```
FUNCTION_BLOCK actuator_3P
VAR_INPUT
        in : REAL;
        force : BOOL;
        on : BOOL;
        end_pos : BOOL;
        diag : BOOL;
END_VAR
VAR_INPUT CONSTANT
        min_ontime : TIME := t#10s;
        min_offtime : TIME := t#10s;
        max_runtime : TIME := t#60s;
        auto_diag_time : TIME := t#10d;
        cal_runtime : TIME := t#600s;
        switch_available: BOOL;
END_VAR
VAR_OUTPUT
        out1 : BOOL;
        out2 : BOOL;
        pos : REAL;
        busy : BOOL;
        error : BOOL;
END_VAR
VAR
        val: REAL;
        tx : TIME;
        last_diag: TIME;
        last : TIME;
        runtime_1 : TIME;
        runtime_2 : TIME;
        state : INT;
        force_on : BOOL;
        force_off : BOOL;
        rout: FT_rmp;
        end : R_TRIG;
        cal_timer: TIME;
END_VAR


(*
version 1.0 10 oct 2006
programmer      hugo
tested by       tobias

actuator_3P is an interface for a 3 point actuator.
a 3P actuator is a motor with 2 directions that drives a valve or flap. the position of the valve or flap
is controlled by runtime of the motor foreward or backward.
the available inputs are:
in is a real value which specifies the position of the actuator, 0 = 0% and 1 = 100% any value <0 or
> 1 is limited to 0 or 1.
a force input must be low for normal operation and if switched high, the motor can only be
controlled for 0% or 100% positions with the on input.
```

an and pos input is available for a end_pos switch, this switch indicates the valve or flap to be at either 0% or 100%
a diag input can at any time start a diag cycle which turns the motor towards the 0% position, then measures the runtime to 100% and then the runtime to 0%
these runtimes are stored and used for absolute positioning of the motor during normal operation.
a setup constand "auto_diag_time can be used to specify how often the actuator repeates this auto_diag cycla automatically.
at power on a cycle is perfomed automatically.
a 0 for this value means auto diag is never performed.

the outputs ar out1 and out2 for up and down direction of the motor. to avoid flickering the output can be programmed for min_ontime and min_offtime.
a pos outout indicates the absulute position of the actuator during operation and a busy output indicates the actuator is busy.
a error output is initiated when the specified max_runtime is reached by any direction during debug. also when the up and down direction differs by more then 10% the error output is set.

for absolute positioning accuracy the actuator has a setup variable "cal_runtime" if this is set the actuator measures the total runtime of the motors and when this
cal_runtime is exceeded it resets the actuator to 0% position and sets the output again to assure absolute accuracy.

autodiag and calibration can be shut off by setting the appropriate values to 0.
a switch_available setup variable specifies if end_pos switches are available.

*)

```
(* setup *)
tx := TIME() - last;
end(clk := end_pos);

(* init only when first started *)
IF state = 0 THEN
        runtime_1 := max_runtime;
        runtime_2 := max_runtime;
        state := 1;
        rout(KR := 1000 / TIME_TO_REAL(runtime_1), KF := 1000 /
TIME_TO_REAL(runtime_1) );
        tx := t#0s;
        last := TIME();
        last_diag := last;
END_IF;

(* check for min_offtime and min_ontime and return if it has not elapsed *)
IF NOT busy AND tx < min_offtime THEN RETURN;
ELSIF busy AND tx < min_ontime THEN RETURN;

(* check for last diag cycle and start diag if auto_diag_time has elapsed *)
ELSIF (tx + last - last_diag > auto_diag_time AND state > 10 AND auto_diag_time > t#0s) OR
(diag AND state >= 10) THEN state := 1;

(* check if force pin is high *)
ELSIF force THEN state := 10;
(* check if calibration is necessary *)
ELSIF cal_timer > cal_runtime AND state = 99 AND cal_runtime > t#0s THEN state := 20;
END_IF;

(* state machine for operation *)
CASE state OF

1 :     (* diagnostic start *)
        (* all data is reset and out2 is turned on to make sure we start from top position *)
        last_diag := tx + last;
        diag := 0;
        state := 2;
        pos := 0;
        busy := 1;
        last := tx + last;
        out1 := 0;
        out2 := 1;
        RETURN;

2:      (* wait for max_runtime or end_pos to be reached *)
        IF tx < max_runtime AND NOT end.Q THEN RETURN; END_IF;
        state := 3;
        last := tx + last;
        out1 := 1;
        out2 := 0;
        RETURN;
```

```
3:          (* measure runtime_2 *)
            IF tx < max_runtime AND NOT end.Q THEN RETURN; END_IF;
            state := 4;
            runtime_1 := tx;
            last := tx + last;
            out1 := 0;
            out2 := 1;
            RETURN;

4:          (* measure runtime_1 *)
            IF tx < max_runtime AND NOT end.Q THEN RETURN; END_IF;
            state := 99;
            runtime_2 := tx;
            last := tx + last;
            out2 := 0;
            busy := 0;
            (* reset the output ramp generator to the 0% position as the valve is after diags *)
            rout(rmp := 0, in := 0);
            rout.Rmp := 1;
            cal_timer := t#0s;
            IF differ(TIME_TO_REAL(runtime_1), TIME_TO_REAL(runtime_2),
TIME_TO_REAL(max_runtime) * 0.1) THEN error := TRUE;
            ELSIF runtime_1 > multime(max_runtime, 0.9) AND switch_available THEN error :=
TRUE;
            ELSIF runtime_2 > multime(max_runtime, 0.9) AND switch_available THEN error :=
TRUE;
            END_IF;
            RETURN;

10: (* force pin is active, outputs are forced on or off by the on pin *)
            IF on AND NOT out2 THEN val := 1;
            ELSIF NOT on AND NOT out1 THEN val := 0;
            END_IF;
            IF NOT force THEN state := 99; END_IF;

20 : (* calibrate to allow for correction of positioning error addig up *)
            cal_timer := t#0s;
            state := 21;
            pos := 0;
            busy := 1;
            last := tx + last;
            out1 := 0;
            out2 := 1;
            RETURN;

21:         (* after end_sw or max_runtime we can be sure to have reached the 0% position *)
            IF tx < max_runtime AND NOT end.Q THEN RETURN; END_IF;
            state := 99;
            out2 := 0;
            busy := 0;
            last := tx + last;
```

```
            rout(rmp := 0, in := 0);
            rout.Rmp := 1;
            RETURN;

99: (* normal operation *)
            (* only if input changes for an equivalent of min_ontime then change the output *)
            (* if there is a directionalmchange then wait until last step is finished *)
            IF differ(LIMIT(0,in,1), val, TIME_TO_REAL(min_ontime) /
TIME_TO_REAL(max_runtime)) OR in = 0 OR in = 1 THEN
                        IF in < val AND NOT out1 THEN val := in;
                        ELSIF in > val AND NOT out2 THEN val := in;
                        END_IF;
            END_IF;

END_CASE;

(* output ramp generator *)
rout(in := val);
out1 := rout.busy AND rout.UD;
out2 := rout.busy AND NOT rout.UD;
pos := rout.out;

(* set the busy output and if busy goes low, then set the last time in order to check for min_offtime
*)
IF NOT busy AND rout.busy THEN
            last := tx + last;
ELSIF busy AND NOT rout.busy THEN
            cal_timer := cal_timer + tx;
            last := tx + last;
END_IF;
busy := rout.busy;
```