Experimental Physics and Industrial Control System (EPICS)

Sequencer and State Notation Language

Tutorial Slides

Bob Dalesio & Andy Kozubal

August 11, 1999

Outline

- What is state notation and what function does it serve
- Components of a state notation program
- Building, running and debugging a state notation program
- Additional Features
- Some Notes on the Runtime Sequencer

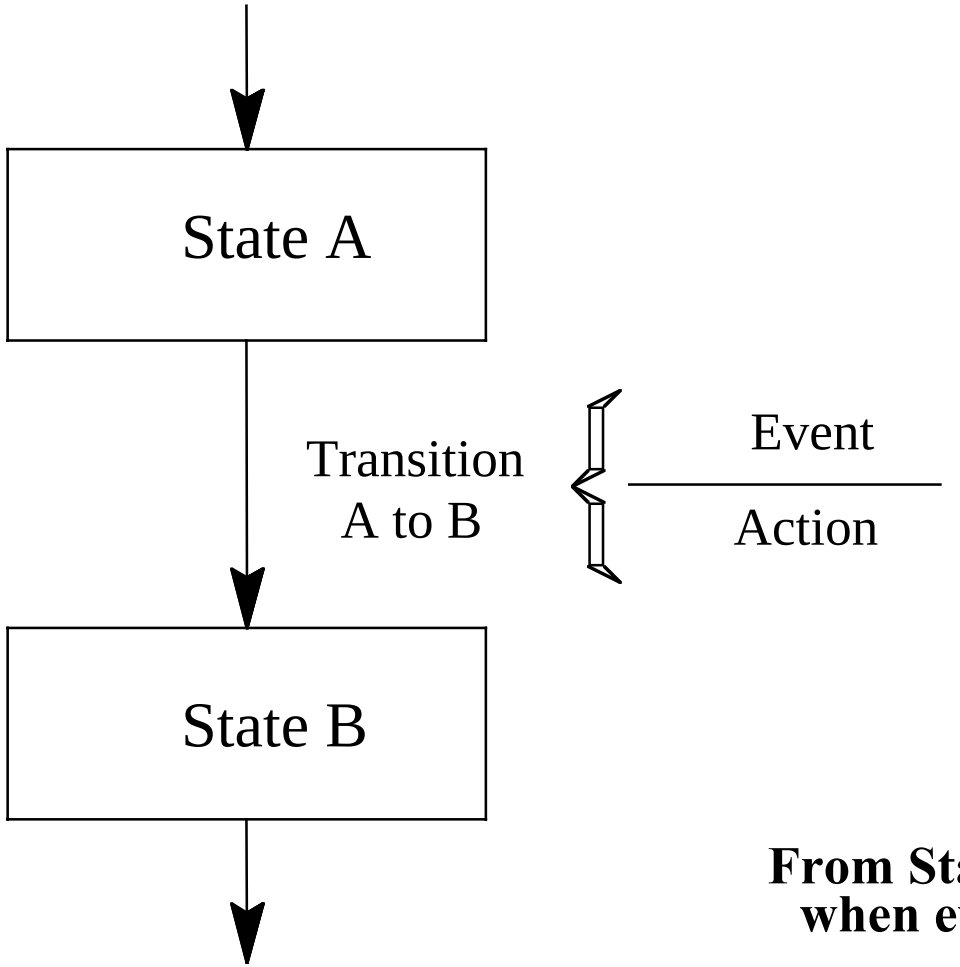
Purpose

- A language to facilitate sequential programming in the EPICS real-time environment
- Fast execution - compiled code
- Programming interface to extend EPICS in the real-time environment
- Easy for first-time user to learn and apply

Common Uses of the State Notation Language

- Provide automated start-up sequences like vacuum or RF where subsystems need coordination
- Provide fault recovery or transition to a safe state
- Provide access to the Unix file system for save/restore or restoration of parameters on reboot
- Provide automatic calibration of equipment

State Transition Diagram

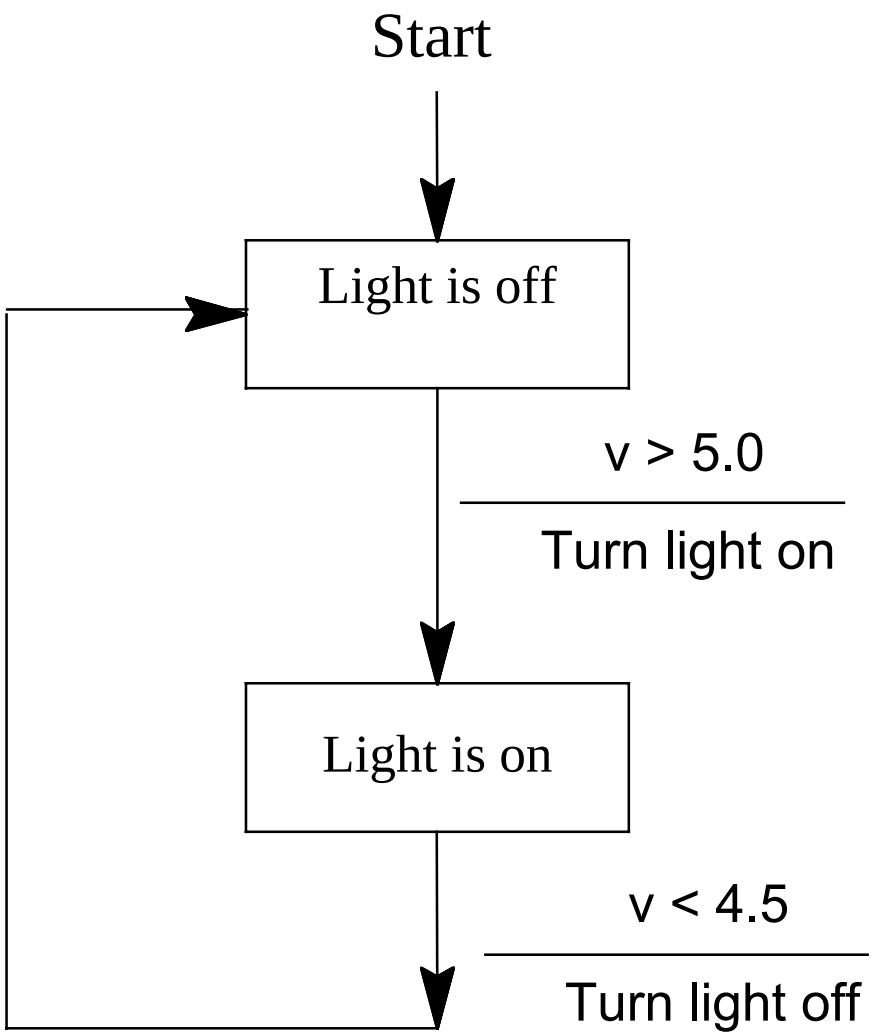


From State A:
when event occurs

perform action

then enter state B

Example of a STD



State Definitions and State Transitions

```
state light_off
{
        when (v > 5.0)
        {
                /*
                * turn light on
                */
        } state light_on
}

state light_on
{
        when (v < 4.5)
        {
                /*
                * turn light off
                */
        } state light_off
}
```

Channel Access in SNL

Variables assigned to a channel

```
short       light;
assign      light to "PS1:Lt_High";

            /* turn light on */
            light = TRUE;
            pvPut(light);

            /* turn light off */
            light = FALSE;
            pvPut(light);
```

Asynchronous Channel Access

The "monitor" concept

```
float       V1;
assign      V1 to "PS1:Vout;

monitor     V1;

                when (V1 > 5.0) { . . .
```

A Complete Program

```
program example
float                   V1;
assign                  V1 to "PS1:Vout;
monitor     V1;
short                   light;
assign                  light to "PS1:Lt_High";
ss volt_check
{
        state light_off
        {
                when (V1 > 5.0)
                {
                        /* turn light on */

                        light = TRUE;
                        pvPut(light);
                } state light_on
        }

        state light_on
        {
                when (V1 < 4.5)
                {
                        /* turn light off */
                        light = FALSE;
                        pvPut(light);
                } state light_off
        }
}
```

SNL: General Structure and Syntax

```
program program_name
declarations
ss state_set_name
{
        state state_name
        {
                when (event)
                {
                        action_statements
                } state new_state
                when...
        state state_name
        {
                ....
        }
```

Declarations

```
int       variable_name;
short     variable_name;
long      variable_name;
char      variable_name;
float     variable_name;
double    variable_name;
string    variable_name; /* this is used for db strings */

int       variable_name[array_length];
short     variable_name[array_length];
long      variable_name[array_length];
char      variable_name[array_length];
float     variable_name[array_length];
double    variable_name[array_length];

assign  var_name to database_name;
*assign           var_name[] = {db_name1, db_name2, db_name3};
monitor var_name;
```

Events

when (*any C expression*)

Possible events:

Change in value of a variable

A time delay:

delay(*delay_in_seconds*)

An internally generated event (event flag):

efTest(*event_flag_name*)

efTestAndClear(*event_flag_name*)

Change in the channel access connection status.

Actions

any C expression

%% escape one line of C code

%{
escape any number of lines of C code
}%

Built-in action function:

pvPut (variable_name);
pvGet (variable_name);
efSet (event_flag_name);
efClear (event_flag_name);

Event flags

1. Declaration:

```
evflag  event_flag_name;
```

2. Communicate between state sets:

```
efSet(event_flag_name);
efTest(event_flag_name)
efClear(event_flag_name)
efTestAndClear(event_flag_name)
```

3. Notification when a monitor completes:

```
sync    variable_name  event_flag_name;

    efTest(event_flag_name);
```

Building a state program

1. Use editor to build the source file: file name must end with ".st", e.g. "example.st".

2. Compile the state program to produce C code:

   snc example.st

   This produces the file "example.c".

3. Compile "example.c" with the C compiler:

   make example.o

This produces the file "example.o", which is ready to be loaded by VxWorks.

The C preprocessor is generally used before snc is executed.

An applications Makefile simplifies these steps.

Executing a state program

Assume that VxWorks is running in an IOC and the proper database is loaded. Also, assume the UNIX working directory is the same as where to IOC was booted.

1. Telnet to the IOC:

   ```
   telnet  ts1
   log in
   ts1>            you should get a prompt
   ```

2. Load the object module:

   ```
   ts1> ld < example.o
   ```

3. Execute the state program:

   ```
   ts1> seq  &example   this is the program name
   ```

   This will create one task for each state set.

4. Exercise the program.

5. Print a summary of state programs

   ```
   ts1> seqShow
   ```

6. Delete any one of the tasks that were created in step 3.

   ```
   ts1> td "example"
   ```

Hints for debugging a state program

Currently there is no source-level debugger for sequencer.

1. Use printf statements in program:
   printf("entering state: light_on");
2. Manually enter database values using CAU:
   cau: put hv03:temp1 150
3. Print database values using CAU:
   cau: get hv03:temp1
   150.00
4. Use special state program query commands:
   seqShow
   displays information on all running state programs
   seqShow "example"
   displays detailed information on program
   seqChanShow "example"
   displays information on all channels
5. Use spy to find infinite loops with no delays

Example of seqShow output

```
seqTest> seqShow
Program Name     Task ID    Task Name       SS Name

mixTest            26         mixTest         generate_waveform
                   29         mixTest_1         level_det
                   30         mixTest_2         test_status
                   31         mixTest_3         periodic_read
                   32         mixTest_4         mon_array

mixTestX           33          A_seq            alpha

conTest            35         conTest          conTest

value = 0 = 0x0
```

Example of seqShow Output

```
seqTest> seqShow "mixTest"
State Program: "mixTest"
initial task id=26=0x1a

task priority=1
        number of state sets=5
        number of channels=6
        number of channels connected=6
        options: async=0, debug=1, reent=1, conn=1, newef=1
        log file fd=16
        log file name="mix.log"

Hit RETURN to continue
        State Set: "generate_waveform"
        task name=mixTest;    id=26=0x1a
        First state = "start"
        Current state = "gen_wf"
        Previous state = "gen_wf"
        Time state was entered = 269.4 seconds)
        Elapsed time since state was entered = 0.0 seconds)
        Number delays queued=1
        timeout[0] = 269.5 seconds
Hit RETURN to continue
.........................
```

Additional Features

1. Connection management:
   when (pvConnectCount() != pvChannelCount())
   when (pvConnected(Vin))
2. Macros:
   assign Vout to "{unit}:OutputV";
   (must use the +r compiler options for this)
   ts1> seq &example, "unit=HV01"
3. Compiler options:
   +r make program reentrant
   -c don't wait for all channel connections
   +a asynchronous pvGet()
   -w don't print compiler warnings
4. Pass parameters to programs at run time:
   pStr = macValueGet("bias");
   ts1> seq &example, "bias = 2.55"
5. Access to alarm status and severity:
   pvStatus(var_name)
   pvSeverity(var_name)

The Run-Time Sequencer

1. The sequencer executes the state program in the VxWorks environment.
2. The sequencer supports the event-driven execution; no polling needed.
3. Each state set becomes a VxWorks task.
4. The sequencer manages connections to database channels through "channel access".
5. The sequencer provides support for channel access (put, get, and monitor).
6. The sequencer supports asynchronous execution of delay and event flag functions.
7. Only one copy (object module) of the sequencer is required on each IOC.
8. Query commands display information about executing state programs.