**TYPE EventType** : (**Null:= 0**, Message, Alarm ); **END\_TYPE**

**TYPE EventSeverity** : (Verbose:=0, Info, Error, Warning, Critical ); **END\_TYPE**

**TYPE ST\_Event :**

**STRUCT**

EventType : EventType;

EventSeverity : EventSeverity;

EventIdentity : UDINT;

EventText : STRING(255);

TimeStamp : DATE\_AND\_TIME;

**END\_STRUCT**

**END\_TYPE**

**PROGRAM PLC\_PRG**

VAR

EventType\_1 : EventType := EventType.Alarm;

EventSeverity\_1 : EventSeverity := EventSeverity.Warning;

EventIdentity\_1 : UDINT := 23;

EventText\_1 : STRING(255) := 'The sensor value is high';

EventType\_2 : EventType := EventType.Alarm;

EventSeverity\_2 : EventSeverity := EventSeverity.Critical;

EventIdentity\_2 : UDINT := 21;

EventText\_2 : STRING(255) := 'The sensor value is critically high';

EventType\_3 : EventType := EventType.Message;

EventSeverity\_3 : EventSeverity := EventSeverity.Info;

EventIdentity\_3 : UDINT := 11;

EventText\_3 : STRING(255) := 'The sensor value is in range';

StoreEventToTxt : StoreEventToTXTFile;

EventLogger : EventLogger(StoreEventToTxt);

WriteEvent1 : BOOL := FALSE;

WriteEvent2 : BOOL := FALSE;

WriteEvent3 : BOOL := FALSE;

END\_VAR

IF WriteEvent1 THEN

WriteEvent1 := FALSE;

EventLogger.AddEvent(in\_EventType := EventType\_1,

in\_EventSeverity:= EventSeverity\_1,

in\_EventIdentity := EventIdentity\_1,

in\_EventText := EventText\_1);

END\_IF

IF WriteEvent2 THEN

WriteEvent2 := FALSE;

EventLogger.AddEvent(in\_EventType := EventType\_2,

in\_EventSeverity := EventSeverity\_2,

in\_EventIdentity := EventIdentity\_2,

in\_EventText := EventText\_2);

END\_IF

IF WriteEvent3 THEN

WriteEvent3 := FALSE;

EventLogger.AddEvent(in\_EventType := EventType\_3,

in\_EventSeverity := EventSeverity\_3,

in\_EventIdentity := EventIdentity\_3,

in\_EventText := EventText\_3);

END\_IF

writeTextTest();

**FUNCTION\_BLOCK** **EventLogger**

VAR

EventBuffer : ARRAY[1..MAXIMUM\_SIZE\_OF\_EVENT\_BUFFER] OF ST\_Event;

EventStringBuffer : ARRAY[1..MAXIMUM\_SIZE\_OF\_EVENT\_BUFFER] OF STRING(255);

CurrentEventBufferIndex : INT := 0;

\_PersistentEventStorage : I\_PersistentEventStorage;

END\_VAR

VAR CONSTANT

MAXIMUM\_SIZE\_OF\_EVENT\_BUFFER : INT := 100;

END\_VAR

METHOD FB\_Init: BOOL

VAR\_INPUT

bInitRetains: BOOL; // TRUE: the retain variables are initialized (reset warm / reset cold)

bInCopyCode: BOOL; // TRUE: the instance will be copied to the copy code afterward (online change)

iPersistentEventStorage : I\_PersistentEventStorage;

END\_VAR

THIS^.\_PersistentEventStorage := iPersistentEventStorage;

**INTERFACE** I\_PersistentEventStorage

**METHOD** StoreEvent : BOOL

VAR\_INPUT

stEvent : ST\_Event;

END\_VAR

(\*Returns TRUE if Add is sucessful and returns FALSE if buffer is full\*)

**METHOD PUBLIC AddEvent : BOOL**

VAR\_INPUT

in\_EventType : EventType;

in\_EventSeverity : EventSeverity;

in\_EventIdentity : UDINT;

in\_EventText : STRING(255);

EventBufferIsFULL : BOOL := FALSE;

END\_VAR

IsEventBufferFull(Q => EventBufferIsFULL);

IF NOT EventBufferIsFULL THEN

CurrentEventBufferIndex := CurrentEventBufferIndex + 1;

EventBuffer[CurrentEventBufferIndex].EventType := in\_EventType;

EventBuffer[CurrentEventBufferIndex].EventSeverity := in\_EventSeverity;

EventBuffer[CurrentEventBufferIndex].EventIdentity := in\_EventIdentity;

EventBuffer[CurrentEventBufferIndex].EventText := in\_EventText;

UpdateSystemTimestampWithCurrentTime(stEvent := EventBuffer[CurrentEventBufferIndex]);

\_PersistentEventStorage.StoreEvent(stEvent := EventBuffer[CurrentEventBufferIndex] );

AddEvent := TRUE;

ELSE

AddEvent := FALSE;

END\_IF

**METHOD PRIVATE IsEventBufferFull : BOOL**

VAR\_OUTPUT

Q : BOOL; // Q is TRUE if eventBuffer is FULL

END\_VAR

//IsEventBufferFull := ((CurrentEventBufferIndex + 1) > MAXIMUM\_SIZE\_OF\_EVENT\_BUFFER);

Q := ((CurrentEventBufferIndex + 1) > MAXIMUM\_SIZE\_OF\_EVENT\_BUFFER);

**FUNCTION UpdateSystemTimestampWithCurrentTime**

VAR\_INPUT

stEvent : REFERENCE TO ST\_EVENT;

END\_VAR

VAR

UTC\_DateAndTime : UDINT;

Local\_DateAndTime : UDINT;

Result : UDINT;

END\_VAR

//get UTC date and time, store as UDINT

UTC\_DateAndTime := TO\_UDINT(SysTimeRtcGet(Result)); // UDINT#1528268918

// convert to local zone time, store as UDINT

Result := SysTimeRTCConvertUTCToLocal(UTC\_DateAndTime, Local\_DateAndTime); // UDINT#1528276118

//convert from UDINT to Date\_AND\_TIME data type

stEvent.TimeStamp := TO\_DT(Local\_DateAndTime); // DT#2018-6-6-9:8:38

**FUNCTION\_BLOCK** StoreEventToTXTFile IMPLEMENTS I\_PersistentEventStorage

VAR

strFileName : STRING := 'CodesysTutorial.txt'; //C:\ProgramData\CODESYS\Simulation\PlcLogic\CodesysTutorial.txt

strStringToWrite : STRING(255); //$N new line control character

strPrevWrittenString: STRING(255):= '';

stFileHandle : SysFile.RTS\_IEC\_HANDLE;

stResult : SysFile.RTS\_IEC\_RESULT;

pbyBufferToWrite : POINTER TO BYTE := ADR(strStringToWrite);

nSizeWritten : \_\_XWORD;

bIsWritten : BOOL;

eWriteState : (FILE\_OPEN\_TRIGGER, FILE\_OPEN, WAIT\_FOR\_EVENT, WRITE\_EVENT, FILE\_CLOSE\_TRIGGER, FILE\_CLOSE, ERROR);

OpenFile : File.Open(sFileName:= 'Events.log',eFileMode := FILE.MODE.MAPPENDPLUS);

WriteToFile : File.Write();

CloseFile : File.Close();

END\_VAR

CASE eWriteState OF

FILE\_OPEN\_TRIGGER:

OpenFile(xExecute:= TRUE);

eWriteState:= FILE\_OPEN;

FILE\_OPEN:

OpenFile(xExecute := FALSE);

IF OpenFile.xError THEN

eWriteState:= ERROR;

ELSIF NOT OpenFile.xBusy AND OpenFile.hFile <> 0 THEN

eWriteState := WAIT\_FOR\_EVENT;

END\_IF

WAIT\_FOR\_EVENT:

// DO NOTHING

WRITE\_EVENT:

WriteToFile(xExecute := TRUE);

FILE\_CLOSE\_TRIGGER:

//

FILE\_CLOSE:

//

END\_CASE

**METHOD** EventToStringConversion : STRING

VAR\_INPUT

in\_Event : ST\_Event;

END\_VAR

VAR

EventTypeString : STRING;

EventSeverityString : STRING;

EventIdentityString : STRING;

EventTextString : STRING;

EventTimeStampString : STRING;

NewLine : STRING := '$N';

Comma : STRING := ',';

END\_VAR

CASE in\_Event.EventType OF

EventType. Message:

EventTypeString := 'Message,';

EventType.Alarm:

EventTypeString := 'Alarm,';

END\_CASE

//Event Severity

CASE in\_Event.EventSeverity OF

EventSeverity.Verbose :

EventSeverityString := 'Verbose,';

EventSeverity.Info:

EventSeverityString := 'Info,';

EventSeverity.Error:

EventSeverityString := 'Error,';

EventSeverity.Warning:

EventSeverityString := 'Warning,';

EventSeverity.Critical:

EventSeverityString := 'Critical,';

END\_CASE

EventIdentityString := UDINT\_TO\_STRING(in\_Event.EventIdentity);

EventTextString := in\_Event.EventText;

EventTimeStampString := DT\_TO\_STRING(in\_Event.TimeStamp);

EventToStringConversion:='';

EventToStringConversion:= CONCAT(EventTypeString, EventSeverityString);

EventToStringConversion:= CONCAT(EventToStringConversion, EventIdentityString);

EventToStringConversion:= CONCAT(EventToStringConversion, Comma);

EventToStringConversion:= CONCAT(EventToStringConversion, EventTextString);

EventToStringConversion:= CONCAT(EventToStringConversion, Comma);

EventToStringConversion:= CONCAT(EventToStringConversion, EventTimeStampString);

EventToStringConversion:= CONCAT(EventToStringConversion, NewLine);

**METHOD** IsEventStored : BOOL

IF strStringToWrite <> strPrevWrittenString THEN

IsEventStored := FALSE;

strPrevWrittenString := strStringToWrite;

ELSE

IsEventStored := TRUE;

END\_IF

**METHOD** StoreEvent : BOOL

VAR\_INPUT

stEvent : ST\_Event;

END\_VAR

IF eWriteState = WAIT\_FOR\_EVENT THEN

eWriteState := WRITE\_EVENT;

strStringToWrite := eventToStringConversion(in\_Event := stEvent);

END\_IF

(\*

bIsWritten := isEventStored();

IF NOT bIsWritten THEN

//Open File

stFileHandle := SysFileOpen(szFile := strFileName, am:= SysFile.AM\_APPEND , pResult:=ADR(stResult));

//Write File

nSizeWritten := SysFileWrite(hFile := stFileHandle, pbyBuffer:= pbyBufferToWrite

,ulSize:= TO\_\_\_XWORD(LEN(strStringToWrite)),pResult := ADR(stResult));

//Close File

stResult := SysFileClose(hFile:= stFileHandle);

IF nSizeWritten > 0 THEN

bIsWritten := TRUE;

END\_IF

END\_IF \*)