールを利用する場合は、IOC の通信コードのコンパイルに使用するモジュール情報を、"lightupLEDAp/src/Makefile" に記載し、コンパイルできるようにする。

```
@ lightupLEDAp/src/Makefile

lightupLED_DBD += stream.dbd
lightupLED_DBD += asyn.dbd
lightupLED_DBD += drvAsynSerialPort.dbd

lightupLED_LIBS += stream
lightupLED_LIBS += asyn
```

4. StreamDevice の設定ファイル ("protocol") の作成

- ディレクトリ "protocols"を作成し、StreamDevice の入出力情報を記載する。

```
$ mkdir $HOME/epics/app/lightupLED/protocols
```

リスト 9 lightupLED.proto

```
1 Terminator = CR LF;
2
3 putval{
4     out "%s";
5 }
```

5. IOC 初期化スクリプト "st.cmd" の編集

- IOC 初期化スクリプト (iocBoot/ioclightupLED/st.cmd) に、以下の情報を記載する。

リスト 10 st.cmd

```
#!/../bin/linux-arm/lightupLED

#- You may have to change lightupLED to something else
#- everywhere it appears in this file

< envPaths
# -- n.k. -- #
epicsEnvSet("STREAM_PROTOCOL_PATH", "...../protocols")

cd "${TOP}"

## Register all support components
dbLoadDatabase "dbd/lightupLED.dbd"
lightupLED_registerRecordDeviceDriver pdbbase

## Load record instances
#dbLoadRecords("db/xxx.db", "user=epics")

# -- n.k. -- #
dbLoadRecords("db/lightupLED.db")
drvAsynSerialPortConfigure("PS1", "/dev/ttyACM0")
asynSetOption("PS1", 0, "baud", "19200")

cd "${TOP}/iocBoot/${IOC}"
iocInit

## Start any sequence programs
#seq sncxxx, "user=epics"
```

Warning: (陰路事項) dbLoadRecord, dbLoadDatabase の順番が逆になったりすると、うまく動作しない。しかも、".db"ファイルの1行目がおかしいというエラーメッセージがでるので、ミスリー

ディングである。st.cmd 前後の状態も確認すべきである。

- スクリプトに実行権限を与えておく。

```
$ chmod u+x $HOME/epics/app/lightupLED/iocBoot/ioclightupLED/st.cmd
```

6. make 及び、初期化スクリプト "st.cmd" の実行

- ベースディレクトリにて make する。

```
$ cd $HOME/epics/app/lightupLED/  
$ make distclean  
$ make
```

- 初期化スクリプトを実行する。

```
$ cd $HOME/epics/app/lightupLED/iocBoot/ioclightupLED/  
$ sudo ./st.cmd
```

2.5.5 ADC の動作状況の確認

ローカルからの camonitor

- 別コンソールを立ち上げて、以下コマンドを実行

```
epics@raspberrypi: ~ $ caput epics:lightupLED "H"      ( H も可 )
```

OPI (手元 macOS) からの CA

- pyEpics から CA.

```
$ python3  
>>> import epics  
>>> epics.caput( "epics:lightupLED", "H" )
```

- OPI から IOC を介して、"L チカ" を実施することができた。

2.5.6 参考 URL

- Arduino-EPICS サンプル (KEK-EPICS Users JP, <https://cerldev.kek.jp/trac/EpicsUsersJP/wiki/epics/arduino/simpleRead>)
- Github:inigoalonso/setup-epics-serial-arduino (arduino-EPICS <https://gist.github.com/inigoalonso/99d9076c672661a4b821>)
- StreamDevice -protocol Files- (<https://paulscherrerinstitute.github.io/StreamDevice/protocol.html>)

2.6 基本動作例 4 : EPICS-Arduino による LED-PWM 調光制御

2.6.1 検証目標

- ハードウェア (Arduino) の出力状態を PWM 制御する検証.

2.6.2 前提条件

- 前回同様.
- PWM で LED を調光する.

2.6.3 Arduino プログラム : (PWM 制御) の転送

- シリアル制御 (USB) から、明るさ信号を送信し、PWM 制御する.
- Arduino プログラムは以下である.

リスト 11 arduino_LEDcontrol_02.ino

```
// arduino PWM output program //
```

```
int    pin_LED    = 5;  
int    usb_bpm    = 19200;  
  
void setup() {  
    // put your setup code here, to run once:  
    pinMode( pin_LED, OUTPUT );  
    Serial.begin( usb_bpm );  
}  
  
void loop() {
```

(次のページに続く)

(前のページからの続き)

```
// put your main code here, to run repeatedly:

if ( Serial.available() > 0 ){
  int iRecv = Serial.parseInt();
  if ( iRecv > 255 ){
    iRecv = 255;
  } else if ( iRecv < 0 ){
    iRecv = 0;
  }
  analogWrite( pin_LED, iRecv );
}
}
```

- python からの制御テストコードは以下である.

リスト 12 test_pwmLED.py

```
# -*- coding: utf-8 -*-
import serial

com_num    = "/dev/cu.usbmodem142201"
baud_rate  = 19200

def main():

    ser = serial.Serial( com_num, baud_rate, timeout=1)
    while True:

        print( " input PWM control value ( 0 < val < 255, or type quit ) >>> ", end="" )
        ↵
        int_cmd    = input()
        byte_cmd   = bytes( int_cmd, encoding="ascii" )

        if ( int_cmd == "quit" ): break
        ser.write( byte_cmd )

    ser.close()

if __name__ == '__main__':
    main()
```

2.6.4 IOC 構築

IOC 構築 / テスト の手順

- アプリ名は、参考 URL（下記）に従い、pwmLED とする。
- また、.db やレコード名などは、全て pwmLED へ統一する。
- IOC-App 構築の手順は、以下である。
 1. ベースアプリ の作成。
 2. **configure/RELEASE** に共有するモジュールのインストールパスを追記。
 3. "**pwmLEDApp**" 内の データベース情報、及び、コンパイル用の Makefile へ追記する。
 4. **StreamDevice** 用の プロトコル を、 **protocols/pwmLED.proto** として作成する。
 5. IOC 初期化スクリプト (**iocBoot/iocpwmLED/st.cmd**) を編集し、実行可能とする。
 6. make 、及び、st.cmd の実行、camonitor により、経時変化を観察する。

以下、上記手順について詳述する。

1. ベースアプリの作成

- makeBaseApp.pl を用いたベースアプリの作成

```
$ mkdir -p ~/epics/app/pwmLED
$ cd ~/epics/app/pwmLED
$ makeBaseApp.pl -t ioc pwmLED
$ makeBaseApp.pl -i -t ioc pwmLED
```

2. 共通コンパイル設定事項の編集 (**configure/RELEASE**)

- **configure/RELEASE** に、共通のコンパイル設定（モジュールの場所等、）を例えば以下のように記載する。

```
ASYN    = /home/epics/epics/support/asyn
STREAM  = /home/epics/epics/support/StreamDevice
```

3. データベースファイルとコンパイルの準備

- データベース及び使用するモジュールの情報を記載し、`${HOME}/epics/app/pwmLED/pwmLEDAp/Db/pwmLED.db`を作成する。

リスト 13 pwmLED.db

```
1 record( stringout, "epics:pwmLED" )
2 {
3     field( DESC, "pwm of LED" )
4     field( DTYP, "stream" )
5     field( OUT , "@pwmLED.proto putval PS1")
6     field( SCAN, "I/O Intr" )
7 }
```

- データベースのコンパイル対象として、上記のを追加。

```
@ pwmLEDAp/Db/Makefile

DB += pwmLED.db
```

- その他モジュールを利用する場合は、IOC の通信コードのコンパイルに使用するモジュール情報を、`"pwmLEDAp/src/Makefile"`に記載し、コンパイルできるようにする。

```
@ pwmLEDAp/src/Makefile

pwmLED_DBD += stream.dbd
pwmLED_DBD += asyn.dbd
pwmLED_DBD += drvAsynSerialPort.dbd

pwmLED_LIBS += stream
pwmLED_LIBS += asyn
```

4. StreamDevice の設定ファイル ("protocol") の作成

- ディレクトリ `"protocols"`を作成し、StreamDevice の入出力情報を記載する。

```
$ mkdir $HOME/epics/app/pwmLED/protocols
```


リスト 14 pwmLED.proto

```
1 Terminator = CR LF;  
2  
3 putval{  
4     out "%s";  
5 }
```

5. IOC 初期化スクリプト "st.cmd" の編集

- IOC 初期化スクリプト (iocBoot/iocpwmLED/st.cmd) に、以下の情報を記載する。

リスト 15 st.cmd

```
#!/../bin/linux-arm/pwmLED  
  
#- You may have to change pwmLED to something else  
#- everywhere it appears in this file  
  
< envPaths  
epicsEnvSet("STREAM_PROTOCOL_PATH", "...../protocols")  
  
cd "${TOP}"  
  
## Register all support components  
dbLoadDatabase "dbd/pwmLED.dbd"  
pwmLED_registerRecordDeviceDriver pdbbase  
  
## Load record instances  
#dbLoadRecords("db/xxx.db", "user=epics")  
  
dbLoadRecords("db/pwmLED.db" )  
drvAsynSerialPortConfigure ("PS1", "/dev/ttyACM0")  
asynSetOption("PS1", 0, "baud", "19200" )  
  
cd "${TOP}/iocBoot/${IOC}"  
iocInit  
  
## Start any sequence programs  
#seq sncxxx, "user=epics"
```

Warning: (陰路事項) dbLoadRecord, dbLoadDatabase の順番が逆になると、うまく動作しない。しかも、".db"ファイルの1行目がおかしいというエラーメッセージがでるので、ミスリーディングである。st.cmd 前後の状態も確認すべきである。

- スクリプトに実行権限を与えておく.

```
$ chmod u+x $HOME/epics/app/pwmLED/iocBoot/iocpwmLED/st.cmd
```

6. make 及び、初期化スクリプト "st.cmd" の実行

- ベースディレクトリにて make する.

```
$ cd $HOME/epics/app/pwmLED/  
$ make distclean  
$ make
```

- 初期化スクリプトを実行する.

```
$ cd $HOME/epics/app/pwmLED/iocBoot/iocpwmLED/  
$ sudo ./st.cmd
```

2.6.5 LED の PWM 制御状況の確認

ローカルからの camonitor

- 別コンソールを立ち上げて、以下コマンドを実行

```
epics@raspberrypi: ~ $ caput epics:pwmLED 0    ( H も可 )  
epics@raspberrypi: ~ $ caput epics:pwmLED 10   ( H も可 )  
epics@raspberrypi: ~ $ caput epics:pwmLED 20   ( H も可 )  
epics@raspberrypi: ~ $ caput epics:pwmLED 240  ( H も可 )
```

- 明るさが変更されることを確認した.

OPI (手元 macOS) からの CA

- pyEpics から CA.

```
$ python3  
>>> import epics  
>>> epics.caput( "epics:pwmLED", 10 )  
>>> epics.caput( "epics:pwmLED", 240 )
```

- OPI から IOC を介して、"LED の PWM 制御" を実施することができた.

2.6.6 参考 URL

- Arduino-EPICS サンプル (KEK-EPICS Users JP, <https://cerldev.kek.jp/trac/EpicsUsersJP/wiki/epics/arduino/simpleRead>)
- Github:inigoalonso/setup-epics-serial-arduino (arduino-EPICS <https://gist.github.com/inigoalonso/99d9076c672661a4b821>)
- StreamDevice -protocol Files- (<https://paulscherrerinstitute.github.io/StreamDevice/protocol.html>)
- Arduino PWM 制御 (<https://deviceplus.jp/arduino/how-to-control-led-with-arduino-pwm/>)

第 3 章

References

- EPICS-controls (newer version of website, <https://epics-controls.org/>)
- EPICS (older version of website, <https://epics.anl.gov/>)
- EPICS Users JP (KEK EPICS wiki, <https://cerldev.kek.jp/trac/EpicsUsersJP>)
- その他資料リンク (EPICS Users JP, <https://cerldev.kek.jp/trac/EpicsUsersJP/wiki/intro>)
- Getting-Started EPICS controls (<https://docs.epics-controls.org/projects/how-tos/en/latest/getting-started/installation.html>)
- 参 考 ノ ー ト : " https://note.com/dev_associate/n/nfa4605c70f60", " https://note.com/dev_associate/n/nd886d700b10a"
- OPI/IOC 通信時のポート番号、IP アドレスの設定 (https://epics.anl.gov/EpicsDocumentation/AppDevManuals/ChannelAccess/cadoc_4.htm)
- Arduino-EPICS サンプル (KEK-EPICS Users JP, <https://cerldev.kek.jp/trac/EpicsUsersJP/wiki/epics/arduino/simpleRead>)
- Github:inigoalonso/setup-epics-serial-arduino (arduino-EPICS <https://gist.github.com/inigoalonso/99d9076c672661a4b821>)
- StreamDevice -protocol Files- (<https://paulscherrerinstitute.github.io/StreamDevice/protocol.html>)
- Arduino PWM 制御 (<https://deviceplus.jp/arduino/how-to-control-led-with-arduino-pwm/>)
- EPICS Record Reference (<https://epics.anl.gov/base/R7-0/6-docs/RecordReference.html>)