OVERVIEW

While all standard debugging techniques work (printf, reading through code etc), there are two specific functions within ES_FRAMEWORK, that will significantly aid in debugging: tattletale and ES_KEYBOARD_INPUT. These will allow you to determine the state of your state machine as well as virtually debug. Use them well and often within the boot loader. This document outlines how to translate and set them up.

Tattletale

Within ES_FRAMEWORK we’ve added an event trace function for debugging. This allows you to see not only the state(s) you are in but the events that trigger the transitions. This will ease the debugging process significantly and can be used in conjunction with ES_Keyboard_Input to virtually run and debug your state machine.

When tattletale is enabled, it will first spit out the init trace as such (for hierarchical roach):

```
RunFancyRoachHSM[InitPState(ES_INIT,0)]-
>RunSubStoppedHSM[InitPSubState(ES_INIT,0)]->
RunSubStoppedHSM[InitPSubState(ES_EXIT,0)]-
>RunSubStoppedHSM[Halted(ES_ENTRY,0)]->
RunFancyRoachHSM[InitPState(ES_EXIT,0)]-
>RunFancyRoachHSM[Stopped(ES_ENTRY,0)];
```

RunFancyRoachHSM[InitPState(ES_INIT,0)] translates to:
```
state_machine_file[State(Event,EventParameter)]
```

The first line translates as such: upon run, the ES_Init event called the highest state machine (RunFancyRoachHSM) which starts in its initial state. From its initial state, it activates the first sub state machine (RunSubStoppedHSM).

The second line: RunSubStoppedHSM changes CurrentState from InitPSubState to Halted and exits the InitPSubState using an ES_EXIT and ES_ENTRY to get into Halted.

The third line: RunFancyRoachHSM changes CurrentState from InitPState to Stopped and exits the

1Courtesy of Prof. Carryer at Stanford University
InitPSubState using an ES_EXIT and ES_ENTRY to get into Halted.

While you might expect the third line to appear first (logically) substate machines actually transition before their supers.

Tattletale will output a trace at the end of every state machine run. For example, if you have an event (Such as WAS_DARK_NOW_LIGHT) while in the sub state “Halted”, the output would look like this:

RunFancyRoachHSM[Stopped(WAS_DARK_NOW_LIGHT,8)]->
RunSubStoppedHSM[Halted(WAS_DARK_NOW_LIGHT,8)]->
RunFancyRoachHSM[Stopped(ES_EXIT,0)]->RunSubStoppedHSM[Halted(ES_EXIT,0)]->
RunSubStoppedHSM[InitPSubState(ES_INIT,0)]->
RunSubStoppedHSM[Halted(ES_EXIT,0)]->
RunFancyRoachHSM[Fleeing(ES_ENTRY,0)]-> RunSubRunHSM[Waiting(ES_ENTRY,0)];

In this trace it has passed the trace down to the RunSubStoppedHSM, which has generated an exit event. This in turn, resets the sub state machine (so its ready for its next go) and transitions RunFancyRoachHSM to its next state, Fleeing. Fleeing in turn initiates its first substate machine RunSubRunHSM which initiates its first state Waiting.

While ES_ENTRY and ES_EXIT events lead to these traces being somewhat verbose they will help you quickly find errors in your state machines transitions and events. You can turn them off by uncommenting `#define SUPPRESS_EXIT_ENTRY_IN_TATTLE` in `ES_CONFIGURE.h`
Using Tattletale

Follow the steps below to enable tattletale.

1. You need to generate the string forms of the events names for tattletale to work. For example if your events enum is as follows (inside ES_CONFIGURE.h):

   ```
   typedef enum {
       ES_INIT, /* used to transition from initial pseudo-state */
       ES_ENTRY,
       ES_EXIT, /* used to exit a state*/
   } ES_EventTyp_t;
   ```

   You need to generate a char array that have names that match EXACTLY the events. Place this directly below your typedef enum as so:

   ```
   static const char *EventNames[] ={
       "ES_INIT",
       "ES_ENTRY",
       "ES_EXIT"};
   ```

   You can save yourself a few minutes and typos by copying your events enum, clicking the generate string array from enum icon on the desktop and then pasting it back into the ES_CONFIGURE.h.

   You will also need to write the number of events defined as so:
   ```
   #define NUMBEROFEVENTS (3)
   ```

2. You now need to generate the string forms of ALL of the states. For example if your states enum is as follows (inside RunFSM.h):

   ```
   typedef enum { InitPState, Running, Turning} RunState_t ;
   ```

   NOTE: Make sure all your state enums end with t. General convention is to name them "nameOfStateState_t". Also make sure that each has the first state as InitPState for the finite state machines. For the hierachal state machines you need to have InitPSubState as the first state (it should match the initial assignment for CurrentState).

   Just like the events, generate a char array that have names that match EXACTLY the states inside the .c file (RunFSM.c). You can use the generate string array from enum for this too.

   ```
   static const char *StateNames[] ={
       "InitPState",
       "Running",
       "Turning"};
   ```

   You do not need to write the number of events but you do need to ensure you have done this for EVERY STATE.

3. Finally uncomment #define USE_TATTLETALE inside of ES_CONFIGURE.h.
ES_KEYBOARD_INPUT

ES_KEYBOARD_INPUT allows you to artificially post events and parameters to your bot. It can be used in conjunction with tattleTale or alone. This is an excellent way to walk through your state machines and ensure it works the way you think it does.

- **Enabling Keyboard input:**
  - Follow the instructions in tattletale to setup the events correctly.
  - Uncomment `#define USE_KEYBOARD_INPUT` inside of `ES_CONFIGURE.h`.

- **Using keyboard Input:**
  - When you initialize your program with activated `KEYBOARD_INPUT`, you should see this:

    Printing all events available in the system

    0: ES_NO_EVENT
    1: ES_ERROR    2: ES_INIT    3: ES_ENTRY
    4: ES_EXIT    5: ES_KEYINPUT    6: ES_LISTEVENTS
    7: ES_TIMEOUT    8: ES_TIMERACTIVE    9: ES_TIMERSTOPPED
    10: WAS_DARK_NOW_LIGHT    11: WAS_LIGHT_NOW_DARK    12: BUMPED
    13: DONE_EVADING

    Keyboard input is active, no other events except timer activations will be processed. You can redisplay the event list by sending a 6 event

This gives you a list of all of the events available to type in. Note that it will not process othe events from the roach. To create an event, you need to type (for example) “10->0;” into the input and press enter. This creates a WAS_DARK_NOW_LIGHT event and the hex parameter (in this case 0). If you wanted to see what happens when timer 8 stops, you would type “9->8;”.