SNL and the Sequencer

- The sequencer runs programs written in State Notation Language (SNL)
- SNL is a ‘C’ like language to facilitate programming of sequential operations
- Fast execution - compiled code
- Programming interface to extend EPICS in the real-time environment
- Common uses
  - Provide automated start-up sequences like vacuum or RF where subsystems need coordination
  - Provide fault recovery or transition to a safe state
  - Provide automatic calibration of equipment

Outline

- What is State Notation Language (SNL)
- Where does it fit in the EPICS toolkit
- Components of a state notation program
- Some Notes on the Runtime Sequencer
- Building, running and debugging a state notation program
- Additional Features
- When to use it
- This talk covers Sequencer version 2.0.8
- This talk does not cover all the features of SNL and the sequencer. Consult the manual for more information.

http://www.slac.stanford.edu/comp/unix/package/epics/sequencer/
Where's the Sequencer Now?

The Best Place for the Sequencer

- From Sequencer Version 2.0.0 can be either in the IOC or on a workstation
- Traditionally in the IOC
- Locating it in the IOC probably makes it easier to manage
- Running on workstation could make testing easier
- Workstation gives an easy way to write the CA parts of CA clients

SNL implements State Transition Diagrams

STD Example

Start

Low vacuum

pressure > 5.1 uTorr
Open the valve

High vacuum

pressure < 4.9 uTorr
Close the valve
Some Definitions

- **SNL**: State Notation Language
- **SNC**: State Notation Compiler
- **sequencer**: The tool 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

SNL: General Structure and Syntax

```plaintext
program program_name
declarations
ss state_set_name {
  state state_name{
    entry{
      action statements
    }
    when (event){
      action statements
    state state_new_state_name
    when(event)
      exit{
        action statements
      }
    }
    state state_name{
      ...}
}
```

Declarations

- Occur before a state set and have a scope of the entire program.
  - **Scalar types**
    - int variableName;
    - short variableSname;
    - long variableLname;
    - char variableCname;
    - float variableFname;
    - double variableDname;

  - **Vector types**
    - int arrayName[array_length];
    - short arraySname[array_length];
    - long arrayLname[array_length];
    - char arrayCname[array_length];
    - float arrayFname[array_length];
    - double arrayDname[array_length];
Declarations - Assignments

- Assignment to channel access server channels
  
  ```
  float pressure;
  assign pressure to "CouplerPressureRB1";
  
  double pressures[2];
  assign pressures to ("CouplerPressureRB1", "CouplerPressureRB2", "CouplerPressureRB3");
  ```

- To use these channel in when clauses, they must be monitored
  ```
  monitor pressure;
  monitor pressures;
  ```

- Can be written like this to aid readability
  ```
  float pressure; assign pressure to "PressureRB1"; monitor pressure;
  ```

Declarations – Event Flags

Declaring Event Flags

Event for state sets to set, clear and test

```

evflag event_flag_name;

Flag monitor is set when PV changes (posts a monitor)

evflag flag_monitor;
sync pressure flag_monitor;
```

Events

An event is the condition on which statements following a when are executed and a state transition is made

Possible events:

- Change in value of a variable that is being monitored
  ```
  example: when(achan < 10.0)
  ```

- A timed event (not a task delay!)
  ```
  example: when(delay(1.5))
  ```
  The delay value is in seconds. It is declared internally as a double and constant arguments to the delay function must contain a decimal point.
  A delay is normally reset whenever the state containing it is exited. Use the state specific option -t to keep it from being reset when exiting to the same state..

Events (continued)

- An internally generated event (event flag)
  ```
  examples:
  when(efTestAndClear(myflag))
  when(efTest(myflag))
  ```
  efTest does not clear the flag. efClear must be called sometime later to avoid an infinite loop.
  The event flag can be set internally by efSet(event_flag_name) or if the flag is synced to a monitored channel it will be set when the channel changes.

- Change in the channel access connection status.
  ```
  examples:
  when(pvConnectCount() < pvChannelCount())
  when(pvConnected(mychan))
  ```
Actions

- Built-in action function, e.g.:
  - `pvPut (variable_name);`
  - `pvGet (variable_name);`
  - `efSet (event_flag_name);`
  - `efClear (event_flag_name);`
- Almost any C expression
  
  switch is not implemented and code using it must be escaped.

  ```
  // escape one line of C code
  \%
  // escape any number of lines of C code
  \%
  ```

Example - State Definitions and Transitions

Example – State Transitions

```c
program vacuum_control
    ss coupler_control
    {
        state init{
            when (pressure > .0000051) {
                state low_vacuum
            }
            when (pressure <= .0000049) {
                state high_vacuum
            }
        }
        state high_vacuum{
            when (pressure > .0000051) {
                state low_vacuum
            }
            when (pressure <= .0000049) {
                state high_vacuum
            }
            when (delay(600.0)) {
                state fault
            }
        }
        state low_vacuum{
            when (pressure <= .0000049) {
                state high_vacuum
            }
            when (delay(600.0)) {
                state fault
            }
        }
    }
```

Example - Declarations

```c
double pressure;
assign pressure to "Tank1Coupler1PressureRB";
monitor pressure;
short RoughPump;
assign RoughPump to "Tank1Coupler1RoughPump";
short CryoPump;
assign CryoPump to "Tank1Coupler1CryoPump";
short Valve;
assign Valve to "Tank1Coupler1IsolationValve";
String CurrentState;
assign CurrentState to "Tank1Coupler1VacuumState";
```
Example – Init State

```c
state init{
  entry{
    strcpy(CurrentState,"Init");
    pvPut(CurrentState);
  }
  when(pressure > .0000051){
    RoughPump = 1;
    pvPut(RoughPump);
    CryoPump = 0;
    pvPut(CryoPump);
    Valve = 0;
    pvPut(Valve);
    state low_vacuum
  }
  when(pressure <= .0000051){
    RoughPump = 0;
    pvPut(RoughPump);
    CryoPump = 0;
    pvPut(CryoPump);
    Valve = 0;
    pvPut(Valve);
  }
}
```

Example – State low_vacuum

```c
state low_vacuum{
  entry{
    strcpy(CurrentState,"Low Vacuum");
    pvPut(CurrentState);
  }
  when(pressure <= .0000049){
    RoughPump = 0;
    pvPut(RoughPump);
    CryoPump = 1;
    pvPut(CryoPump);
    Valve = 1;
    pvPut(Valve);
    state high_vacuum
  }
  when(pressure < .0000049){
    RoughPump = 0;
    pvPut(RoughPump);
    CryoPump = 1;
    pvPut(CryoPump);
    Valve = 1;
    pvPut(Valve);
    state high_vacuum
  }
  when(delay(600.0)){
    state fault
  }
}
```

Example – State high_vacuum

```c
state high_vacuum{
  entry{
    strcpy(CurrentState,"High Vacuum");
    pvPut(CurrentState);
  }
  when(pressure > .0000051){
    RoughPump = 1;
    pvPut(RoughPump);
    CryoPump = 0;
    pvPut(CryoPump);
    Valve = 0;
    pvPut(Valve);
    state low_vacuum
  }
}
```

Example – State fault

```c
state fault{
  entry{
  strcpy(CurrentState,"Vacuum Fault");
  pvPut(CurrentState);
  }
}
```
Building an SNL program

- Use editor to build the source file: file name must end with ".st", e.g. "example.st".
- "make" automates these steps:
  - Optionally runs the C preprocessor
  - Compiles the state program with SNC to produce C code:
    
  
  \[
  \text{snc example.st -> example.c}
  \]
  - Compiles the resultant C code with the C compiler:
    
  
  \[
  \text{cc example.c -> example.o}
  \]
  - The file "example.o" becomes part of the application library, which is ready to be loaded by VxWorks.
  - For Unix systems an executable file "example" is created

Run Time Sequencer

- The sequencer executes the state program.
- The sequencer supports the event-driven execution; no polling needed.
- Each state set becomes a VxWorks task or UNIX thread.
- The sequencer manages connections to database channels through "channel access".
- The sequencer provides support for channel access (put, get, and monitor).
- The sequencer supports asynchronous execution of delay, event flag, pv put and pv get functions.
- Only one copy (object module) of the sequencer is required on an IOC.
- Query commands display information about executing state programs.

Executing a State Program – IOC

Assumes you are at an IOC console and database is loaded
1. Load the sequencer
   \[
   \text{ld < pvLibrary}
   \]
   \[
   \text{ld < sequencer}
   \]
2. Load a state program
   \[
   \text{ld < example.o}
   \]
3. Execute program
   \[
   \text{seq &vacuum_control}
   \]
4. Er... That's it! Exercise program
5. To stop program
   \[
   \text{seqStop vacuum_control}
   \]

Debugging

- Use special state program query commands:
  - \text{seqShow}
    \[
    \text{displays information on all running state programs}
    \]
  - \text{seqShow vacuum_control}
    \[
    \text{displays detailed information on program}
    \]
  - \text{seqChanShow vacuum_control}
    \[
    \text{displays information on all channels}
    \]
  - \text{seqChanShow vacuum_control, -}
    \[
    \text{displays information on all disconnected channels}
    \]
**Debugging (continued)**

- Use printf functions to print to the console
  ```c
  printf("Here I am in state xyz \n");
  ```
- Put strings to pvs
  ```c
  sprintf(seqMsg1, "Here I am in state xyz");
  pvPut(seqMsg1);
  ```
- Reload and restart
  ```c
  seqStop vacuum_control
  Edit
  ld < example.o
  seqStart &vacuum_control
  ```

---

**Debugging - seqShow**

- **seqShow**

  ```c
  epics> seqShow
  Program Name     Thread ID  Thread Name      SS Name
  stabilizer       ede78      stabilizer       stabilizerSS1
  beamTrajectory db360      beamTrajectory bpmTrajectorySS
  autoControl ed620      autoControl autoCtlSS
  ```

---

**Debugging - seqChanShow**

- **seqChanShow stabilizer**

  ```c
  epics> seqChanShow stabilizer
  Number of channels=3
  #1 of 3:
  Channel name: "jfm:OP:stabilizerC"
  Unexpanded (assigned) name: "{user}:OP:stabilizerC"
  Variable name: "enableButton"
  address = 154120 = 0x25a08
  type = short
  count = 1
  Value = 0
  Monitor flag = 1
  Assigned
  Connected
  Get not completed or no get issued
  Put not completed or no put issued
  Status = 17
  Severity = 3
  Message =
  Time stamp = <undefined>
  Next? ( skip count)
  ```
### Additional Features

- **Connection management:**
  - when ( pvConnectCount() != pvChannelCount() )
  - when ( pvConnected(Vin) )

- **Macros:**
  - assign Vout to "{unit}:OutputV";
  - (must use the +r compiler options for this if more than one copy of the sequence is running on the same ioc)
  - seq &example, "unit=RV01"

- **Compiler options:**
  - +r make program reentrant (default is -r)
  - -c don't wait for all channel connections (default is +c)
  - +a asynchronous pvGet() (default is -a)
  - -w don't print compiler warnings (default is +w)
  - +e eftest automatically clears flag (default is -e)

### Additional Features (continued)

- **Access to alarm status and severity:**
  - pvStatus(var_name)
  - pvSeverity(var_name)

- **Queueable monitors -- saves monitors in queue in the order they come in -- no missing monitors.**
  - syncQ variableName to eventFlagname [optionally the length of the queue]
  - pvGetQ( variableName )
  - true until queue is empty.
  - pvFreeQ( variable Name )

### 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

### When to use the sequencer

- **For sequencing complex events**
- E.g. Parking and unparking a telescope mirror

![Photograph courtesy of the Gemini Telescopes project](Image)
Should I Use the Sequencer?

START

CAN I DO THIS IN A DB?

V

CAN I DO THIS IN A DB?

V

USE THE SEQUENCER

USE A DATABASE

END

Acknowledgements

- Slides for this presentation have been taken from talks prepared by the following people
  - Bob Dalesio (LANL/SNS/LCLS)
  - Deb Kerstiens (LANL)
  - Rozelle Wright (LANL)
  - Ned Arnold (APS-Controls)