EPICS 講習会 実習資料

Revision	1.1
Status	RELEASE
Author	KEKB 制御グループ
Last modification	2015 年 3 月 23 日
Created	2015 年 2 月 6 日

Creation	*Date*	*Author*	*Section*	*Modification*
0.0	2013/06/24	中本 崇	All	実習書作成

Revision	*Date*	*Author*	*Section*	*Modification*
0.1	2015/02/06	浅野 和哉	1,2,3,4	「実習環境の解説」、 「実習 1」～「実習 3」の変更
0.2	2015/02/26	廣瀬 雅哉	2,3,4	新規実習の追加 実習手順の変更
1.0	2015/03/17	廣瀬 雅哉	All	全体の見直し
1.1	2015/03/23	廣瀬 雅哉	All	全体の見直し

公開範囲

本文書は、公開文書です。本文書の全体または一部は、「対象者」に挙げられている方であれば、どなたでも閲覧することができます。

目的

本文書は、KEKにおけるEPICS講習会の実習のための資料です。

対象者

KEKにおけるEPICS講習会の参加者

参考文献

[1] EPICS Input/Output Controller Application Developer’s Guide:
http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide/

[2] EPICS Record Reference Manual:
http://www.aps.anl.gov/epics/wiki/index.php/RRM_3-14

[3] KEKB制御ネットワークにおけるレコード命名規則:
http://kekb-co-web.kek.jp/top_contents/RecordName.htm

[4] CSS 初心者向け講習会:
http://www-linac.kek.jp/cont/epics/css/CSS_beginners_20120208.pdf
http://www-linac.kek.jp/cont/epics/css/CSS_beginners_20120215.pdf

[5] StreamDevice:
http://epics.web.psi.ch/software/streamdevice/doc/

[6] asynDriver:
http://www.aps.anl.gov/epics/modules/soft/asyn/
[7] State Notation Language and Sequencer
http://www-csr.bessy.de/control/SoftDist/sequencer/

凡例

本文書中において、以下のように“host$”で始まる行は、実習環境上で実行するbash コマンドを表して います。

```
host$ command
```

bashコマンド中の“~”は自身のホームディレクトリを表しています。

以下のように、“>”で始まる行は、iocsh（IOCシェル）のコマンドを表しています。

```
> command
```

目次

1. 実習環境の解説

1.1. 実習環境（pc-cont-kyoka）へのログイン

本実習では、所内ネットワーク上にある pc-cont-kyoka(130.87.82.39) を実習環境として使用します。このマシンにログインするには、端末を立ち上げ、以下のコマンドを実行してください。ただし、*user*は自身のユーザー名に置き換えてください。

```
$ ssh -Y user@pc-cont-kyoka.kek.jp
```

パスワードの入力が求められますので、パスワードを入力しEnterキーを押下してください。パスワードはユーザー名と同じです。ログインをしたら、passwdコマンドでパスワードを変更してください。

1.2. シェル

本実習で使用するアカウントは全てbashシェルで作成しています。

自身の使用しているシェルは、以下のコマンドを実行することにより確認できます。

```
host$ echo $SHELL
```

“/bin/bash”と表示されているのであれば、お使いのシェルはbashです。

1.3. 環境変数の設定

EPICS に関連するコマンドを実行するために、PATH と EPICS に関する環境変数の設定が必要になります。通常、これは制御システムの管理者により行われます。
以下のコマンドを実行し、実習環境向けの設定を行ってください。

```
host$ source /opt/epics/R314.12.4/set_path.bash
```

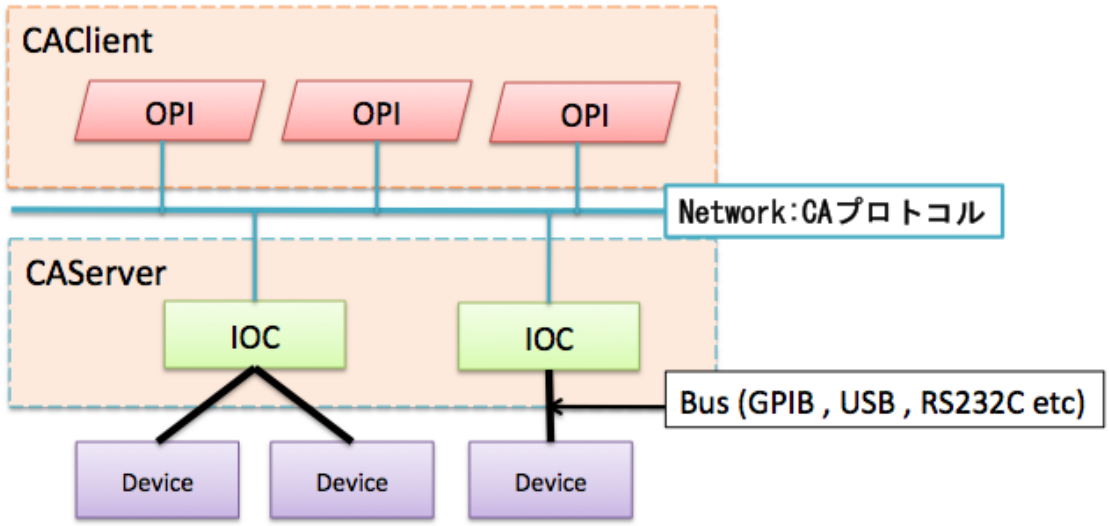
上記コマンドは、ssh でログインする度に実行する必要があります。
この手順を簡略化したい方は、~/.bashrc をエディタで開き、最終行あたりに上記コマンドを追加してください。

本実習では複数の端末を使用することになるので、xtermを使用して2つほど端末を立ち上げておいてください。

2. 実習 1: コマンドによるレコードアクセス

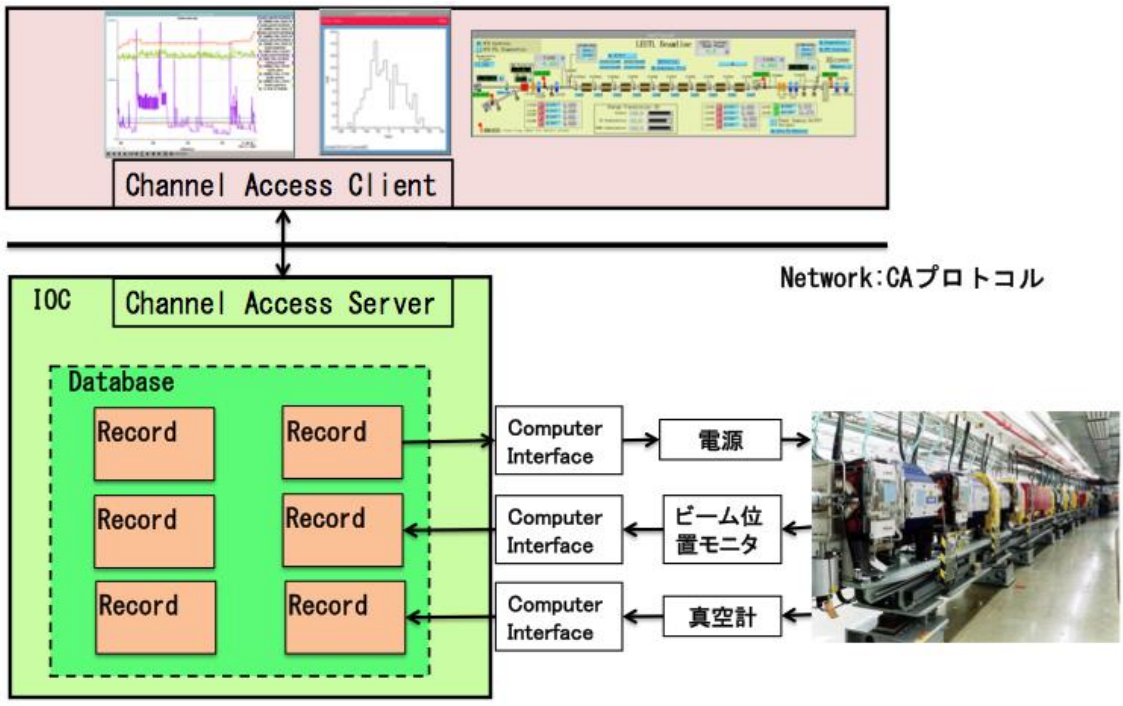
EPICS (Experimental Physics and Industrial Control System) とは、
加速器や物理実験装置の制御システムを開発・実装するためのソフトウェアです。
EPICS は IOC (Input Output Controller) 経由でハードウェアデバイスを制御する事ができ、このデータのやり取りは CA (Channel Access) という通信プロトコルで行われます。
IOC は入出力制御装置のようなものであり、EPICS の中で中心的な役割を持つデータベースを有しています。
データベースはレコードの集まりから構成され、そのレコードの一つ一つはデバイスを制御するために必要な情報 (ステータス, パラメータなど) を保持しています。
CA とは、クライアント (OPI) とサーバー (IOC) 又は IOC 同士でデータにアクセスするための EPICS の通信プロトコルです。

<EPICS 構成図>



ネットワーク分散型のクライアントサーバモデル

〈EPICS 制御システム 例〉



2.1. 目的

本実習では、すでに用意されている IOC 上のレコードに caget, caput, camonitor コマンドでアクセスします。この IOC は定電流電源を制御するためのものです（実際には電源をシミュレーションしています)。本実習では、コマンドを使用してこの電源を操作することで、どのようにしてコマンドを用いてレコード値の get、put、monitor を行うことができるのかを学習します。

2.2. 演習

以下のコマンドを実行し、電流値を 10mA に設定してください。

ただし、*user* は自身のユーザー名に置き換えてください。

例えば、user-1 の方は“ET_user-1”となります。

```
host$ caput ET_user:PS:SET_CURRENT 10
```

電流値を設定した後に、実際に現在の電流値がどのようになっているのかを確認してください。

```
host$ caget ET_user:PS:READ_CURRENT
```

電流値がどのように変化しているのかを監視してください。

```
host$ camonitor ET_user:PS:READ_CURRENT
```

別の端末で、電流値を 20mA に設定してください。

```
host$ caput ET_user:PS:SET_CURRENT 20
```

このとき、camonitor による値がどのように変化するのかを確認してください。

最後に、camonitor コマンドを Ctrl+C により終了させてください。

💡 ET_*user* の ET は、 Epics Training の略称です。

3. 実習 2: EPICS IOC の作成

3.1. 目的

本実習では、EPICS IOCの作成方法[1]について学習します。

EPICS Baseに元々用意されているサンプルアプリケーションを元にEPICS IOCを作成し、以下の事項を習得します。

- EPICS IOCの作成、ビルド、実行
- iocshの使用方法
- アプリケーションを生成するのに使用するmakeBaseApp.plというPerlスクリプトの使用方法

3.2. 演習

ここでは、myexampleAppという名前のEPICSアプリケーションと、iocmyexampleという名前のIOCを作成します。

3.2.1. EPICS_HOST_ARCH の確認

以下のコマンドを実行し、EPICS_HOST_ARCH環境変数に何が設定されているのかを確認してください。

```
host$ echo $EPICS_HOST_ARCH
```

ログインしているマシンのOSとCPUアーキテクチャ（例: linux-x86_64、win32_x86）が表示されるはずです。

何も表示されない場合には、EPICS_HOST_ARCHに適切な値を設定してください。現在のOSとCPUアーキテクチャを確認するには、以下のコマンドを実行してください。

```
host$ /opt/epics/R314.12.4/base-3.14.12.4/startup/EpicsHostArch
```

実行した結果、例えば“linux-x86_64”と表示された場合は、以下のようにしてEPICS_HOST_ARCH環境変数を設定してください。

```
host$ export EPICS_HOST_ARCH=linux-x86_64
```

3.2.2. サンプルアプリケーションの作成

以下のコマンドにより、サンプルアプリケーションを~/lecture/myexample以下に作成することができます。

```
host$ mkdir -p ~/lecture/myexample
host$ cd lecture/myexample
host$ makeBaseApp.pl -t example myexample
host$ makeBaseApp.pl -i -t example myexample
```

最後のコマンドを実行すると、このIOCはどのアプリケーションをブートするのかを聞かれます。今回は1つしかアプリケーションがありませんので、そのままEnterキーを押すだけです。

makeBaseApp.plコマンドが見つからないエラーが発生してしまった場合は、PATH環境変数が適切に設定されていない可能性があります。以下のコマンドを実行して、PATH環境変数を設定してください。

```
host$ export PATH=$PATH:/opt/epics/R314.12.4/base-3.14.12.4/
      bin/$EPICS_HOST_ARCH
```

makeBaseApp.plのオプション

-t <type>　：テンプレートを指定します。

- support ： サポートアプリケーションのビルド向けテンプレート
- ioc ： 汎用IOCアプリケーション（空のIOCアプリケーション作成）
- example ： IOCアプリケーションのサンプル　（デフォルト）
- caClient ： CAクライアントツールのテンプレート
- caServer ： CAサーバツールのテンプレート

-i IOCのブートディレクトリ作成

-l テンプレート一覧の表示

-a <arch>　OSとCPUアーキテクチャ設定

-h ヘルプ

3.2.3. 作成されたファイルの確認

この段階で、~/lecture/myexample以下にどのようなディレクトリやファイルが生成されたのかを確認してください。例えば、以下のようにlsコマンドやfindコマンドを~/lecture/myexampleディレクトリで実行すると、~/lecture/myexampleディレクトリ以下にあるディレクトリとファイルを確認することができます。

```
host$ cd lecture/myexample
host$ ls
host$ find
```

この作業は、ビルドを行う前に行ってください。これにより、ビルドをするのに必要なファイルがどういったものであるかを確認することができます。

3.2.4. アプリケーションのビルド

~/lecture/myexampleディレクトリにて、以下のコマンドを実行してください。

```
host$ make
```

ここで、~/lecture/myexample以下にどのようなディレクトリやファイルが生成されたのかを確認し、make前後でどのような違いがあるのかを確認してみましょう。

3.2.5. IOC の起動

以下のコマンドを実行し、IOCを起動してください。

```
host$ cd ~/lecture/myexample/iocBoot/iocmyexample
host$ ../../bin/linux-x86_64/myexample st.cmd
```

以下のように st.cmd ファイルを実行ファイル化して、IOC を起動することもできます。

```
host$ chmod u+x ~/lecture/myexample/iocBoot/iocmyexample/st.cmd
host$ ./st.cmd
```

3.2.6. iocsh コマンドの実行

IOCが起動したら、iocsh上でいくつかのコマンド（例: dbl、dbpr、dbpf、 dbgf等）を試してみてください。

例えば、dblコマンドを実行すると、レコードリストが表示されます。

また、help の機能が付いているので、

```
> help
```

または、

```
> help <cmd>
```

でヘルプを確認してください。ここで、<cmd>はdblなどといったコマンド名です。

helpコマンドではワイルドカードを使えるので、

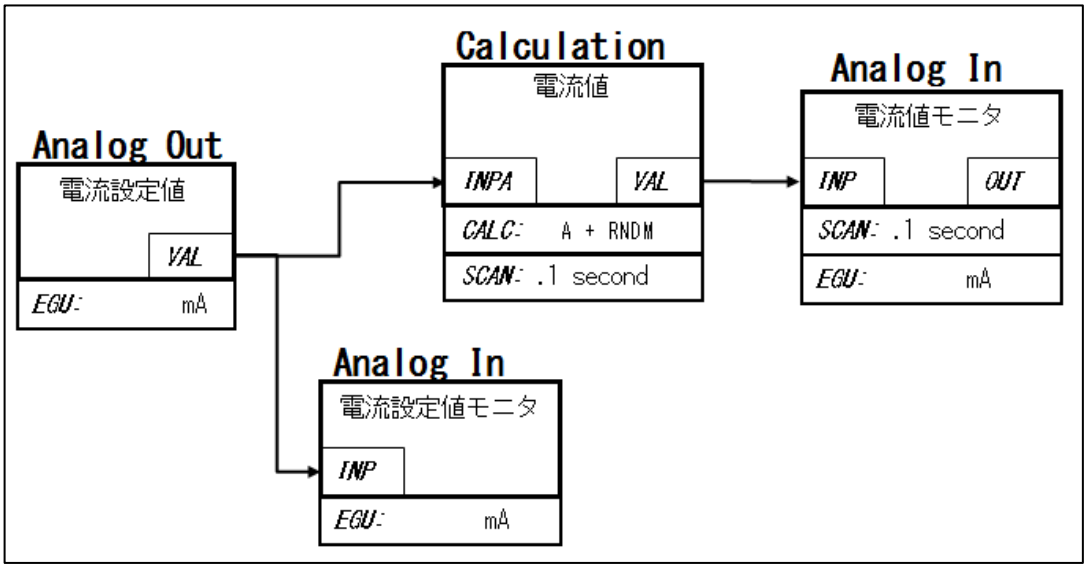
```
> help db*
```

とすれば、”db”で始まるコマンドの使用方法を見ることができます。

4. 実習 3: 電源のシミュレーション

4.1. 目的

本実習では、電源を模擬するEPICSデータベースを作成します。



- レコード名はマクロを使用し、“ET_$(user):$(name):xxxx”となるようにします。$(user)は自身のユーザー名、$(name)は電源の名前に置き換えるようにします。ここでは、電源名を PS とします。
- 開発途中の段階では、cagetやcamonitorなどを使ってデバッグをしてみることも検討してください。

4.2. 演習

ここでは、powerSupplyApp という名前の EPICS アプリケーションと、iocpowerSupply という名前の IOC を作成します。

4.2.1. 空の EPICS IOC の作成

~/lecture/powerSupplyディレクトリに空のEPICS IOCを作成するところからはじめます。以下のコマンドを実行してください。

```
host$ mkdir -p ~/lecture/powerSupply
host$ cd ~/lecture/powerSupply
host$ makeBaseApp.pl -t ioc powerSupply
host$ makeBaseApp.pl -i -t ioc powerSupply
```

これらのコマンドを実行することにより、空の EPICS IOC が作成されます。

4.2.2. EPICS データベースの作成

本実習では、dbファイル名をpowerSupply.dbとし、以下の機能を実装します。
ここで、①～④の単位はmVとします。

① 電流設定値の書き込み： レンジは0.0～100.0とします。
② 電流設定値の読み出し:
③ 電流値(疑似電流の値): 電流設定値を中心に値をランダムに上下させます。
④ 電流値の読み出し:
⑤ ON/OFFスイッチ: 1でON、0でOFFとします。
⑥ ON/OFF状態の読み出し

💡 レコードの型やフィールドの詳細は[2]を参照してください。

まず、~/lecture/powerSupply/powerSupplyApp/DbにあるMakefileをエディタで開き、“#DB += xxx.db”と記述されている行の次の行に、以下の命令を挿入してください。

```
DB += powerSupply.db
```

また、~/lecture/powerSupply/iocBoot/iocpowerSupplyディレクトリにあるst.cmdをエディタで開き、“#dbLoadRecords(...”と記述されている行の次の行に、以下の命令を挿入してください。ただし、*user*は自身のユーザー名としてください。

```
dbLoadRecords("db/powerSupply.db", "user=user, name=PS")
```

4.2.3. 電流設定値レコードの作成（①、②）

~/lecture/powerSupply/powerSupplyApp/Db ディレクトリに powerSupply.db というファイルを作成してください。

powerSupply.db をエディタで開き、以下を記述します。

```
record(ao, "ET_$(user):$(name):SET_CURRENT")
{
  field(DESC, "Set the current value")  # Description
  field(SCAN, "Passive") # Scanning Rate
  field(VAL, "0.0")      # Value
  field(DRVH, "100.0")   # Drive High
  field(DRVL, "0.0")     # Drive Low
  field(PREC, "1")       # Display Precision
  field(EGU, "mA")       # Engineering Units
  field(FLNK, "ET_$(user):$(name):SET_CURRENT_RB PP NMS")  # Forward Link
}

record(ai, "ET_$(user):$(name):SET_CURRENT_RB")
{
  field(DESC, "Read back set the current value")
  field(SCAN, "Passive")
  field(INP, "ET_$(user):$(name):SET_CURRENT")  # Input Link
  field(PREC, "1")
  field(EGU, "mA")
}
```

この段階で、アプリケーションのビルド（make）を行い、IOCを起動して、レコードの動作を確認してみましょう。

別の端末にて、以下のコマンドを実行し、現在の電流設定値のリードバック値を確認してください。ただし、*user*は自身のユーザー名に置き換えてください。

```
host$ caget ET_user:PS:SET_CURRENT_RB
```

電流設定値を10mAに設定してください。

```
host$ caput ET_user:PS:SET_CURRENT 10
```

電流設定値のリードバック値がどのようになっているのかを確認してください。

```
host$ caget ET_user:PS:SET_CURRENT_RB
```

4.2.4. 電流値レコードの作成（③、④）

powerSupply.db をエディタで開き、最終行あたりに以下の記述を追加します。

```
record(calc, "ET_$(user):$(name):CURRENT")
{
  field(DESC, "The current value")
  field(SCAN, ".1 second")
  field(CALC, "A+RNDM")  # Calculation (A is INPA value)
  field(INPA, "ET_$(user):$(name):SET_CURRENT")  # Input Link A
  field(PREC, "1")
}

record(ai, "ET_$(user):$(name):READ_CURRENT")
{
  field(DESC, "Read the current value")
  field(SCAN, ".1 second")
  field(INP, "ET_$(user):$(name):CURRENT")
  field(PREC, "1")
  field(EGU, "mA")
}
```

- calcレコードのCALCフィールドでは、“RNDM”という関数が使用できます。この関数は、0～1までの間の実数をランダムに返す関数です。

- この段階で、4.2.3同様にデバッグしてみましょう。
  以下のコマンドを実行し、現在の電流値を確認してください。
  ただし、*user*は自身のユーザー名に置き換えてください。

```
host$ caget ET_user:PS:READ_CURRENT
```

  電流設定値を20mAに設定してください。

```
host$ caput ET_user:PS:SET_CURRENT 10
```

  電流値がどのように変化しているのかを監視してください。

```
host$ camonitor ET_user:PS:READ_CURRENT
```

4.2.5. ON/OFF スイッチレコードの作成（⑤、⑥）

powerSupply.db をエディタで開き、最終行あたりに以下の記述を追加します。

```
record(bo, "ET_$(user):$(name):POWER_SW")
{
  field(DESC, "The power switch")
  field(SCAN, "Passive")
  field(VAL, "0")
  field(ZNAM, "OFF")    # Zero Name
  field(ONAM, "ON")     # One Name
  field(FLNK, "ET_$(user):$(name):POWER_SW_RB PP NMS")
}

record(bi, "ET_$(user):$(name):POWER_SW_RB")
{
  field(DESC, "Read back the power switch")
  field(SCAN, "Passive")
  field(INP, "ET_$(user):$(name):POWER_SW")
  field(ZNAM, "OFF")
  field(ONAM, "ON")
}
```

ここで、以下のように 0 で作成した“ET_$(user):$(name):CURRENT”の CALC フィールドを修正し、INPB フィールドを追加します。

```
record(calc, "ET_$(user):$(name):CURRENT")
{
  ...
  field(CALC, "B=1?A+RNDM:0")   # If B=1, random value is generated
  field(INPA, "ET_$(user):$(name):SET_CURRENT")
  field(INPB, "ET_$(user):$(name):POWER_SW")
  ...
}
```

💡 EPICSデータベースを作成し終えたら、アプリケーションのビルドを行い、IOC を起動しましょう。

4.2.6. 電源の操作

先ほど作成したIOC上のレコードにcaget, caput, camonitorコマンドでアクセスします。この電源の操作方法は非常に単純であり、電源のON/OFFと電流値の設定を行うだけで動作します。

以下のコマンドを実行し、電源の現在の ON/OFF の状況を確認してください。ただし、*user* は自身のユーザー名に置き換えてください。

```
host$ caget ET_user:PS:POWER_SW_RB
```

次に、この電源を ON してください。

```
host$ caput ET_user:PS:POWER_SW ON
```

次に、電流値を 10mA に設定してください。

```
host$ caput ET_user:PS:SET_CURRENT 10
```

電流値を設定した後に、実際に現在の電流値がどのようになっているのかを確認してください。

```
host$ caget ET_user:PS:READ_CURRENT
```

電流値がどのように変化しているのかを監視してください。

```
host$ camonitor ET_user:PS:READ_CURRENT
```

別の端末で、電流値を 20mA に設定してください。

```
host$ caput ET_user:PS:SET_CURRENT 20
```

このとき、camonitor による値がどのように変化するのかを確認してください。

最後に、camonitorコマンドをCtrl+Cにより終了させてください。

追加演習

時間に余裕がある場合には、以下の機能も実装してみましょう。

- 過電流となった場合、自動的にOFFにするようにしてください。
- 電流値が少しずつ時間をかけて設定値に近づくようにしてください。
- Busy状態（電流値が設定値に達するまでBusyとする）を表すレコードを作成してください。

5. 実習 4: CSS による操作画面の作成

5.1. 目的

本実習では、どのように EPICS IOC を操作するための画面を作成するのかを学びます。ここでは、実習 3 で作成した電源の IOC を操作するための画面を作成します。実習 3 で、コマンドで行ったことを GUI により簡単且つ直観的に行えるようにしましょう。具体的には、以下の事項を習得します。

- CSSの使用方法
- OPIファイルの作成、実行方法

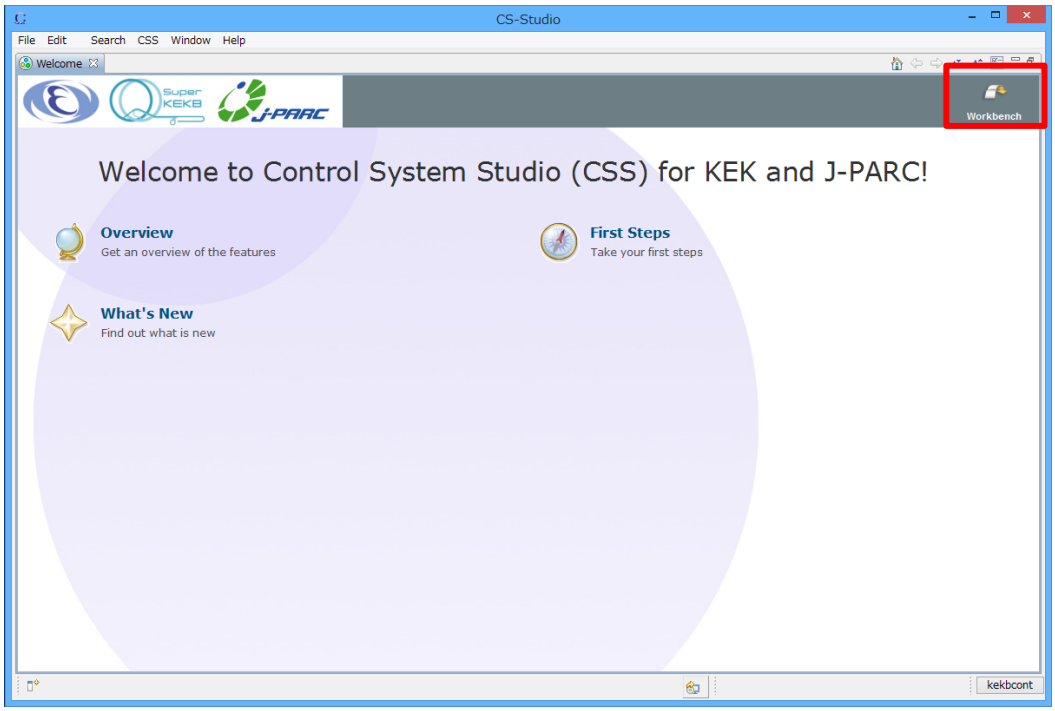
> CSS についての詳細は[4]を参照してください。

5.2. デモンストレーション

5.2.1. CSS の起動

以下のコマンドを実行し、CSS を起動してください。

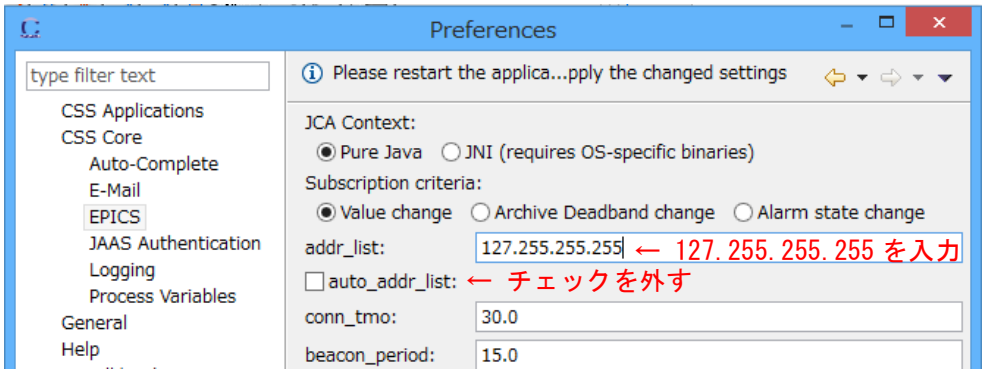
```
host$ /opt/css/kek/3.2.16/css
```



> 初回起動時は、上記の Welecome 画面が表示されます。Workbench というアイコンをクリックし、Workbench 画面に切り替えてください。

5.2.2. Preferences の設定

実習 3 で作成した IOC 上のレコードにアクセスするため、CSS の CA に関する設定を変更します。 CSS のメニューより Edit→**Preferences...**→**CSS Core**→**EPICS** を選択します。

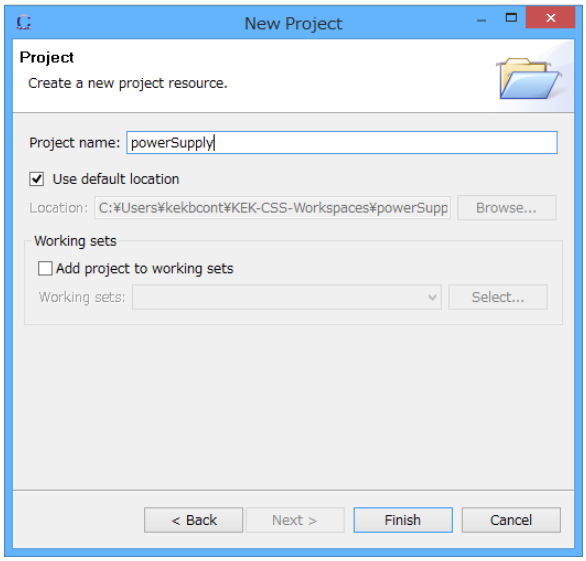


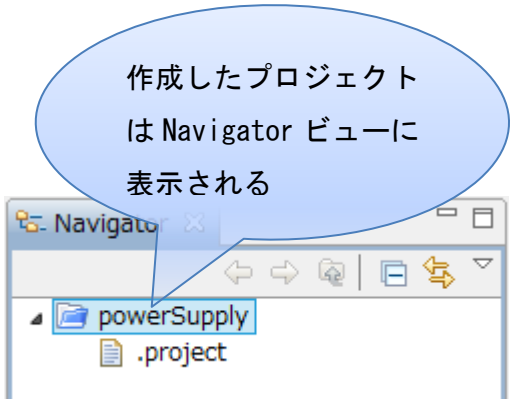
設定後、CSS のメニューより File→**Restart CSS** を選択し、CSS を再起動します。

5.2.3. プロジェクトの作成

CSS のメニューより File→**New**→**General**→**Project** を選択します。

プロジェクト名には powerSupply と入力し、Finish を押下してください。





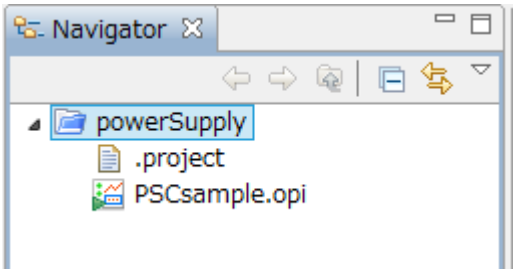
💡 プロジェクトは、Use default location のところに作られます。

5.2.4. サンプル OPI のコピー

別の端末にて、以下のコマンドを実行し、サンプル OPI ファイルを先ほど作成したプロジェクト内にコピーしてください。

```
host$ cd ~/CSS-Workspaces/Default/powerSupply
host$ cp /opt/sample/css_3.2.16/PSCsample.opi ./
```

コピーし終えたら、Navigator ビューで右クリックコンテキストメニューより **Refresh** を実行します。

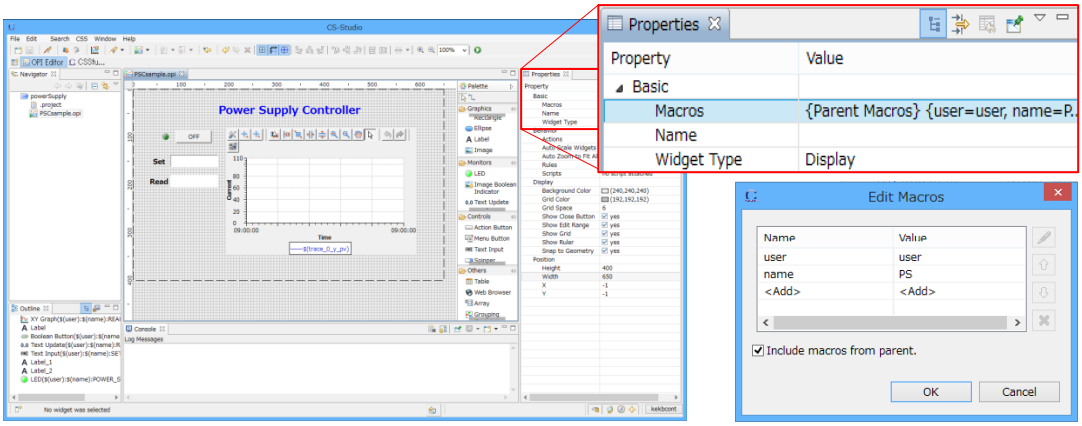


5.2.5. サンプル OPI Macro の変更

PSCsample.opi では、ディスプレイの Macro として$(user)と$(name)が定義されています。この Macro の値は適切に変更する必要があります。

Navigator ビューで PSCsample.opi を右クリックし、**Open With→OPI Editor** を選択します。次に、ディスプレイの Properties ビューより、Basic→Macros プロパティの Value をクリックします。

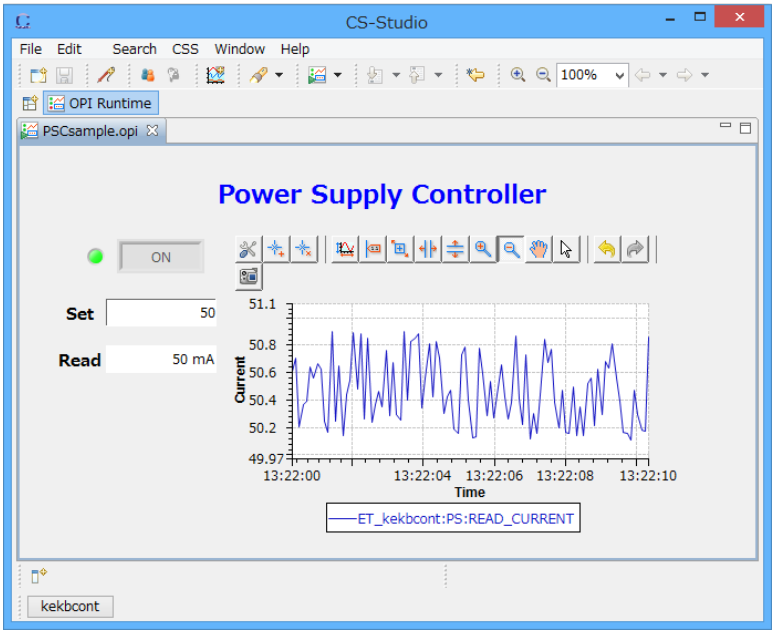
本実習では、$(user)の値を自身のユーザー名、$(name)の値を PS とします。



CSS では OPI Editor 向けの Perspective が用意されています。CSS のメニューより Window→**Open Perspective→Other→OPI Editor** を選択します。

5.2.6. サンプル OPI の実行

CSS 画面右上の  をクリックし、OPI を実行してください。



Navigator ビューで OPI ファイルを右クリックし、**Open With→OPI Runtime** を選択しても実行できます。

5.3. 演習

サンプル OPI では、Text などのシンプルなウィジェットで構成されていました。

CSS では、他にも多くのウィジェットが用意されています。ここでは、他のウィジェットを使用し、ユーザビリティを向上させた OPI を新たに作成してみましょう。

5.3.1. OPI ファイルの作成

Navigator ビューにて、5.2.3 で作成したプロジェクト“powerSupply”を右クリックし、**New → OPI File** を選択します。

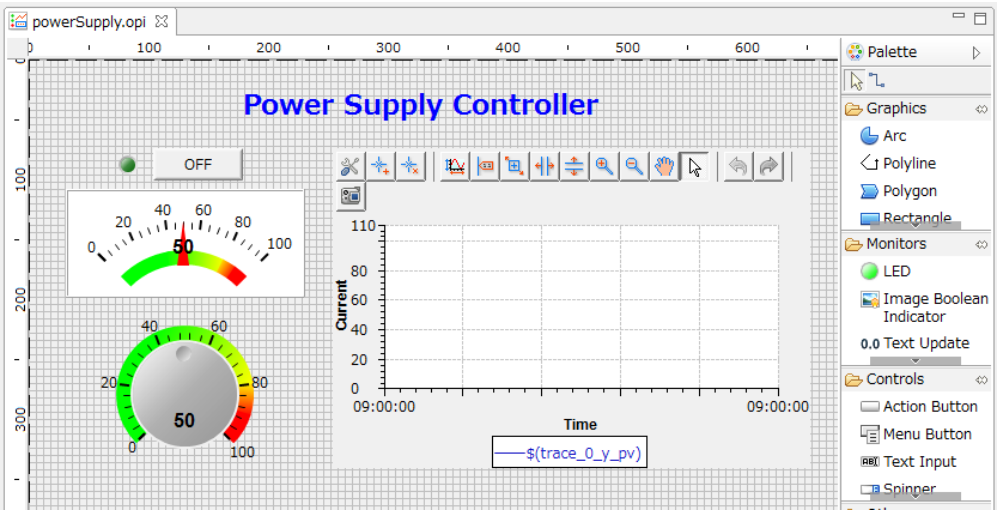
OPI ファイル名は powerSupply とします。

5.3.2. ウィジェットの配置

以下の画面を参考に、それぞれのウィジェットを配置してください。

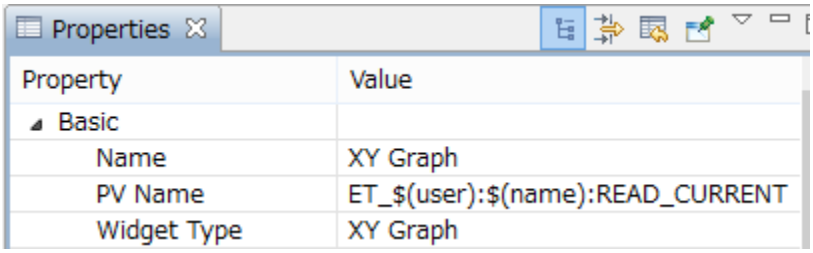
使用するウィジェットは以下の通りです。

- LED
- Boolean Button
- Knob
- Meter
- XY Graph



Palette よりウィジェットをドラッグ＆ドロップすることで、画面内にウィジェットを配置することができます。

表示させたいレコードをウィジェットプロパティの **PV Name** に設定します。



OPI を作成し終えたら、Ctrl+S により保存し、OPI を実行してみましょう。

6. 実習 5: 外部機器と通信する

6.1. 目的

本実習では、LANやGPIBなどのインタフェースで接続された機器とEPICSで通信する方法について学習します。

ここでは、Agilent 34410A測定器と通信するEPICSデータベースを作成します。

Agilent 34410A測定器では、リモート・インタフェースが搭載されており、SCPIコマンドを使った測定器のプログラミングが可能です。

本実習では、LAN インタフェース、StreamDevice[4]を使用します。

6.2. 演習

ここでは、ag34410AApp という名前の EPICS アプリケーションと、
iocag34410A という名前の IOC を作成します。

6.2.1. 空の EPICS アプリケーション作成

~/lecture/ag34410Aディレクトリに空のEPICS IOCを作成するところからはじめます。

以下のコマンドを実行してください。

```
host$ mkdir -p ~/lecture/ag34410A
host$ cd ~/lecture/ag34410A
host$ makeBaseApp.pl -t ioc ag34410A
host$ makeBaseApp.pl -i -t ioc ag34410A
```

これらのコマンドを実行することにより、空のEPICS IOCが作成されます。

6.2.2. デバイスサポートの環境変数を設定

① ~/lecture/ag34410A/configureディレクトリにあるRELEASEをエディタで開き、“#SNCSEQ=$(EPICS_BASE)/../modules/soft/seq”と記述されている行の次の行に、以下の命令を挿入してください。

```
ASYN=/opt/epics/R314.12.4/modules/asyn4-26
STREAM=/opt/epics/R314.12.4/modules/stream-2.6
```

💡 これは、asynDriver及びStreamDeviceがどこにインストールされているのかをEPICSビルドシステムに教えるためです。

② ~/lecture/ag34410A/ag34410AApp/srcディレクトリにregistrar.dbdというファイルを作成してください。

registrar.dbdをエディタで開き、以下を記述します。

```
registrar(drvAsynIPPortRegisterCommands)
registrar(vxi11RegisterCommands)  # For Gpib
```

💡 ここではiocshコマンドを登録しています。

③ ~/lecture/ag34410A/ag34410AApp/srcディレクトリにあるMakefileをエディタで開き、“ag34410A_DBD+=base.dbd”と記述されている行の次の行に、以下の命令を挿入してください。

```
ag34410A_DBD += asyn.dbd
ag34410A_DBD += stream.dbd
ag34410A_DBD += registrar.dbd

ag34410A_LIBS += asyn
ag34410A_LIBS += stream
```

6.2.3. サンプルプログラムのコピー

本実習では、6.2.4 と 6.2.5 で説明する devag34410A.db と devag34410A.proto というファイルを以下のディレクトリに用意しています。

```
/opt/sample/epics_3.14.12.4/ag34410A
```

6.2.4. EPICS データベースの作成

~/lecture/ag34410A/ag34410AApp/Dbディレクトリにdevag34410A.dbというファイルをコピーしてください。このファイルの中身は以下のようになります。

① 現在値の読み出し

```
record(ai, "ET_$(user):$(name):READ_VOLTAGE")
{
  field(DESC, "Read the voltage value")
  field(SCAN, "1 second")
  field(DTYP, "stream")    # Device Type
  field(INP, "@devag34410A.proto getVoltage $(port) $(A)")
  field(PREC, "6")
}
```

② レンジの書き込み

```
record(ao, "ET_$(user):$(name):SET_RANGE")
{
  field(DESC, "Set range")
  field(SCAN, "Passive")
  field(DTYP, "stream")
  field(OUT, "@devag34410A.proto setRange $(port) $(A)")
  field(FLNK, "ET_$(user):$(name):READ_RANGE PP NMS")
  field(PREC, "1")
}
```

③ レンジの読み出し

```
record(ai, "ET_$(user):$(name):READ_RANGE")
{
  field(DESC, "Read range")
  field(SCAN, "Passive")
  field(DTYP, "stream")
  field(INP, "@devag34410A.proto getRange $(port) $(A)")
  field(PREC, "1")
  field(PINI, "YES")  # Process at Initialization
}
```

EPICSデータベースを作成し終えたら、~/lecture/ag34410A/ag34410AApp/Dbにある Makefileをエディタで開き、“#DB += xxx.db”と記述されている行の次の行に、以下の命令を挿入してください。

```
DB += devag34410A.db
```

6.2.5. プロトコルファイルの作成

~/lecture/ag34410A/ag34410AApp/Dbディレクトリに、devag34410A.protoというファイルをコピーしてください。このファイルの中身は以下のようになります。

```
OutTerminator = CR LF;   # Output terminator
InTerminator  = LF;      # Input terminator

getVoltage {
  out "READ?";    # Send
  in  "%f";       # Receive
}

setRange {
  out "SENS:VOLT:DC:RANG %f";
}

getRange {
  out "SENS:VOLT:DC:RANG?";
  in  "%f";
}
```

6.2.6. LAN（TCP/IP）の設定

~/lecture/ag34410A/iocBoot/iocag34410Aディレクトリにあるst.cmdをエディタで開き、“#dbLoadRecords(...”と記述されている行の次の行に、以下の命令を挿入してください。ただし、*user*は自身のユーザー名としてください。

```
epicsEnvSet("STREAM_PROTOCOL_PATH", "../../ag34410AApp/Db")
dbLoadRecords("db/devag34410A.db", "user=user, name=ag34410A, port=L0,
A=0")  # "A" is GPIB address
drvAsynIPPortConfigure("L0", "192.168.0.20:5025", 0, 0, 0)
```

💡 LAN/GPIBを使用する

```
E5810Reboot("192.168.1.101", 0)  # Reboot E5810(LAN/GPIB)
vxi11Configure("L0", "192.168.1.101", 1, 1000, "gpib0")
```

💡 デバッグを使用する

```
asynSetTraceMask("L0", 0, 0x9)
asynSetTraceIOMask("L0", 0, 0x2)
```

6.2.7. ビルドと IOC の実行

~/lecture/ag34410A ディレクトリにて、以下のコマンドを実行し、アプリケーションのビルドを行ってください。

```
host$ make
```

6.2.8. Agilent 34410A 測定器の操作

先ほど作成したIOC上のレコードにcaget, caput, camonitorコマンドでアクセスします。

今回の操作は、DC電圧値の読み出しとDC電圧レンジの設定を行います。

以下のコマンドを実行し、現在の DC 電圧値がどのようになっているのかを確認してください。ただし、*user* は自身のユーザー名に置き換えてください。

```
host$ caget ET_user:ag34410A:READ_VOLTAGE
```

次に、現在の DC 電圧レンジがどのようになっているのかを確認してください。

```
host$ caget ET_user:ag34410A:READ_RANGE
```

次に、DC 電圧レンジを 10V に設定してください。

```
host$ caput ET_user:ag34410A:SET_RANGE 10
```

DC 電圧レンジを設定した後に、実際に現在の DC 電圧値がどのようになっているのかを確認してください。

```
host$ caget ET_user:ag34410A:READ_VOLTAGE
```

DC 電圧値がどのように変化しているのかを監視してください。

```
host$ camonitor ET_user:ag34410A:READ_VOLTAGE
```

別の端末で、DC 電圧レンジを 100V に設定してください。

```
host$ caput ET_user:ag34410A:SET_RANGE 100
```

このとき、camonitor による値がどのように変化するのかを確認してください。

最後に、camonitor コマンドを Ctrl+C により終了させてください。

6.2.9. CSS による操作画面の作成

先ほど作成した IOC を操作するための画面を作成してください。

> /opt/sample/css_3.2.16 ディレクトリに ag34410A.opi という OPI ファイルが用意されています。
> 作成が難しいと感じた場合は、ag34410A.opi をベースに作成してみましょう。

6.3. 付録：asynDriver を使用する場合

asynDriveを使用する場合は、以下のようにCプログラムファイル（devag34410A.c）及び定義ファイル（devag34410A.dbd）を~/lecture/ag34410A/ag34410AApp/srcディレクトリに作成します。詳細は0を参照してください。

devag34410A.dbd

```
# device(<record type>, <link type>, <DSET name>, <DTYP name>)
device(ai, GPIB_IO, devAiTestGpib, "ag34410AGpib")
device(ao, GPIB_IO, devAoTestGpib, "ag34410AGpib")
```

devag34410AGpib.c

```
/* define all desired DSETs */
#define DSET_AI devAiTestGpib
#define DSET_AO devAoTestGpib

#include <stddef.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#include <alarm.h>
#include <errlog.h>
#include <cvtTable.h>
#include <dbDefs.h>
#include <dbAccess.h>
#include <devSup.h>
#include <recSup.h>
#include <drvSup.h>
#include <link.h>
#include <dbCommon.h>
#include <aiRecord.h>
#include <aoRecord.h>

#include <devCommonGpib.h>
#include <devGpib.h>  /* must be included after DSET defines */

#define TIMEOUT 1.0
#define TIMEWINDOW  2.0

static struct gpibCmd gpibCmds[] =
{
 /* Param 0 */
```

```
 {&DSET_AI, GPIBREAD, IB_Q_LOW, "READ?¥r¥n", "%lf¥n", 32, 32, 0, 0, 0, 0, 0, 0},
 /* Param 1 */
 {&DSET_AO, GPIBWRITE, IB_Q_LOW, "SENS:VOLT:DC:RANG %lf¥r¥n", 0, 32, 32, 0, 0,
0, 0, 0, 0},
 /* Param 2 */
 {&DSET_AI, GPIBREAD, IB_Q_LOW, "SENS:VOLT:DC:RANG?¥r¥n", "%lf¥n", 32, 32, 0, 0,
0, 0, 0, 0}
};

/* The following is the number of elements in the command array above. */
#define NUMPARAMS sizeof(gpibCmds)/sizeof(struct gpibCmd)

/*****************************************************************
* Initialization for device support
* This is called one time before any records are initialized with a parm
* value of 0. And then again AFTER all record-level init is complete
* with a param value of 1.
*****************************************************************/
static long init_ai(int parm)
{
  if(parm==0)  {
    devSupParms.name = "devTestGpib";
    devSupParms.gpibCmds = gpibCmds;
    devSupParms.numparams = NUMPARAMS;
    devSupParms.timeout = TIMEOUT;
    devSupParms.timeWindow = TIMEWINDOW;
    devSupParms.respond2Writes = -1;
  }
  return(0);
}
```

~/lecture/ag34410A/ag34410AApp/srcディレクトリにあるMakefileをエディタで開き、“ag34410A_DBD+=base.dbd”と記述されている行の次の行に、以下の命令を挿入してください。

```
ag34410A_SRCS += devag34410A.c
ag34410A_DBD += devag34410A.dbd
```

このとき、~/lecture/ag34410A/ag34410AApp/Dbディレクトリにあるdevag34410A.dbをエディタで開き、各レコードのDTYP・INP(OUT)フィールドを以下のように修正します。

```
record(ai, "ET_$(user):$(name):READ_VOLTAGE")
{
  ...
  field(DTYP, "ag34410AGpib")
  field(INP, "#$(port) A$(A) @0")  # @<number> is Param number
  ...
}

record(ao, "ET_$(user):$(name):SET_RANGE")
{
  ...
  field(DTYP, "ag34410AGpib")
  field(OUT, "#$(port) A$(A) @1")
  ...
}

record(ai, "ET_$(user):$(name):READ_RANGE")
{
  ...
  field(DTYP, "ag34410AGpib")
  field(INP, "#$(port) A$(A) @2")
  ...
}
```

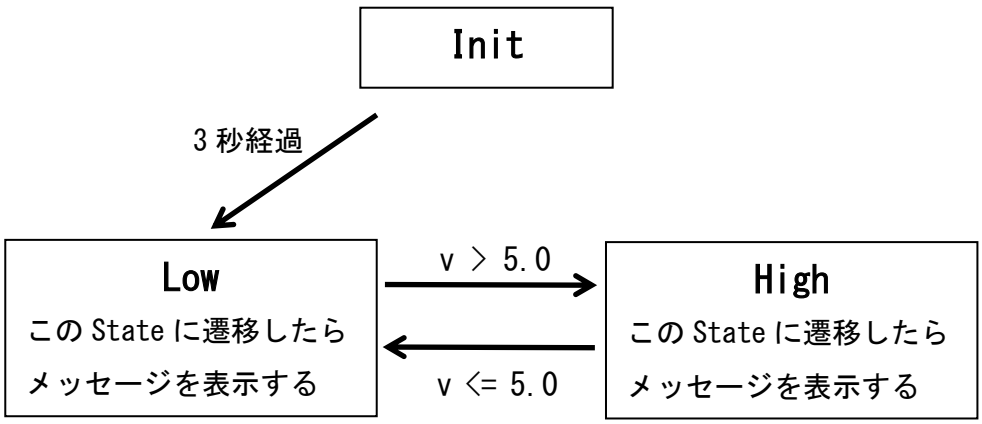
7. 実習 6: SNL プログラムの作成

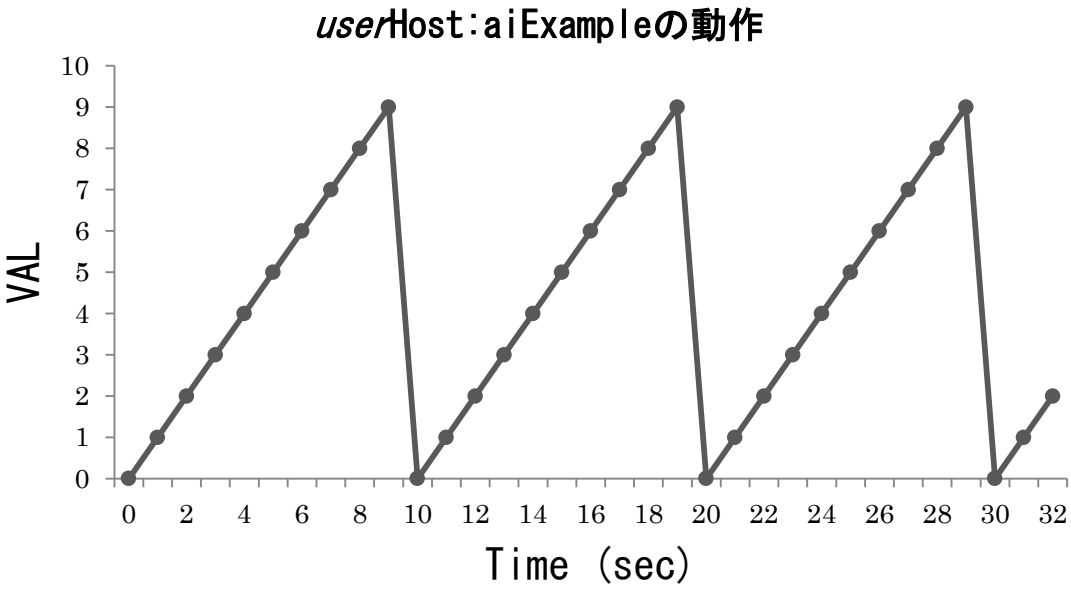
7.1. 目的

本実習では、SNL プログラム[7]の作成方法について学習します。

ここでは、実習 2 で作成した EPICS アプリケーションを使用します。

今回は、“*user*Host:aiExample”というレコードをモニタし、5.0 以下及び 5.0 を超えたときにメッセージを表示するプログラムを作成します。





7.2. 演習

7.2.1. SNL の環境変数を設定

~/lecture/myexample/configure ディレクトリにある RELEASE をエディタで開き、“#SNCSEQ=...”と記述されている行を、以下の内容に置き換えてください。

```
SNCSEQ=/opt/epics/R314.12.4/modules/seq-2.2.1
```

7.2.2. サンプルプログラムのコピー

本実習では、7.2.3 で説明する sncMyexample.stt というファイルを以下のディレクトリに用意しています。

```
/opt/sample/epics_3.14.12.4/myexample
```

7.2.3. SNL プログラムの作成

~/lecture/myexample/myexampleApp/srcディレクトリにsncMyexample.sttというファイルをコピーしてください。このファイルの中身は以下のようになります。

```
program sncMyexample   /* Program Name */

double v;
assign v to "{user}:aiExample";  /* v is value of {user}:aiExample */
monitor v;  /* camonitor */

ss ss1 {
  state init {
    when (delay(3)) {   /* wait for 3 sec */
      printf("sncMyexample: Startup delay over¥n");
    } state low
  }
  state low {
    entry {
      printf("sncMyexample: Enter to state low¥n");
    }
    when (v > 5.0) {
      printf("sncMyexample: Changing to high¥n");
    } state high
  }
  state high {
    entry {
     printf("sncMyexample: Enter to state high¥n");
```

```
    }
    when (v <= 5.0) {
      printf("sncMyexample: Changing to low¥n");
    } state low
  }
}
```

~/lecture/myexample/myexampleApp/srcディレクトリにsncMyexample.dbdというファイルを作成し、中身を以下のようにしてください。

```
registrar(sncMyexampleRegistrar)
```

~/lecture/myexample/myexampleApp/src ディレクトリにある Makefile をエディタで開き、“ifneq ($(SNCSEQ),)”と記述されている行の次の行に、以下の命令を挿入してください。

```
sncMyexample_SNCFLAGS += +r   # Compiler option
myexample_DBD += sncMyexample.dbd
myexampleSupport_SRCS += sncMyexample.stt
myexampleSupport_LIBS += seq pv
myexample_LIBS += seq pv
```

7.2.4. ビルドと実行

3.2.4 同様にアプリケーションをビルドし、IOC を起動してください。
このとき、~/lecture/myexample/iocBoot/iocmyexample ディレクトリにある st.cmd をエディタで開き、“#seq sncExample...”と記述されている行の次の行に、以下の命令を挿入してください。ただし、*user* は自身のユーザー名としてください。

```
seq sncMyexample, "user=userHost"
```

別の端末で“*user*Host:aiExample”を camonitor し、値の変化とともに SNL プログラムがどのように動作しているのかを確認してみましょう。

iocsh 上でいくつかのコマンド（例：seqShow、seqChanShow、seqStop 等）を試してみましょう。
以下のコマンドにより、SNL プログラムの状態を確認することができます。

```
> seqShow sncMyexample
```

8. 付録

ここでは、SuperKEKB における EPICS IOC の開発規則に関する情報を紹介します。

EPICS version

R3.14.12.3（compactSA 対応版）

EPICS base

/proj/epics/R314/R31412/base-3.14.12.3-CSA

EPICS module

/proj/epics/R314/R31412/modules/R314123-CSA

EPICS アプリケーションのベースディレクトリ

/cont/epics314/app/KEKB

- グループディレクトリ（例：BT、MG、RF、VA、CO 等）
  - 用途別サブディレクトリ（各グループのポリシーによる）

例）制御グループのディレクトリ（CO）に panda という名前の IOC を作成する手順は、

1. /cont/epics314/app/KEKB/CO/の下に panda というディレクトリを作成する
2. panda ディレクトリ内で makeBaseApp.pl を実行する