

## MEMO

El objetivo de este documento es la explicación de una máquina de estados para Arduino, lo más sencilla posible.

La máquina de estados dispone de 3 elementos: ESTADO, FUNCIÓN y EVENTO.

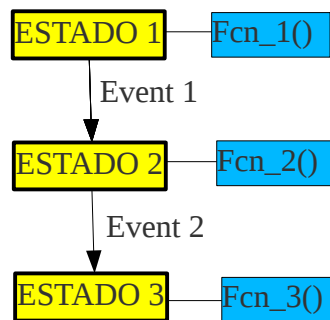
Dentro de los EVENTOS, dividiremos conceptualmente entre EXTERNOS e INTERNOS.

Los EXTERNOS son recibir dato mediante puerto serie, pulsación de un botón, entrada analógica, etc.

Los INTERNOS son generados/lanzados dentro de la máquina de estados.

El funcionamiento constante de la FSM es el siguiente:

1. Leo EVENTOS externos.
2. Actualizo maquina de estados.
3. Ejecuto función correspondiente a dicho estado.
4. Actualizo la maquina de estados mientras exista EVENTOS internos que se hayan generado.



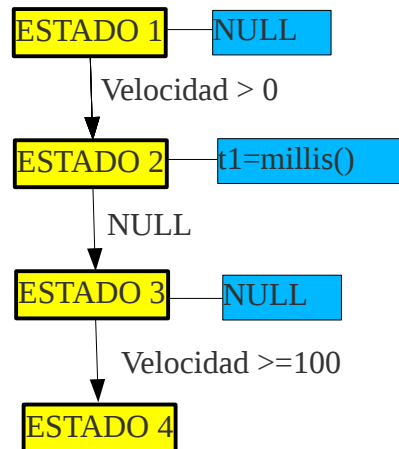
El funcionamiento de este diagrama es sencillo:

- Si estamos en ESTADO1 y se recibe EVENT1, entonces pasamos al ESTADO2.
- Si estamos en ESTADO2 y se recibe EVENT2, entonces pasamos al ESTADO3.
- Mientras estamos en cada uno de los estados, se ejecuta la función asociada. Es decir, mientras estamos en ESTADO1, se ejecuta Fcn\_1().

Se dispone de un EVENTO y una FUNCIÓN especial, llamada NULL (0=cero). Por ejemplo, usaremos la función NULL cuando no quereamos realizar una función en el estado actual. Por otro lado, tendremos el evento NULL para pasar de un estado a otro automáticamente.

**NO SE DEBE DEFINIR NINGÚN ESTADO COMO CERO, YA QUE SE USA NULL (0) PARA REPORTAR ERROR.**

¿para qué sirve ésto? Imaginemos que queremos capturar el tiempo cuando un evento concreto ocurre, como puede ser cuando la velocidad de nuestro coche es mayor que cero.



Podemos ver, que estando en nuestro estado inicial ESTADO1 (en el cual no realizamos ninguna acción), pasaremos al ESTADO2 si la velocidad es mayor que cero. En ese momento capturaremos en la variable t1 el tiempo actual, y automáticamente la siguiente vez que se actualice la máquina de estados, pasaremos al ESTADO 3 en el cual se quedará esperando sin hacer nada hasta que la velocidad sea mayor o igual que 100 km/h.

### Funciones de nuestra FSM:

- Función begin() → Tiene como entradas las estructuras descriptivas del digrama de flujo, sus tamaños y el estado inicial.
- Función State() → Devuelve el estado actual de la maquina de estados.
- Función Update() → Actualiza la maquina de estados. Mientras se esten generando EVENTOS internos, se continua actualizando la máquina de estados.
- Función AddEvent() → Añade el EVENTO actual.

La construcción descriptiva de nuestra FSM se realiza mediante dos estructuras:

- FSM\_State → Tabla que contiene en cada fila el nombre del estado y la función asociada.
- FSM\_NextState → Tabla que contiene Estado actual, Evento y Siguiente estado.

```

const FSM_State_t FSM_State[] PROGMEM= {
// STATE STATE_FUNC
{State1,Fcn1},
{State2, Fcn2},
};
  
```

*FSM\_State → Si estoy en State1, realizo Fcn1. Si estoy en State2, realizo Fcn2.*

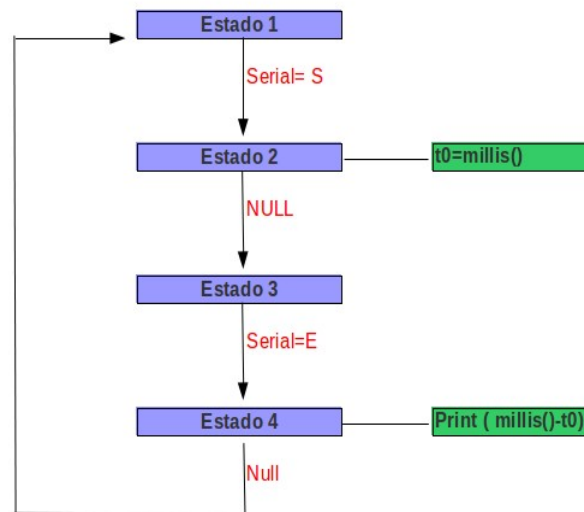
```

const FSM_NextState_t FSM_NextState[] PROGMEM= {
// STATE INPUT NEXT_STATE
{State1,Event1,State2},
{State2,Event2,State1},
};
  
```

*FSM\_NextState → Si estoy en State1 y recibo Event1, saltaré a State2. Si estando en State2, recibo Event2, saltaré a State1.*

Veámos un ejemplo particular, se quiere crear un programa que al recibir una S (start) por el puerto serie empieza a cronometrar y al recibir una E (end), termine dicho cronometraje y saque por puerto serie el resultado.

Para ello, creamos el digrama de flujo de funcionamiento del programa:



Ahora, nos creamos una tabla descriptiva de dicho diagrama:

ESTADO	EVENTO	NUEVO EVENTO
Estado1	Recibir_S	Estado2
Estado2	NULL	Estado3
Estado3	Recibir_E	Estado4
Estado4	NULL	Estado1

ESTADO	ACCIÓN
Estado1	NULL
Estado2	Recoger tiempo en t0
Estado3	NULL
Estado4	Sacar por serie tiempo cronometrado

Resulta directo crear las estructuras del programa que necesitamos gracias a dichas tablas:

```
const FSMClass::FSM_NextState_t FSM_NextState[] PROGMEM= {  
// STATE,EVENT,NEXT_STATE  
{STATE1,EV_S,STATE2},  
{STATE2,0,STATE3},  
{STATE3,EV_E,STATE4},  
{STATE4,0,STATE1},  
};
```

```
const FSMClass::FSM_State_t FSM_State[] PROGMEM= {  
// STATE,STATE_FUNC  
{STATE1,0},  
{STATE2,func2},  
{STATE3,0},  
{STATE4,func4},  
};
```

Estas estructuras son guardadas en memoria de programa para evitar consumo de memoria RAM innecesariamente.

Como se puede observar, se han creado unos identificadores para nuestros estados y eventos, de esta manera nos resulta muy sencillo escribir nuestro programa usando dichos nombres.

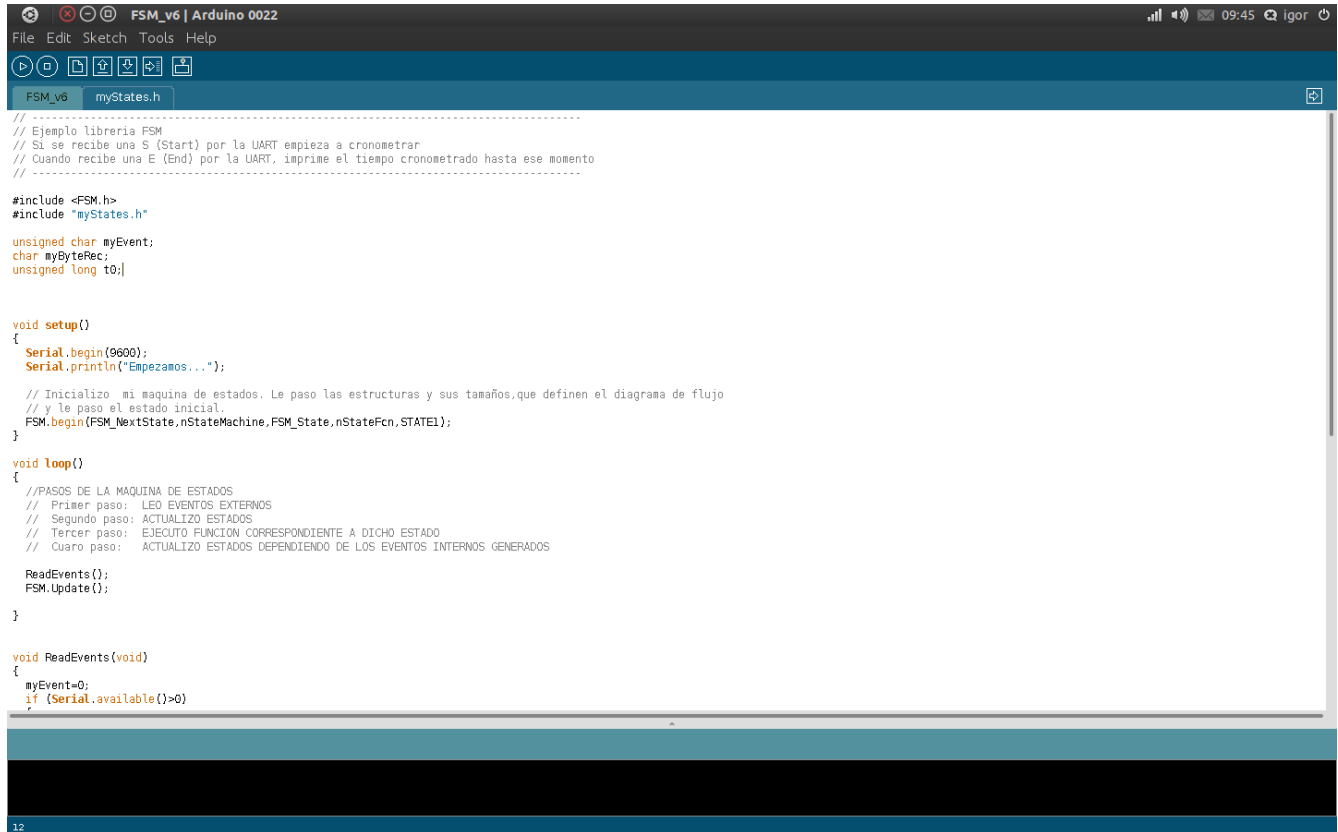
```
#define STATE1      0x01  //Nunca definir un estado como cero (0)  
#define STATE2      0x02  
#define STATE3      0x03  
#define STATE4      0x04  
  
#define EV_S        0x01  //Nunca definir un evento como cero (0).  
#define EV_E        0x02
```

Nuestro sketch tan sólo tiene que realizar lo siguiente:

```
void setup()  
{  
    //Comienza la maquina de estados, para ello se le envia las estructuras descriptivas, su tamaño y  
    //el estado inicial  
    FSM.begin(FSM_NextState,nStateMachine,FSM_State,nStateFcn,STATE1);  
}  
  
void loop()  
{  
    //Leer eventos externos  
    ReadEvents();  
    //Actualizar maquina de estados  
    FSM.Update();  
}
```

Por sencillez y claridad de nuestro código, creamos las definiciones de estados, eventos y estructuras en un \*.h, el cual incluiremos en nuestro sketch gracias a la directiva #include.

Por ejemplo, myStates.h



```
FSM_v6 | Arduino 0022
File Edit Sketch Tools Help

FSM_v6 myStates.h

// .....
// Ejemplo libreria FSM
// Si se recibe una S (Start) por la UART empieza a cronometrar
// Cuando recibe una E (End) por la UART, imprime el tiempo cronometrado hasta ese momento
// .....

#include <FSM.h>
#include "myStates.h"

unsigned char myEvent;
char myByteRec;
unsigned long t0;

void setup()
{
  Serial.begin(9600);
  Serial.println("Empezamos...");

  // Inicializo mi maquina de estados. Le paso las estructuras y sus tamaños, que definen el diagrama de flujo
  // y le paso el estado inicial.
  FSM.begin(FSM_NextState, nStateMachine, FSM_State, nStateFcn, STATE1);
}

void loop()
{
  //PASOS DE LA MAQUINA DE ESTADOS
  // Primer paso: LEO EVENTOS EXTERNOS
  // Segundo paso: ACTUALIZO ESTADOS
  // Tercer paso: EJECUTO FUNCION CORRESPONDIENTE A DICHO ESTADO
  // Cuarto paso: ACTUALIZO ESTADOS DEPENDIENDO DE LOS EVENTOS INTERNOS GENERADOS

  ReadEvents();
  FSM.Update();
}

void ReadEvents(void)
{
  myEvent=0;
  if (Serial.available()>0)
  {
    myEvent=Serial.read();
  }
}
```