Beckhoff PLC (TwinCAT 3)

PLC programming by using OOP approach

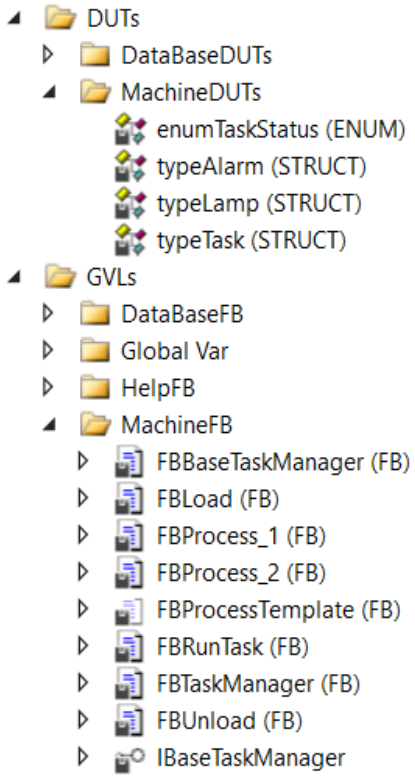
In this paper, we are going to have a look OOP programming more deeply and basic machine example which is consisted of 4 tasks. These tasks might be used for any machine type thanks to their flexibility as a template. In this example, we cover a basic concept but they can be used more complex structure.

What we will do;

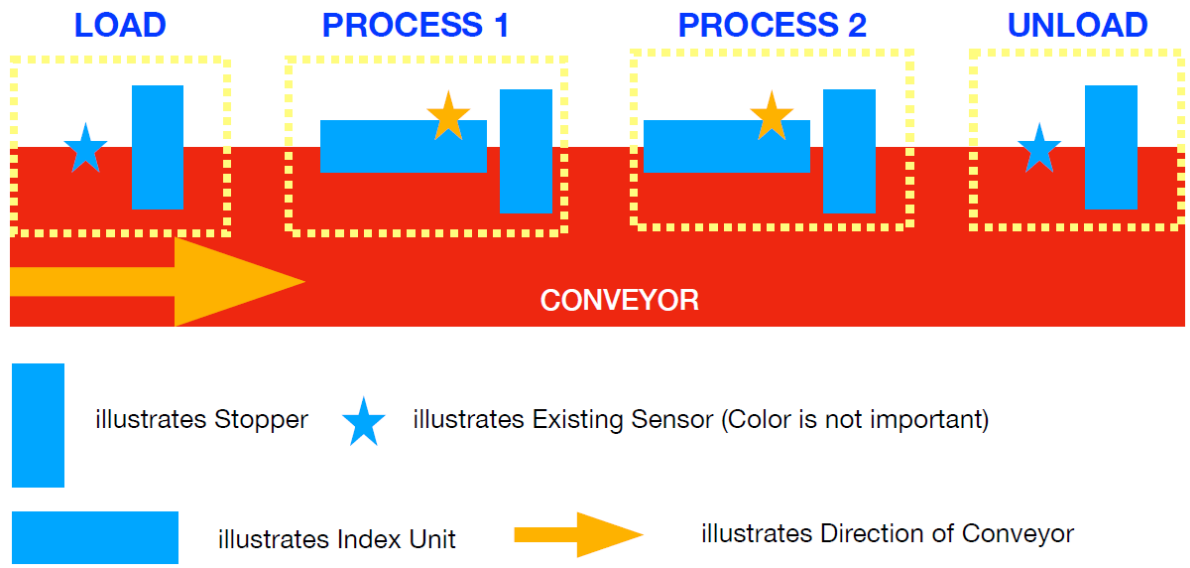
- Program Structure
- FBRunTask
- Execution of a Task

Program Structure

Programming structure of machine Fbs, is similar to database example because our aim is to create a program which obeys solid programming standard and is flexible as possible as it can.

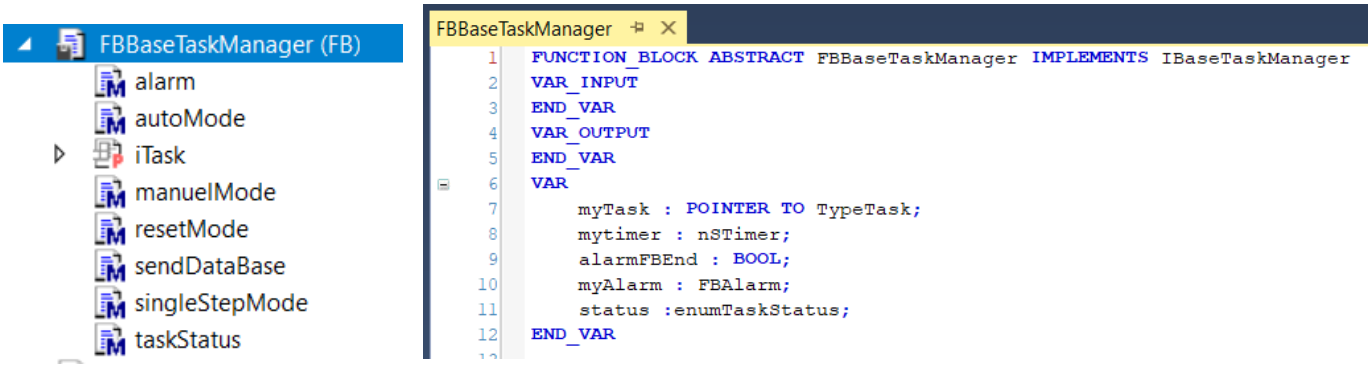


FBBaseTaskManager is our abstract function block which all tasks extend it and IbaseTaskManager which abstract FB implements keeps our property to allow us to use any of structure type for our machine. FBTaskManager is used for being able to create polymorphic programming structure. FbRunTask which is used for avoiding repeat code executes our tasks which are load, process1, process2 and unload. In addition, enumTaskStatus is defining for status which are stop, reset, door, manual, auto and typeTask is structure which is used for all tasks. Furthermore, you can see a simple design of this example below.



Basic specification of machine;

- Conveyor is always on.
- When part is present at load side and process 1 is ready to take part, then load stopper is opened and part is sent to process 1 side.
- When part is present at process 1 side, index unit is moved up and then after finishing process, it is moved down again and if process 2 side is ready to take part, part is sent to process 2 side.
- When part is present at process 2 side, index unit is moved up and then after finishing process, it is moved down again and if unload side is ready to take part, part is sent to unload side.
- When part is present at unload side, stopper is moved down and part is unloaded.
- Index units and stoppers have only one input and 2 sensors. When this input is True, they are moved up, otherwise go down. One sensor is for up position, the other is for down position.
- When machine is at reset status, all index units are moved down and all stoppers are moved up.
- When pressing start input, machine goes to reset status and after finishing reset process, it goes to stop status and then manual status.
- When machine is at manual status and if start input is pressed, then machine goes to automatic status.
- When machine is at automatic status and if stop input is pressed, then machine goes to manual status..



In this example, alarm method is just created for future use because next paper will be about alarm handling and data and state logger. Auto, manual and reset mode are abstract method, so any piece of code wasn't implemented and sendDataBase, singleStepMode and taskStatus are common method for all tasks, so they were implemented in the FBBaseTaskManager.

Inside sendDataBase, when partStatus of task is not empty, first database variables are prepared, then they are checked by blackList and then are sent to database.

FBBaseTaskManager.sendDataBase

```
METHOD  sendDataBase : BOOL
VAR_INPUT
END_VAR

    IF myTask^.partStatus <> '' THEN
        sqlVarInsert.stationName := myTask^.taskName;
        sqlVarInsert.partCode := CONCAT (myTask^.taskName,' Test Part');
        sqlVArInsert.quality := myTask^.partStatus;
        dataBaseManager.blackList();
        IF dataBaseManager.insertData() THEN
            myTask^.partStatus :='';
        END_IF;
    END_IF;
```

SingleStepMode is used for single execution. This means that if stepActive is True, tasks are not working until singleStep is True. Therefore, pressing one time singleStep, all tasks are doing only one process and stops again.

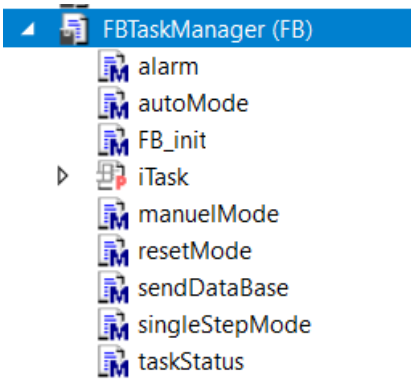
FBBaseTaskManager.singleStepMode

```
METHOD singleStepMode : BOOL
VAR_INPUT
END_VAR


(*****************************************************************************
                                    STOP / STEP MANAGEMENT
*****************************************************************************)
IF myTask^.status = taskStop OR (myTask^.alarmCode <>0 OR myTask^.status = taskDoor) THEN
    singleStepMode := TRUE ;
ELSIF stepActive AND NOT disableStep THEN
    IF NOT singleStep THEN // SignleStep execute (if not disabled) one PLC cycle
        singleStepMode := TRUE ;
    ELSE
        singleStepMode := FALSE ;
    END_IF;
ELSE
    singleStepMode := FALSE ;
END_IF;
```

TaskStatus method is determining which status task need to go and in some cases, task is forced to stop, such as alarm case.

FBBaseTaskManager.taskStatus

```
METHOD taskStatus : BOOL
VAR_INPUT
END_VAR

(*****************************************************************************
                                TASK Status
*****************************************************************************)
IF NOT myTask^.resetOK AND resetReq AND NOT alarmReq AND NOT alarmON THEN myTask^.status:=taskReset; END_IF;  // -->reset cycle
IF myTask^.resetOK AND myTask^.status=taskReset THEN myTask^.status:=taskStop; END_IF;  // -->end reset
IF myTask^.status=taskStop AND automaticON THEN myTask^.status:=taskAuto; END_IF;  // -->start automatic cycle
IF myTask^.status=taskStop AND manualON THEN myTask^.status:=taskMan; END_IF;  // -->start manual cycle
// Store TIME at which there is STOP request
IF NOT alarmReq AND NOT manualReq AND NOT automaticReq  THEN mytimer.setTime(); END_IF; // Timer store the time when AutoMan to
// ITSELF IN ALARM: STOP after 0.2 sec to manage AlarmReq before setting TASK_STOP
IF myTask^.status<>taskStop AND myTask^.status<>taskDoor AND myTask^.alarmCode <>0 AND mytimer.getTime()>LREAL#0.2 THEN
      myTask^.status:=taskStop;  // FORCED STOP!
   END_IF;
// Timeout for Stopping
IF myTask^.status<>taskStop AND myTask^.status<>taskDoor AND (alarmReq OR manualReq OR automaticReq ) AND NOT disableStep THEN
   IF (mytimer.getTime()>LREAL#10)  THEN // 10 sec Tout
      myTask^.status:=taskStop;  // FORCED STOP!
   END_IF;
END_IF;
```



FBTaskManager

```
FUNCTION_BLOCK FBTaskManager
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    functionGroup : REFERENCE TO FBBaseTaskManager;
END_VAR
```

FBTaskManager is used for being able to create polymorphic programming structure and keeps address of FBBaseTaskManager which is extended by tasks, then inside init FB, address of task which will be used assign to functionGroup and iTask property is set.

FBTaskManager.FB_init

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start / cold start)
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy code (online change)
    taskGroup : REFERENCE TO FBBaseTaskManager;
    taskIndex : POINTER TO UDINT ;
END_VAR
```

```
        // İnit DataBase
    functionGroup REF= taskGroup;
    functionGroup.iTask := taskIndex ;
```

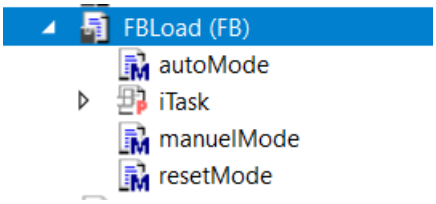
The methods of FBTaskManager just call related functionGroup's method. You can below;

FBTaskManager.alarm

```
METHOD alarm : BOOL
VAR_INPUT
END_VAR
```

```
alarm := functionGroup.alarm();
```

Now we are going to look load task and type of task. You see below that FBLoad has only methods and one property which they have to be there because FBLoad extends FBBaseTaskManager which is abstract FB.



ad

```
FUNCTION_BLOCK FBLoad EXTENDS FBBaseTaskManager
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
```

Task

```
TYPE typeTask :
STRUCT
    taskName            :STRING;
    taskNumber          :INT;
    alarmCode           :INT;
    messageCode         :INT;
    resetOK             :BOOL;
    status              :enumTaskStatus;
    stepR               :INT;
    stepA               :INT;
    stepM               :INT;
    stepD               :INT;
    cycleTime           :LREAL;
    partStatus          :STRING;
END_STRUCT
END_TYPE
```

In autoMode, manualMode and resetMode, we implemented specification of machine and you can change your own needs.

FBLoad.resetMode

```
METHOD resetMode : BOOL
VAR_INPUT
END_VAR
```

```
IF singleStepMode() THEN RETURN; END_IF;
(*********************************************************
                                    RESET
*********************************************************)
IF myTask^.Status=taskReset THEN
    CASE myTask^.StepR OF
        0: // Start Reset
            IF NOT resetReq THEN myTask^.Status:=taskStop;
            ELSE myTask^.stepR:=10;
            END_IF;
        10://
            myTask^.stepR:=20;
        20://
            loadStopperUp := TRUE;
            myTask^.stepR:=30;
        30:
            IF loadStopperUpSens THEN
                myTask^.stepR:=40;
            END_IF;
        40:
            myTask^.resetOK:=TRUE;
            myTask^.stepR:=0;
    END_CASE;
END_IF;
```

FBLoad.manuelMode

```
METHOD manuelMode : BOOL
VAR_INPUT
END_VAR
```

```
IF singleStepMode() THEN RETURN; END_IF;
(*********************************************************
                                    MANUAL
*********************************************************)
IF myTask^.Status=taskMan  THEN
    CASE myTask^.stepM OF
        //*******************  MANUAL IN  *******************
        0:// ENTERING MAN
            myTask^.stepM:=100;
        //*******************  MANUAL CYCLE  *******************
        100:// MAN CYCLE
            IF automaticReq THEN myTask^.stepM:=200; // AUTO
            ELSIF NOT manualON THEN myTask^.status:=taskStop; //ALM
            ELSE myTask^.stepM:=110; // MAN CYCLE
            END_IF;
        110: //
            myTask^.stepM:=100;
        //*******************  MANUAL OUT  *******************
        200:// EXITING MAN
            myTask^.stepM:=0;
            myTask^.status:=taskStop;
    END_CASE;
END_IF;
```

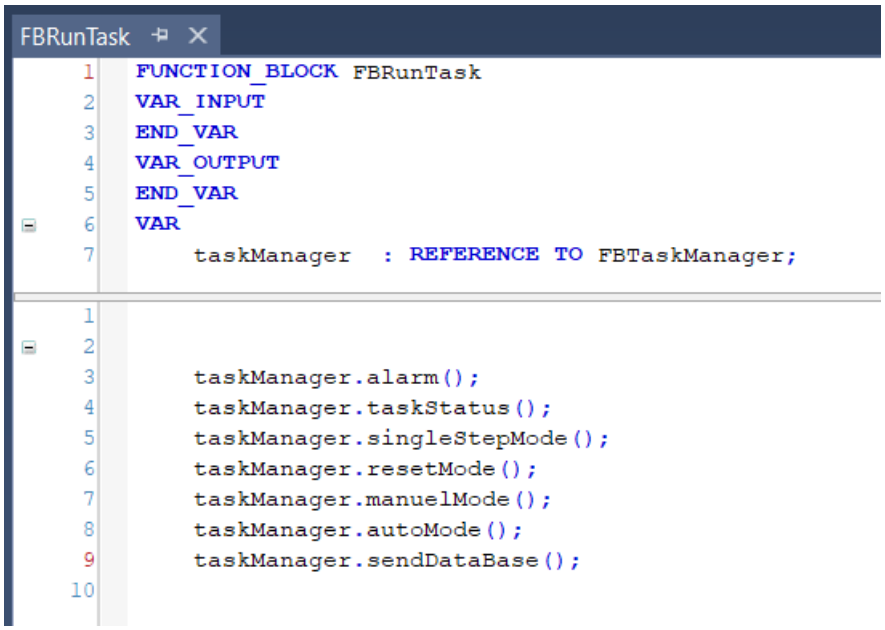
Actually, inside manualMode, program just waits for going automatic mode in this case but in more complex machine, we might need to do some staff in the manual mode when it goes in or goes out. For simplicity, we suppose that part is always ‘OK’ in autoMode but in real machine, we need to take this information from outside of PLC or result of some process.

FBLoad.autoMode

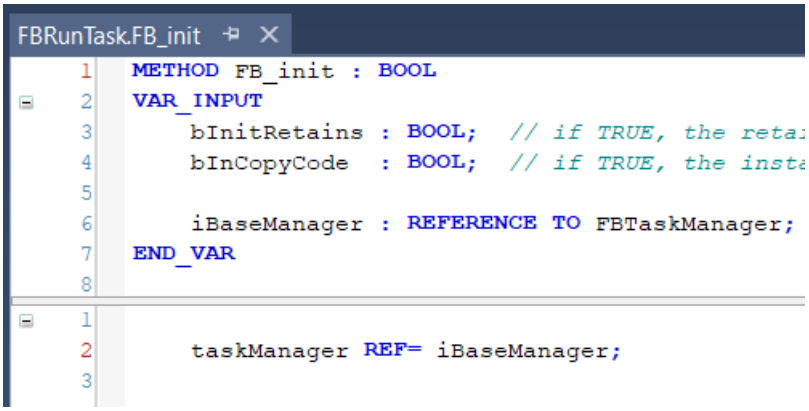
```
                                        AUTO
(*************************************************************************)
IF singleStepMode() THEN RETURN; END_IF;
IF myTask^.Status = taskAuto THEN
    CASE myTask^.stepA OF
        0:  //
            IF NOT automaticON THEN myTask^.status:= taskStop;
            ELSE myTask^.stepA:=10;
            END_IF;
        10:
            IF partPresentAtLoad THEN
                myTask^.stepA:=20;
            END_IF
        20:
            IF process1ReadyToTakePart THEN
                myTask^.stepA:=30;
            END_IF;
        30:
            loadStopperUP := FALSE ;
            myTask^.stepA:=40;
        40:
            IF loadStopperDownSens THEN
                myTask^.stepA:=50;
            END_IF;
        50:
            IF NOT partPresentAtLoad THEN
                myTask^.stepA:=60;
            END_IF
        60:
            loadStopperUP := TRUE; ;
            myTask^.stepA:=70;
        70:
            IF loadStopperUpSens THEN
                myTask^.stepA:=80;
            END_IF;
        80:
            myTask^.partStatus := WSTRING_TO_STRING("OK") ;
            myTask^.stepA:=90;
        90:
            IF myTask^.partStatus = '' THEN
                myTask^.stepA:=100;
            END_IF;
        100:
            myTask^.stepA:=0;
   END_CASE;
```

FBRunTask

This function block helps us to avoid repeat code because we have 4 tasks and every task has seven method, so it makes twenty eight methods in total. To be able to use those methods, we have to call but we don't want to write the same code more than one if we have no any other option.



FBRunTask keeps reference of FBTaskManager and init function sets address of it.



In the main program, we created instance of tasks and FBRunTask, and then we called them in the main.

MAIN

```
PROGRAM MAIN
VAR
    localTime: NT_GetTime;

    fbLoad  : FBLoad;
    taskManagerLoad : FBTaskManager(fbLoad,ADR(task[1]));
    runLoad : FBRunTask(taskManagerLoad);

    fbProcess1  : FBProcess_1;
    taskManagerProcess1 : FBTaskManager(fbProcess1,ADR(task[2]));
    runProcess1 : FBRunTask(taskManagerProcess1);

    fbProcess2  : FBProcess_2;
    taskManagerProcess2 : FBTaskManager(fbProcess2,ADR(task[3]));
    runProcess2 : FBRunTask(taskManagerProcess2);

    fbUnload    : FBUnload;
    taskManagerUnload : FBTaskManager(fbUnload,ADR(task[4]));
    runUnload : FBRunTask(taskManagerUnload);
END_VAR
```

```

    AutoManManager();
    runLoad();
    runProcess1();
    runProcess2();
    runUnload();
    InternalCommunicaiton();

localTime(START:= TRUE , TMOUT := T#0.001S);
localTime(START:= FALSE , TMOUT := T#0.001S);
myLocalTime := localTime.TIMESTR;
```

Execution

Now we will look an execution of a task but if I add pic of it here, it will be so many photo here, so I uploaded a video about it instead of picture. You can find video here ;

https://www.youtube.com/watch?v=g2m1lVcR_6Q

As a conclusion, we did a machine by using OOP, so this gives us real flexibility and solid programming standard. Result of these, Our program will be open to changes and support customers needs as possible as it can. Please reach out to me If you have any question or criticism about it, I would like to answer or hear.

I wish you all the best

Cheers

Murat TOPRAK