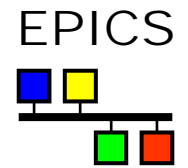


State Notation Language (SNL)

Ned D. Arnold
APS



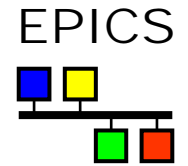
State Notation Compiler and Sequencer



- ◆ Allows programming of sequential state-oriented operations to run in the IOC
- ◆ The program interacts with the run-time database(s) via channel access
- ◆ Latest manual :
http://www.atdiv.lanl.gov/doc/epics/sequencer/sn1_1.9_man.html

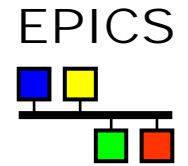


Uses



- ◆ State machines
- ◆ Startup sequences
- ◆ Enforce prudent operational procedures
- ◆ Watch for likely fault modes that are hard to detect via alarms
- ◆ Implement complex closed loop control schemes
- ◆ Coordinate control of multiple devices

Advantages

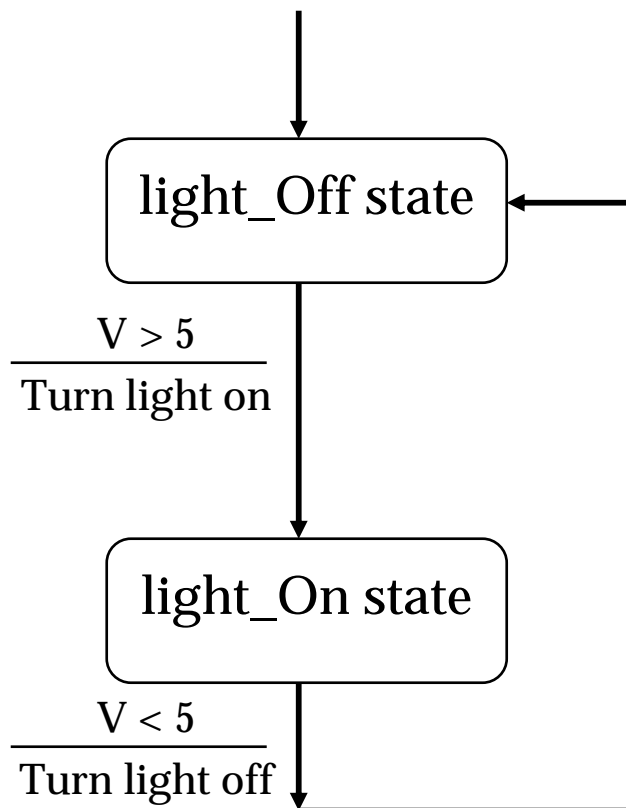


- ◆ Can implement complicated algorithms
- ◆ Can stop, reload, restart a sequence program without rebooting
- ◆ Interact with the operator through string records and mbbo records
- ◆ C code can be embedded as part of the sequence
- ◆ All Channel Access details are taken care of for you
- ◆ File access can be implemented as part of the sequence

Definitions

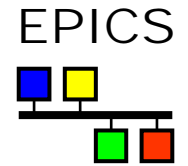
- ◆ *SNL* : State Notation Language
- ◆ *SNC* : State Notation Compiler
- ◆ *sequencer* : The tool within the IOC that executes the compiled SNL code
- ◆ *Program* : A complete SNL application consisting of declarations and one or more state sets
- ◆ *State Set* : A set of states that make a complete finite state machine
- ◆ *State* : A particular mode of the state set in which it remains until one of its transition conditions is evaluated to be TRUE

- ◆ The SNL code structure follows a state transition diagram format



```
state light_off {  
    when (v > 5.0){  
        light = TRUE;  
        pvPut(light);  
    } state light_on  
}  
  
state light_on {  
    when (v < 5.0){  
        light = FALSE;  
        pvPut(light);  
    } state light_off  
}
```

Basics (cont ...)



- ◆ Each state has one or more `when` statements which specify which state to enter next if their condition is met
- ◆ Action statements are executed during the transition from one state to another
- ◆ Access to Process Variables via channel access is accomplished by simply assigning a PV to a sequence variable

A Complete State Program (with 2 state sets)

```
program level_check
```

```
float v;
assign v to "ts1:ai1";
monitor v;

short light;
assign light to "ts1:bo1";

float vout;
float delta;
assign vout to "ts1:ai1";
```

```
ss volt_check {

state light_off {
  when (v > 5.0) {
    /* turn light on */
    light = TRUE;
    pvPut(light);
  } state light_on
}

state light_on {
  when (v < 5.0) {
    /* turn light off */
    light = FALSE;
    pvPut(light);
  } state light_off
}

}
```

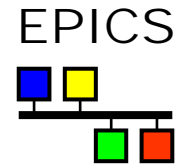
```
ss generate_voltage {

state init {
  when ( ) {
    vout = 0.0;
    pvPut(vout);
    delta = 0.2;
  } state ramp
}

state ramp {
  when (delay(0.1) {
    if ((delta > 0.0 && vout >= 11.0) ||
        (delta < 0.0 && vout <= -11.0) )
      delta = -delta; /* change direction */
    vout += delta;
  } state ramp;
}

}
```

More Basics ...



- ◆ A state can have multiple `when` statements. The first one to be evaluated to be `TRUE` will be executed. This allows conditional branching within the sequence program.
- ◆ When entering a state, all `when` conditions are evaluated in the order given in the source code
- ◆ If no `when` condition is true, the sequence program task pends until an event occurs, which causes all `when` conditions to be re-evaluated
- ◆ It is easy to create a loop and consume all available CPU time
 - ◆ When this happens, all CA clients connected to this IOC will disconnect

Example of “Multiple whens”

```
state checks
{
  when(interlockChasPwrBI==0)
  {
    sprintf(seqMsg1, "Electron Gun not ready ...");
    pvPut(seqMsg1);
    sprintf(seqMsg2, "Gun Interlock Chassis off ");
    pvPut(seqMsg2);
  } state initialChecks

  when(gunLocal)
  {
    sprintf(seqMsg1, "Electron Gun not ready ...");
    pvPut(seqMsg1);
    sprintf(seqMsg2, "Egun in local control ");
    pvPut(seqMsg2);
  } state initialChecks

  when(gunInterlocksRdyCC==0)
  {
    sprintf(seqMsg1, "Electron Gun not ready ...");
    pvPut(seqMsg1);
    sprintf(seqMsg2, "Interlocks not OK ");
    pvPut(seqMsg2);
  } state initialChecks
}
```

```
when(interlockChasPwrBI &&
      (gunLocal==0) && gunInterlocksRdyCC)
{
  gunAutoStart = 0;
  pvPut(gunAutoStart);
  gunAutoStop = 0;
  pvPut(gunAutoStop);
  sprintf(seqMsg1, "Push Auto-Start to begin autostart
...");
  pvPut(seqMsg1);
  sprintf(seqMsg2, "Push Auto-Stop to begin autostop ...");
  pvPut(seqMsg2);
%% taskDelay(60);
} state waitForRequest

state initialChecks
{
  when(delay(2.0))
  {
    sprintf(seqMsg1, "Initial Checks");
    pvPut(seqMsg1);
    sprintf(seqMsg2, "");
    pvPut(seqMsg2);
%% taskDelay(60);
} state checks
}
```

Other Features

- ◆ Assignment of macros at program startup for multiple copies of same sequence (must specify *+r* compiler flag)

```
program level_check ("unit=ts1")  
  
float v;  
assign v to "{unit}:ai1";  
  
short light;  
assign light to "{unit}:bo1";
```

In startup script ...

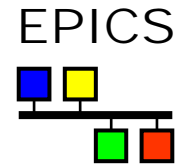
```
ld < level_check.o
```

```
seq &level_check, "unit=ts1"
```

```
seq &level_check, "unit=ts2"
```

- ◆ Arrays (each element can be assigned to a PV)
- ◆ Built-In functions (pg 17 of SNL Manual)
- ◆ Dynamic assignment of PVs to variables
- ◆ Connection Management and status

Other Features



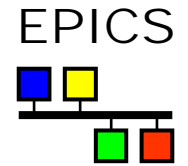
- ◆ Event Flags (used to sync state sets and monitors)
 - ◆ `efSet(name)`, `efTest(name)`, `efClear(name)`, `efTestAndClear(name)`
- ◆ Escape to C code
 - ◆ `%%` escapes a single line
 - ◆ `%{ }%` escapes a block of code
- ◆ Log errors to a log file

Debugging

◆ seqShow

```
ioc> seqShow
Program Name  Task ID  Task Name  SS Name
xx_RF_Cond   10854616 xx_RF_Cond  11AutoConditioning
bpmTraject   10838832 bpmTrajectory  bpmTrajectorySS
xx_autoPha   10680172 xx_autoPhasing  autoPhasing
              10573424 xx_autoPha_1  updatePresets
xx_autoRfT   10589876 xx_autoRfTiming  autoRfTiming
value = 0 = 0x0
```

Debugging



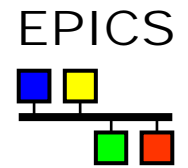
◆ seqShow <taskId>

```
ioc> seqShow 10838832
State Program: "bpmTraject"
initial task id=10838832=0xa56330
task priority=100
number of state sets=1
number of channels=56
number of channels assigned=56
number of channels connected=56
options: async=0, debug=0, newef=0, reent=0, conn=0
log file fd=3
log file name="/tyCo/0"

State Set: "bpmTrajectorySS"
task name=bpmTrajectory; task id=10838832=0xa56330
First state = "init"
Current state = "waitToPlot"
Previous state = "plotWithBeam"
    Elapsed time since state was entered = 0.4 seconds)

value = 0 = 0x0
```

Debugging

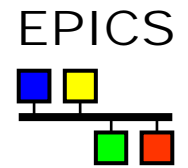


◆ seqChanShow <taskId>

```
ioc> seqChanShow 10838832
State Program: "bpmTraject"
Number of channels=56

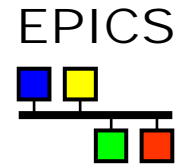
#1 of 56:
Channel name: "L1:PG1:PM1:BPM.XPOS"
  Unexpanded (assigned) name: "L1:PG1:PM1:BPM.XPOS"
  Variable name: "L1PG1PM1_X"
    address = 11931404 = 0xb60f0c
    type = float
    count = 1
  Value = 0
  Monitor flag=1
    Monitored
  Assigned
  Connected
  Get not completed or no get issued
  Status=11
  Severity=2
  Time stamp = 05/21/99 16:43:35.085407596
  Next? (+/- skip count)
```

Debugging



- ◆ `printf("Here I am in state xyz \n");`
- ◆ `sprintf(seqMsg1, "Here I am in state xyz");`
`pvPut(seqMsg1);`
- ◆ Reload and restart
 - ◆ `seqShow`
 - ◆ `td xxxxxx`
 - ◆ `ld < my_sequence_program.o`
 - ◆ `seq &my_sequence_program.o`

Examples



- ◆ BaBar <-> PEP injection handshake
- ◆ PEP Injection control
- ◆ RF startup procedures
- ◆ NLCTA fast trip processing
- ◆ PEP bucket-wise luminosity
- ◆ LLRF trip monitoring