GENERIC FINITE-STATE MACHINES

The `gen_fsm` behaviour is somewhat similar to `gen_server` in that it is a specialised version of it. The biggest difference is that rather than handling *calls* and *casts*, we're handling *synchronous* and *asynchronous events*. Much like our dog and cat examples, each state is represented by a function. Again, we'll go through the callbacks our modules need to implement in order to work.

init

This is the same init/1 as used for generic servers, except the return values accepted are `{ok, StateName, Data}`,`{ok, StateName, Data, Timeout}`, `{ok, StateName, Data, hibernate}` and `{stop, Reason}`. The `stop` tuple works in the same manner as for `gen_server`s, and `hibernate` and *`Timeout`* keep the same semantics.

What's new here is that *`StateName`* variable. *`StateName`* is an atom and represents the next callback function to be called.

StateName

The functions StateName/2 and StateName/3 are placeholder names and you are to decide what they will be.

Let's suppose the `init/1` function returns the tuple `{ok, sitting, dog}`. This means the finite state machine will be in a `sitting` state. This is not the same kind of state as we had seen with `gen_server`; it is rather equivalent to the `sit`, `bark` and `wag_tail`states of the previous dog FSM. These states dictate a context in which you handle a given event.

An example of this would be someone calling you on your phone. If you're in the state 'sleeping on a Saturday morning', your reaction might be to yell in the phone. If your state is 'waiting for a job interview', chances are you'll pick the phone and answer politely. On the other hand, if you're in the state 'dead', then I am surprised you can even read this text at all.

Back to our FSM. The `init/1` function said we should be in the `sitting` state. Whenever the `gen_fsm` process receives an event, either the function `sitting/2` or `sitting/3` will be called. The `sitting/2` function is called for asynchronous events and `sitting/3` for synchronous ones.

The arguments for `sitting/2` (or generally `StateName/2`) are *`Event`*, the actual message sent as an event, and*`StateData`*, the data that was carried over the calls. `sitting/2` can then return the tuples `{next_state, NextStateName, NewStateData}`, `{next_state, NextStateName, NewStateData, Timeout}`, `{next_state, NextStateName, NewStateData, hibernate}` and `{stop, Reason, NewStateData}`.
The arguments for `sitting/3` are similar, except there is a *`From`* variable in between *`Event`* and *`StateData`*. The *`From`*variable is used in exactly the same way as it was for `gen_server`s, including gen_fsm:reply/2. The `StateName/3`functions can return the following tuples:

```
{reply, Reply, NextStateName, NewStateData}

{reply, Reply, NextStateName, NewStateData, Timeout}

{reply, Reply, NextStateName, NewStateData, hibernate}



{next_state, NextStateName, NewStateData}

{next_state, NextStateName, NewStateData, Timeout}

{next_state, NextStateName, NewStateData, hibernate}



{stop, Reason, Reply, NewStateData}

{stop, Reason, NewStateData}
```

Note that there's no limit on how many of these functions you can have, as long as they are exported. The atoms returned as *`NextStateName`* in the tuples will determine whether the function will be called or not.

handle_event

In the last section, I mentioned global events that would trigger a specific reaction no matter what state we're in (the dog smelling food will drop whatever it is doing and will instead look for food). For these events that should be treated the same way in every state, the handle_event/3 callback is what you want. The function takes arguments similar to `StateName/2` with the exception that it accepts a *`StateName`* variable in between them, telling you what the state was when the event was received. It returns the same values as `StateName/2`.

handle_sync_event

The handle_sync_event/4 callback is to `StateName/3` what `handle_event/2` is to `StateName/2`. It handles synchronous global events, takes the same parameters and returns the same kind of tuples as `StateName/3`.

Now might be a good time to explain how we know whether an event is global or if it's meant to be sent to a specific state. To determine this we can look at the function used to send an event to the FSM. Asynchronous events aimed at any `StateName/2` function are sent with send_event/2, synchronous events to be picked up by`StateName/3` are to be sent with sync_send_event/2-3.

The two equivalent functions for global events are send_all_state_event/2 and sync_send_all_state_event/2-3(quite a long name).

code_change

This works exactly the same as it did for `gen_server`s except that it takes an extra state parameter when called like`code_change(OldVersion, StateName, Data, Extra)`, and returns a tuple of the form `{ok, NextStateName, NewStateData}`.

terminate

This should, again, act a bit like what we have for generic servers. terminate/3 should do the opposite of `init/1`.

Source : http://learnyousomeerlang.com/finite-state-machines