Beckhoff PLC (TwinCAT 3)

PLC programming by using OOP approach

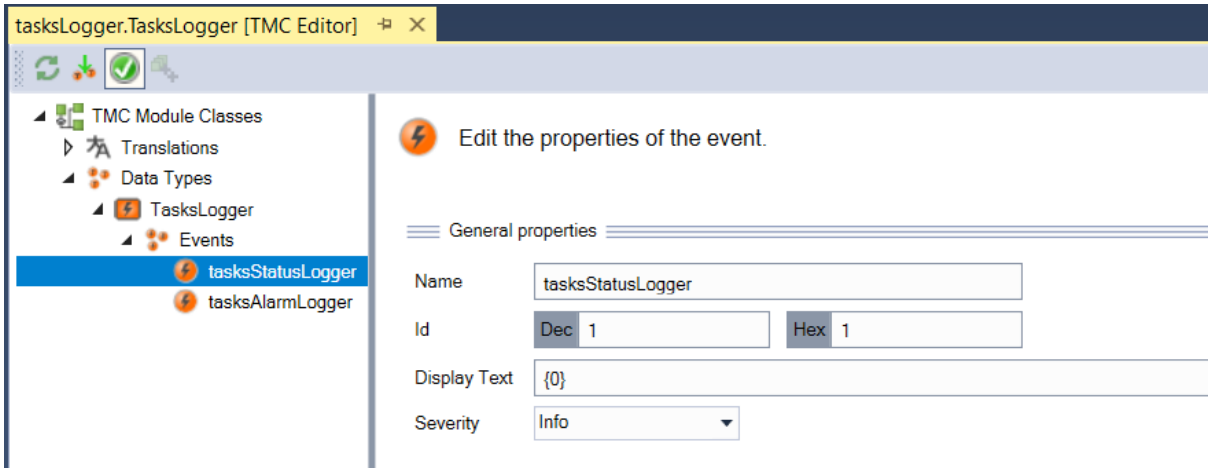
As a programmer, we all try to deal with errors or problems to keep our machine or production running. Therefore, we need a structure to show us a path to find solution for our machine's problems. For this purpose, I believe that the most effective way is to have a logger. Furthermore, we will have more information about our system because we will be able to look history of our system. For full understanding, please read previous papers.

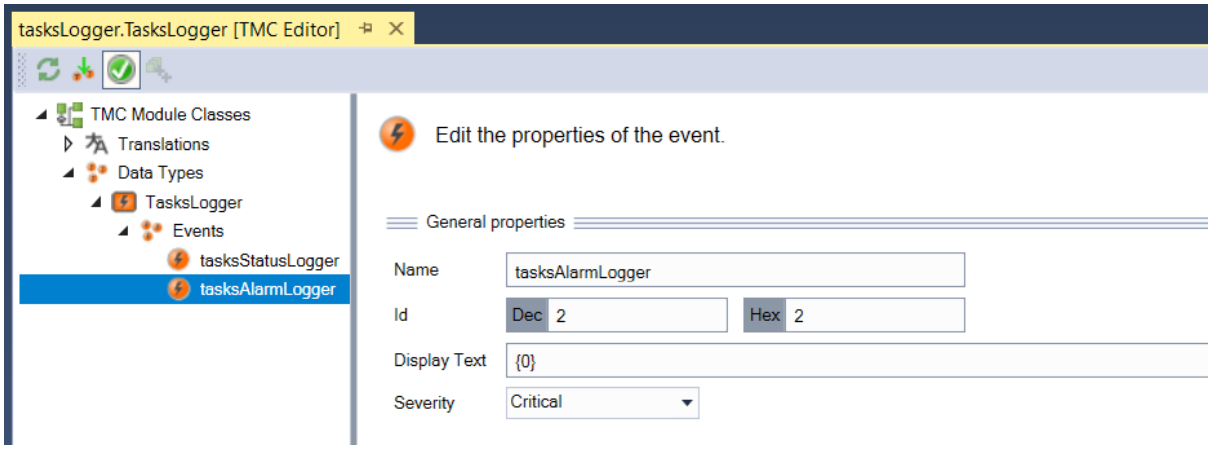
What we will do;

- Event Classes
- Logger FBs
- Calling Loggers

Event Classes

In Beckhoff system, we need to create event class in order to have a logger. In our case, we will compose an event class and two events. One of the event keeps our task status and the other keeps our alarms.

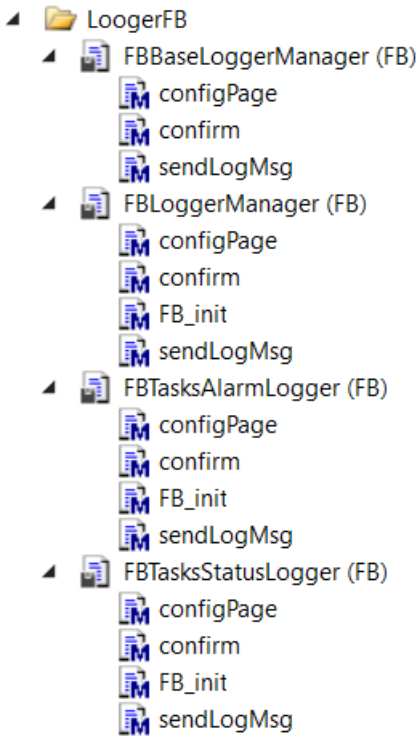




We have to put {0} inside the display text to able to give string of events out side and we need to choose severity of events. In addition, Tc3_EventLogger library must be added to solution.

Logger FBs

We follow the same procedure with database example. Therefore, firstly we are going to create an abstract block which our logger blocks are extending it, then a manager block is created to keep address of logger block in order to have a polymorphic structure.



FBBaseLoggerManager is our abstract block which our alarm and status blocks extend it and FBLoggerManager keeps address of blocks.

After giving an error, we have to confirm it to able to log new error. In addition, we have a configuration page and in this page, error's string is assigned depending on error's code. SendLogMsg is used for logging a string value to corresponding logger.

In two logger blocks, we use init method to transfer which event we are going to use.

FBTasksAlarmLogger.FB_init

```
METHOD FB_init : BOOL
VAR_INPUT
    bInitRetains : BOOL;  // if TRUE, the retain variables are initialized (warm start
    bInCopyCode  : BOOL;  // if TRUE, the instance afterwards gets moved into the copy
    loggerType   : TcEventEntry ;
END_VAR
```

```

    loggerResult := userLogger.CreateEx( loggerType, TRUE,0, ); // create a logger

```

Inside configPage method, we give string value for an error case of all tasks but if you want, you can do this in your HMI and maybe it will be more easier. I like to manage everything from a program as possible as and that's why I did this way.

FBTasksAlarmLogger.configPage

```
METHOD configPage : BOOL
VAR_INPUT
END_VAR
VAR
    ii  : INT;
END_VAR
```

```

FOR ii := 1 TO numberOfTasks DO
    CASE task[ii].alarmCode OF
        (* Task Load *)
        100:
            task[ii].alarmString := ' Test Alarm 0 ';
        101:
            task[ii].alarmString := ' Test Alarm 1 ';
        (* Task Process 1 *)
        200:
            task[ii].alarmString := ' Test Alarm 0 ';
        201:
            task[ii].alarmString := ' Test Alarm 1 ';
        (* Task Process 2 *)
        300:
            task[ii].alarmString := ' Test Alarm 0 ';
        301:
            task[ii].alarmString := ' Test Alarm 1 ';
        (* Task Unload *)
        400:
            task[ii].alarmString := ' Test Alarm 0 ';
        401:
            task[ii].alarmString := ' Test Alarm 1 ';
    END_CASE
END_FOR
```

FBTasksAlarmLogger.confirm

```
METHOD confirm : BOOL
VAR_INPUT
END_VAR
```

```

    userLogger.Confirm(0);
    userLogger.Clear(0,TRUE);
```

Inside sendLogMsg for alarm logger, firstly configPage is called to get string value if there is an error then if there is, we format our string values and final step is to add to logger and to raise.

FBTasksAlarmLogger.sendLogMsg

```
    FOR ii:= 1 TO numberOfTasks DO
        IF (tempTask[ii].alarmCode <> task[ii].alarmCode) AND (task[ii].alarmCode <> 0)  THEN
            fdataTime :=  SYSTEMTIME_TO_STRING(myLocalTime);
            fStationName :=task[ii].taskName ;
            CASE task[ii].status OF
                taskStop:
                    fStationStatus := 'STOP';
                taskReset:
                    fStationStatus := 'RESET';
                taskMan:
                    fStationStatus := 'MANUAL';
                taskAuto:
                    fStationStatus := 'AUTO';
            END_CASE

            fSAuto := INT_TO_STRING(task[ii].stepA);
            fSmanuel := INT_TO_STRING(task[ii].stepM);
            fSReset := INT_TO_STRING(task[ii].stepR);
            fSAlarm := task[ii].alarmString;
            fbFormatString(
                    sFormat    := 'DATE = $'%S$' , STATION NAME = $'%S$' , ALARM = $'%S$' ,STATI
                    arg1       := F_STRING(in := fdataTime),
                    arg2       := F_STRING(in := fstationName),
                    arg3       := F_STRING(in := fSAlarm),
                    arg4       := F_STRING(in := fStationStatus),
                    arg5       := F_STRING(in := fSAuto),
                    arg6       := F_STRING(in := fSmanuel),
                    arg7       := F_STRING(in := fSReset),
                    sOut        => str,
                    bError      => fError,
                    nErrId      => fErrid);

            userLogger.ipArguments.Clear().AddString(str); //set Argument
            userLogger.Raise(0); // send message
            task[ii].alarmString := '' ;
            tempTask := task;
        END_IF;
    END_FOR;
sendLogMsg := TRUE;
```

On the other hand, for status logger we don't call configPage and we check status and if there is a difference, then we log status of task and steps of task.

In addition, raise method is used in alarm logger however send method is used in status logger because one of them is an instance of FB_TcAlarm and the other is an instance of FB_TcMessage.

Normally if a data will be logged, we need to trigger that logger in time which the data is created because the data can be lost. However In this example we try to create a general logger and avoid so much writing issue.

FBTasksStatusLogger.sendLogMsg

```
    FOR ii:= 1 TO numberOfTasks DO
        IF (tempTask[ii].status <> task[ii].status) OR  (tempTask[ii].stepA <> task[ii].stepA) OR (tempTask[ii].stepR <> task[ii].stepR) THEN
            fdataTime :=  SYSTEMTIME_TO_STRING(myLocalTime);
            fStationName :=task[ii].taskName ;
            CASE task[ii].status OF
                taskStop:
                    fStationStatus := 'STOP';
                taskReset:
                    fStationStatus := 'RESET';
                taskMan:
                    fStationStatus := 'MANUAL';
                taskAuto:
                    fStationStatus := 'AUTO';
            END_CASE

            fSAuto := INT_TO_STRING(task[ii].stepA);
            fSmanuel := INT_TO_STRING(task[ii].stepM);
            fSReset := INT_TO_STRING(task[ii].stepR);
            fbFormatString(
                    sFormat   := 'DATE = $'%S$' , STATION NAME = $'%S$' , STATION STATUS = $'%S$' , STATION AUTO STEP = $'%S$' , STATION MANUAI
                    arg1      := F_STRING(in := fdataTime),
                    arg2      := F_STRING(in := fstationName),
                    arg3      := F_STRING(in := fStationStatus),
                    arg4      := F_STRING(in := fSAuto),
                    arg5      := F_STRING(in := fSmanuel),
                    arg6      := F_STRING(in := fSReset),
                    sOut       => str,
                    bError     => fError,
                    nErrId     => fErrid);

            userLogger.ipArguments.Clear().AddString(str); //set Argument
            userLogger.Send(0); // send message
            tempTask := task;
        END_IF;
    END_FOR;
sendLogMsg := TRUE;
```

Inside FBAlarm, in case 1, we wait to stop all tasks and then we reset and confirm alarm with inpRestore.

FBAlarm

```
(* MACHINE KO ************************************************************)
IF MachineKO THEN // Reset FB. Reset of the array in AutoMan task at next RESET
    alarmEnd:=FALSE;
    FBIndex:=0;
    RETURN;
END_IF;

(* START/STOP FB *********************************************************)
IF  enable = TRUE AND FBIndex=0 THEN // Start FB
    alarmEnd:=FALSE;
    FBIndex:=1;
END_IF;
IF  enable = FALSE AND FBIndex=4 THEN // End FB
    alarmEnd:=FALSE;
    FBIndex:=0;
END_IF;

(* FB *********************************************************************)
CASE FBIndex OF
    1:// Wait all tasks stopped ->2
        alarmReq:=TRUE;
        IF AlarmON THEN
            FBIndex:=2;
        END_IF;
    2: // MANAGE ALARM: Wait for button RESTORE ON ->3
        IF inpRestore=TRUE THEN FBIndex:=3; END_IF;
    3:  // Wait button RESTORE OFF, remove alarm + END ->4
        IF inpRestore=FALSE THEN
            tasksAlarmLoggerManager.confirm();
            alarmReq:=FALSE; // Reset Request Main to stop all tasks if there are n
            lamp.Alarm:=FALSE;
            alarmEnd:=TRUE;
            FBIndex:=4;
        END_IF;
END_CASE;
```

Calling Loggers

We have two loggers and we create instance of them in global area because we want to use them anywhere of program.

LoggerGlobal

```
VAR_GLOBAL
    tasksStatusLogger: FBTasksStatusLogger(TC_EVENTS.TasksLogger.tasksStatusLogger);
    tasksAlarmLogger: FBTasksAlarmLogger(TC_EVENTS.TasksLogger.tasksAlarmLogger);
    tasksStatusLoggerManager:FBLoggerManager(tasksStatusLogger);
    tasksAlarmLoggerManager:FBLoggerManager(tasksAlarmLogger);
END_VAR
```

For adding logger to our system, we just need to call them inside FBRunTask.

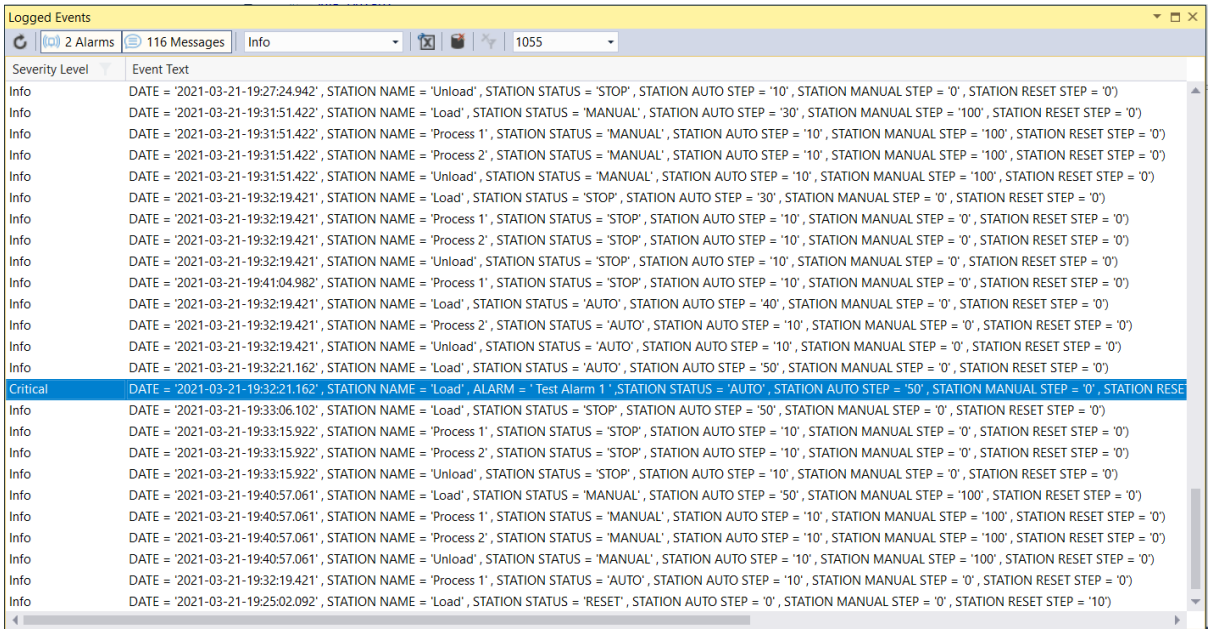
FBRunTask

```
FUNCTION_BLOCK FBRunTask
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
    taskManager  : REFERENCE TO FBTaskManager;
END_VAR
```

```


    taskManager.alarm();
    taskManager.taskStatus();
    taskManager.singleStepMode();
    taskManager.resetMode();
    taskManager.manuelMode();
    taskManager.autoMode();
    taskManager.sendDataBase();
    tasksStatusLoggerManager.sendLogMsg();
    tasksAlarmLoggerManager.sendLogMsg();
```

After running our program and testing our loggers, you can see logged event below.



Now we have lots of information about our system. For example, which task give error, what is the task status or what is the step of that status. Actually we can extend our loggers by adding more information in it.

As a conclusion, we did loggers by using OOP, so we have real information about our system and we can solve many problem of our machine by just looking our logger. Result of these, we can handle many problems easly. 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