PLC-logger system description



Customer: Vistaprint

Foreword: Description oft he implemented logger system

Project: Promo V2 printer

Project number: 10116.02.01

Version: 0.1

Filing: D:\60\_Projekte\10116 Vista Print\02 Printer - GEMINI\01 Development\20\_Engineering\20\_Development\Specification\Logger\_System\_Description\_V0.1.docx

# Contents

[1 Contents 2](#_Toc402278928)

[2 Basics 4](#_Toc402278929)

[2.1 Introduction 4](#_Toc402278930)

[2.2 Reference documentation 4](#_Toc402278931)

[2.3 Version 4](#_Toc402278932)

[2.4 Abbreviations, definitions, glossary 4](#_Toc402278933)

[3 Starting position 5](#_Toc402278934)

[3.1 Short summary of the concept idea 5](#_Toc402278935)

[4 Best practice 6](#_Toc402278936)

[4.1 Define/use state log 6](#_Toc402278937)

[4.1.1 Usage with base controller to write messages 6](#_Toc402278938)

[4.1.2 File write 6](#_Toc402278939)

[5 The log system detailed implementation description 7](#_Toc402278940)

[5.1 General 7](#_Toc402278941)

[5.2 FB\_Queue 7](#_Toc402278942)

[5.2.1 General 7](#_Toc402278943)

[5.2.2 System overview 7](#_Toc402278944)

[5.2.3 Public Methods 8](#_Toc402278945)

[5.2.4 Constants 9](#_Toc402278946)

[5.3 FB\_BasLog 9](#_Toc402278947)

[5.3.1 General 9](#_Toc402278948)

[5.3.2 Structure of a Base-Log entry 10](#_Toc402278949)

[5.3.3 Public Methods 10](#_Toc402278950)

[5.3.4 Constants 11](#_Toc402278951)

[5.4 FB\_DatLog 11](#_Toc402278952)

[5.4.1 General 11](#_Toc402278953)

[5.4.2 Internals 11](#_Toc402278954)

[5.4.3 Public Methods 12](#_Toc402278955)

[5.4.4 Constants 13](#_Toc402278956)

[5.5 FB\_FileLc 13](#_Toc402278957)

[5.5.1 General 13](#_Toc402278958)

[5.5.2 System overview 13](#_Toc402278959)

[5.5.3 Configuration and Template File 14](#_Toc402278960)

[5.5.4 Public Methods 14](#_Toc402278961)

[5.5.5 Constants 15](#_Toc402278962)

[5.6 Other used system functionalities 17](#_Toc402278963)

[5.6.1 Timestamp 17](#_Toc402278964)

[5.6.2 Task information 17](#_Toc402278965)

# Basics

## Introduction

This document describes the concept and the behaviour of the Vistaprint PLC logger system and is intended for all of the involved software-programmer in the Vistaprint PLC software developing.

## Reference documentation

|  |  |  |
| --- | --- | --- |
| Document | Version | Date |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Version

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Author | Status | Version | Date |
| basic version | kvo | d | 0.1 | 07.10.2014 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Status: **d** = draft, **r** = released

## Abbreviations, definitions, glossary

|  |  |
| --- | --- |
| Abbreviation | Designation |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Starting position

The idea is to implement a basic log system which is easy to use and work stand-alone in the PLC software.

The following requirement/features should be possible:

* Configurable over text file or properties
* Extensible for different log application cases

Producer (application task/controller):

* Mutex save (multi-tasking system)
* Independent log entry and file handling (no timing problems or waits)
* Save buffers as retain (never lose any entry even if a system restart occurs)

Consumer/clients

* New file after a time, max number of entries per file or max size of file
* Definable Max count of files in log folder, file stored with a configured period or max total size of all log files in folder
* Option actual log file name stay the same (archive the active log file if it exceeds one of the defined limits to a configurable file name with date and time at the end added)
* Multiple clients for the same log information’s
* Different file formats (csv, xml, html) solved over free format definition or specific clients
* Different client types (e.g. file writer or online watch clients( possible over serial, tcp or other protocols))

Because this system should grow with the application cases the first version will include not all of the features above.

## Short summary of the concept idea

The application task or controller (log writer) subscribes the usage of a logger to the logger system over a name (constant). As soon as a consumer (e.g. csv-file-client) subscribes at the same logger system with the same name a buffer is allocated. The base log system adds a fixed structure to the concerning queue. The application writes over a specific log controller (e.g. DatLog (log a defined structure) or directly to the base log system). The consumer/client reads out the configured log type information’s from the queues and writes this information well formatted to a file, a bus interface or another implemented interface.

# Best practice

## Define/use state log

The state log entry is already predefined in the BaseCtrl FUB so every controller inherits it from there.

### Usage with base controller to write messages

In the base controller in the FB\_INIT the logger entry is switched on per default (“CIf.SwCfg.bolEnStateLogger := TRUE;”). Under me. stStateLogger : T\_StateLogger; a new structure is defined.

TYPE T\_StateLogger :

STRUCT

pfbBaseLog : POINTER TO FB\_BasLog; // should be set from outside over property

uinStatus\_Handle : UINT; // =0 handle ok or not allready requested

pHandle : POINTER TO T\_Bl\_LibHandleRow; // pointer to base logger system

uinStatusWrite : UDINT; // feedback if write was successfully

bolDoLog : BOOL; // state change detected or application cmd to log infos

strText1: STRING(BasLog.cMAX\_CHAR\_TEXT1); // set in CycleEndDuty

// free usable for application

strText2: STRING(BasLog.cMAX\_CHAR\_TEXT2); // free usable from application task for additional

debug information’s

dinPar3: DINT; // free usable from application task for additional debug information’s

dinPar4: DINT; // free usable from application task for additional debug information’s

END\_STRUCT

END\_TYPE

The access point to the log system has to be defined by initialize the new property “BaseLog” at start up with the address of the central base log instance (pCtrlInst[uinIdx]^.BaseLog := pBasLogFB;). For this a new function called setAllCtrlBasLogFB() to initialize all controllers is implemented. Easier is to use the new initAllCtrl() function than the definition of the alarm manager, log system and the CtrlInit is done in one loop. This function has to be called at start up after all controllers are initialized.

The entries of state/sub state change information are logged automatically in the CycleEndDuty. For the application a string and two values are free to use for additional information. By setting the “me.StateLogger.bolDoLog:=TRUE” a log entry can forced from the application for additional log entries.

Summary need for application automatic log entries:

1. MyController.BaseLog := adr(g\_fbBasLog) (set over global function)
2. CIf.SwCfg.bolEnStateLogger := TRUE (per default set)

### File write

Copy from … BaseConcept\Document\Logger\StateLogCsv.cfg the example configuration file to a available PLC folder. In the BasePLC sample a FB (FB\_FileLc\_StateLogger) is already defined with an example of the needed sequence. Copy this FB to your application and change the “In\_strCfgFileName” and the “In\_strPath” to the before copied configuration file name/folder. Now call the FB at start up to initialize until “Out\_bolInitDone”.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Log system - init file client's - takes more than one cycle!

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// state logger file client: init -> readout config file, allocate memory -> cyclic read out state logs and write to csv file

fbFileLc\_StateLogger(In\_p\_fbBasLog:=ADR(g\_fbBasLog),In\_p\_fbFileLC:=ADR(g\_fbFileLc));

IF (fbFileLc\_StateLogger.Out\_bolInitDone=TRUE) OR (g\_bolLogEnableState = FALSE) THEN

g\_uinStartProzess := g\_uinStartProzess + 1; // For StartUp order -> next step!

END\_IF

Now add the cyclic call in the cyclic part of a low priority task. Add a failure detection to your application to recognize problems with the logger if needed.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// file client call for state logger

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

fbFileLc\_StateLogger();

IF (fbFileLc\_StateLogger.Out\_udiStatus<>0) THEN // state log file client error

;

//fbAlm\_Man.setAlarm(p\_fbMAIN^.CtrlCIf^.Info.intCtrlId,E\_ALM\_MAIN.ViperLogFileClient,UDINT\_TO\_DINT(fbFileClient\_ViperLog.Out\_udiStatus),ADR(pTmpSIf\_MAIN^.Alm),E\_AlmState.RST,p\_fbMAIN^.CtrlCIf^.Job.eMe\_ActivAlmClass);

END\_IF

# The log system detailed implementation description

## General

## FB\_Queue

### General

The FB\_Queue provides methods to create initialize and manage a queue for every consumer and task class. A queue is a FIFO-Buffer (First in First out) with configured length. As basic system functionality the FB\_MemRingBuffer from the Beckhoff library Utilities is used.

*Beckhoff help: The FB\_MemRingBuffer function block allows data records of varying lengths to be written into a ring buffer, or for data records that have previously been written there to be removed from the ring buffer. The buffer operates as a FIFO; data records are read from it in the same sequence in which they were first written into the ring buffer. This means that the oldest entries are the first ones that are read. The buffer storage is provided to the function block via the input variables.*

The needed storage buffer size is free definable and is created over the Beckhoff operator \_\_New.

*Beckhoff help: The operator \_\_NEW allocates memory for function block instances or arrays of standard data types. The operator returns a suitably typed pointer to the object.*

With this system functions is the creation of the queues and information storages very flexible and has only the limit of the available dynamic memory. This system has no retain buffer. This means after a power failure or a restart the last not already saved in file log message could be lost. The great advantage is that in this way only the registered writer or reader takes system resources and only a minimum of static not configurable infrastructure is taken from the PLC.

For tests and as example a FUB FB\_QueueTest\_Sender and FB\_QueueTest\_Receiver is also implemented and placed in folder Queue.

### System overview

Queue System  
“q1“

### Public Methods

#### M\_Init

This method should be called one time after start-up of the system. The method allocates an over name defined queue system. Every sender calls the get handle and per caller task class a queue will be allocated automatically.

Interface:

* In\_strQueueName: STRING(Queue.cMAX\_NAME\_LENGTH); // name of the queue system
* In\_uinNrOfElement: UINT; // number of queue elements
* In\_udiSizeOfElement: UDINT; // size of an element
* In\_uinMode:UINT; // 0: no overwrite of buffer, 1: ring buffer overwrite oldest entry
* Out\_pHandle: UDINT; // handle for every task class sender/receiver
* Out\_uinStatus: UINT; // error -> 0:ok

#### M\_GetHandle

Every sender or receiver of log messages must call this method to get a valid access handle for the required log system. Such a call allocates automatically per task class a queue, if it is not already done before.

Interface:

* In\_strQueueName: STRING(Queue.cMAX\_NAME\_LENGTH); // name of the queue system
* Out\_pQueueHandle : POINTER TO BYTE;

#### M\_GetHandle\_TC

This method gives back a valid access handle for the required log system and allocates automatically for the forced task class a queue, if it is not already done before. This function is used for internal behaviour of base log.

Interface:

* In\_strQueueName: STRING(Queue.cMAX\_NAME\_LENGTH); // name of the queue system
* In\_bytTaskIndex: BYTE; // calling task index
* Out\_pQueueHandle : POINTER TO BYTE;

#### M\_Write

Write an element (Variable, Struct, …) based on the before defined M\_Init call dimensions.

Interface:

* In\_pQueueHandle : POINTER TO BYTE; // handle from M\_GetHandle()
* In\_pData: POINTER TO BYTE; // source address of the write element

#### M\_Read

Reads and delete the oldest element from the queue.

Interface:

* In\_pQueueHandle : POINTER TO BYTE; // handle from M\_GetHandle()
* In\_pData: POINTER TO BYTE; // destination address of the read element

#### M\_ReadReference

This method reads the oldest element from the queue.

Interface:

* In\_pQueueHandle : POINTER TO BYTE; // handle from M\_GetHandle()
* Out\_pData: POINTER TO BYTE; // address of the read element

#### M\_Clear

Clear specific queue system.

Interface:

* In\_pQueueHandle : POINTER TO BYTE; // handle from M\_GetHandle()

### Constants

#### Error numbers

cERR\_QUEUE\_FULL : UINT := 1;

cERR\_QUEUE\_EMPTY : UINT := 2;

cERR\_QUEUE\_NAME : UINT := 50000; // not valid or empty queue name

cERR\_NR\_OF\_ELEMENTS : UINT := 50001; // number of elements =0

cERR\_SIZE\_OF\_ELEMENTS: UINT := 50002; // element size =0

cERR\_MEM\_ALLOC : UINT := 50003; // error allocate memory

cERR\_QUEUE\_DOES\_NOT\_EXIST : UINT := 50004; // queue does not exist

cERR\_Q\_INFO\_DOES\_NOT\_EXIST : UINT := 50005; // queue info does not exist

cERR\_DATA\_INVALID : UINT := 50006; // pointer to data not valid

cERR\_HANDLE\_INVALID : UINT := 50007; // handle not ok

cERR\_TC\_INVALID : UINT := 50008; // task class index not ok

cERR\_CLEAR : UINT := 50009; // clear of queue not ok

**Limits**

cMAX\_NAME\_LENGTH : UINT := 80; // maximal string length of queue name

cMAX\_NR\_OF\_TASKS: USINT := 6; // for the moment enough

## FB\_BasLog

### General

The BasLog provides the basic log functionality based on Queue system for the further special loggers. The available methods allow making standardized entries in the queue and reading out from there. Every entry has a set of parameters and gets a time stamp. For the time stamp is the basic FB\_TimeStamp used. So the user can define if he likes time stamps in UTC or local windows time. At start-up of the system there are two method calls required. To allocate the required queues for the data writer/read call M\_ClientInit(). Over the call M\_GetHandle() you get the required handle for writing and reading to the log system. It is possible to manage several clients. The system manages the entry itself based on the defined mask. The bit coded mask allows filtering the entries. If an entry mask does not match any client mask the entry will be rejected. The queue entry is done based on calling task information. So it is important that the received handle from the call of the M\_GetHandle() is only used in the same task.

For tests and as example a FUB FB\_BasLogTest\_Sender and FB\_BasLogTest\_Client is also implemented and placed in folder Queue.

### Structure of a Base-Log entry

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Data Type** | **Wert** |
| FB\_QueueTest\_Receiver | DATE\_AND\_TIME | Actual date and time (UTC or local time) |
| uinMs | UINT | Actual milliseconds |
| udiLogType | UDINT | Typ of log entry (bit coded) |
| intCtrlID | INT | ID of the controller, which has the entry added |
| strText1 | STRING[AE\_BL\_MAX\_CHAR\_TEXT1] | Information text 1 (free choice) |
| strText2 | STRING[AE\_BL\_MAX\_CHAR\_TEXT2] | Information text 2 (free choice) |
| dinPar1 | DINT | Parameter 1 (free choice, 4 byte max.) |
| dinPar2 | DINT | Parameter 2 (free choice, 4 byte max.) |
| dinPar3 | DINT | Parameter 3 (free choice, 4 byte max.) |
| dinPar4 | DINT | Parameter 4 (free choice, 4 byte max.) |

The Parameters 1 to 4 are declared as DINT, could be used for every data type with max size of 4 Byte. The different types have to be checked in the readout method!

### Public Methods

#### M\_GetHandle

Every sender or receiver of log messages must call this method to get a valid access handle to the log system. Such a call allocates automatically per task class a queue, if it is not already done before.

Interface:

Out\_pHandle : POINTER TO T\_Bl\_LibHandleRow; // handle of log system for write (M\_Log)

Feedback = 0 => successful done, <>0 => error

#### M\_Log

Add base log with actual time stamp to log system.

Interface:

* In\_pHandle : POINTER TO BYTE; // handle of log system
* In\_udiLogType: UDINT; // log type
* In\_intCtrlID: INT; // unique controller ID which has created this entry
* In\_strText1: STRING(BasLog.cMAX\_CHAR\_TEXT1); // information text1
* In\_strText2: STRING(BasLog.cMAX\_CHAR\_TEXT2); // information text2
* In\_dinPar1: DINT; // value parameter 1
* In\_dinPar2: DINT; // value parameter 2
* In\_dinPar3: DINT; // value parameter 3
* In\_dinPar4: DINT; // value parameter 4

Feedback = 0 => successful done, <>0 => error

#### M\_ClientInit

This method should be called one time after start-up of the system for reading client. The method allocates an over name defined queue system.

Interface:

* In\_strClientName: STRING(Queue.cMAX\_NAME\_LENGTH); // name of the client
* In\_udiMask: UDINT; // filter mask which log type should be logged for this client
* In\_uinNrOfElement: UINT; // number of elements in queue
* In\_p\_fbTimeStamp: POINTER TO FB\_TimeStamp;
* In\_bolUseLocalTimeStamp : BOOL; // use local time stamp or UTC
* Out\_pHandle: POINTER TO T\_Queue\_Info; // handle for read
* Out\_uinStatus: UINT; // feedback value 0:ok, <>0 => error

#### M\_Read

Read and remove oldest base log with defined mask from the log system.

Interface:

* In\_pHandle : POINTER TO POINTER TO BYTE; // handle for read
* In\_pData: POINTER TO T\_Bl\_BaseStruc; // Pointer for feedback element

Feedback = 0 => successful done, <>0 => error

#### M\_ChangeMask

This method changes the log mask of a client.

Interface:

* In\_strClientName: STRING(Queue.cMAX\_NAME\_LENGTH); // name of the client
* In\_udiNewMask: UDINT; // new filter mask which log type should be logged for this client

Feedback = 0 => successful done, <>0 => error

### Constants

#### Error numbers

cERR\_TOO\_MANY\_CLIENTS : UINT := 51001; // max number of clients connected

cERR\_SYSTEM\_NOT\_FOUND : UINT := 51002; // the system with this name does not exist

cERR\_INVALID\_LOGTYPE : UINT := 51003; // log type is not correct bit coded

cERR\_NO\_WRITE\_TASK : UINT := 51004; // no client added (no handle from a task)

cERR\_NO\_HANDLE : UINT := 51005; // no handle received => fbGetHandle()

cERR\_NO\_FB\_TIME\_STAMP\_POINTER : UINT := 51006; // no pointer for FB\_TimeStamp

cERR\_NO\_FB\_TIME\_STAMP\_INIT\_NOT\_OK : UINT := 51007; // time stamp value not ready for read out

#### Log types

cLOGTYPE\_ALARM : UDINT:=1; //logtyp alarm

cLOGTYPE\_WARNING : UDINT:=2; //logtyp warning

cLOGTYPE\_INFO : UDINT:=4; //logtyp info

cLOGTYPE\_STATELOG : UDINT:=8; //logtyp state log (for statmachine diagnostic)

cLOGTYPE\_PV\_EVENT : UDINT:=16; //logtyp triggered value log

cLOGTYPE\_PV\_CYCLIC : UDINT:=32; //logtyp cyclic value log

cLOGTYPE\_DATA : UDINT:=64; //logtyp data logger

**Limits**

cMAX\_CLIENTS : USINT := 5; // maximal number of clients

cMAX\_CHAR\_TEXT1: USINT := 80; // maximal number of chars for information text1

cMAX\_CHAR\_TEXT2: USINT := 80; // maximal number of chars for information text2

## FB\_DatLog

### General

The DatLog FB can be used for logging application defined complex data structures. The on this way logged information’s could also be read out from a client (i.e. FileCl). The advantage to the base log system is that the data points will be type casted and the number of structure elements is not fixed. The data log system has to be initialized one time (M\_Init). For the logging task the method M\_GetHandle() has to be called one time.

### Internals

The single data point of the given strut is sent as one base log message. To be sure that one set is together a start and stop record control message is sent.

#### Example of a data record

Initialized with start Ident 100!

stMesswerte.usint1 := 33;

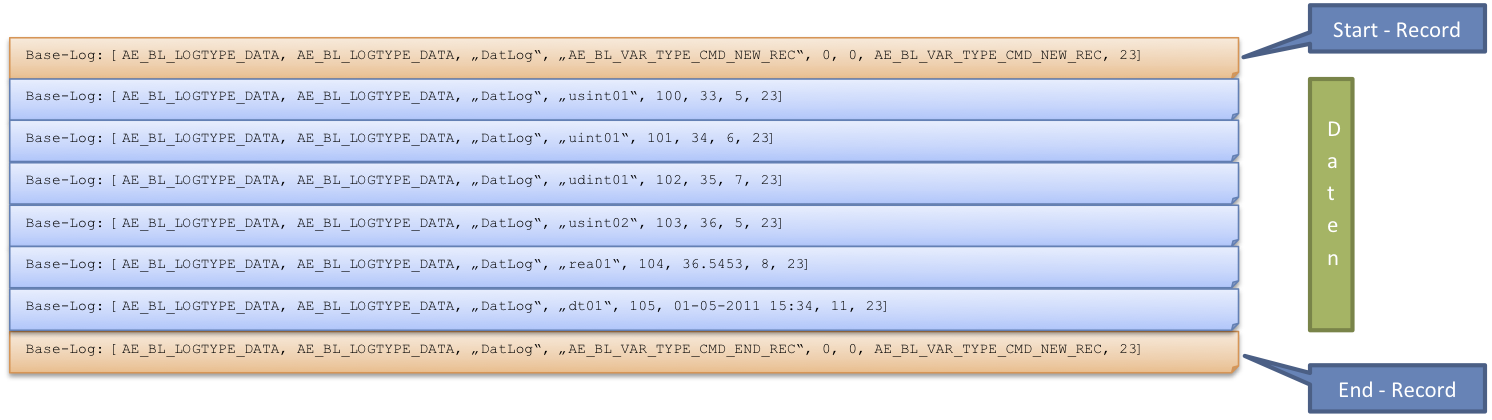
stMesswerte.uint1 := 34;

stMesswerte.udint1 := 35;

stMesswerte.usint2 := 36;

stMesswerte.rea1 := 36.5453;

stMesswerte.dt1 := DT#1973-02-12-04:33;



### Public Methods

#### M\_GetHandle

Every sender or receiver of log messages must call this method to get a valid access handle to the log system. Such a call allocates automatically per task class a queue, if it is not already done before.

Interface:

* In\_strDatLogName: STRING(DatLog.cMAX\_SYS\_NAME\_LENGTH); // name of the client
* In\_p\_fbBasLog: POINTER TO FB\_BasLog; // only one central FB\_BasLog instance allowed per PLC!!!
* Out\_pHandle : POINTER TO BYTE; // handle of log system for write (M\_Log)
* Out\_udiStatus : UDINT; //0 => successful done, <>0 => error

#### M\_Log

Add dat log with actual time stamp to log system.

Interface:

* In\_pHandle : POINTER TO BYTE; // handle of log system for write (M\_Log)
* In\_a\_pData: ARRAY [0..DatLog.cMAX\_NR\_OF\_VAR] OF POINTER TO BYTE; // address of structure elements
* Out\_udiStatus: UDINT; // 0 => successful done, <>0 => error

#### M\_Init

This method should be called one time after start-up of the system for reading client. The method allocates an over name defined log system.

Interface:

* In\_bolExecute : BOOL;
* In\_strDatLogName: STRING(DatLog.cMAX\_SYS\_NAME\_LENGTH); // name of the client
* In\_a\_strDataDefinition: ARRAY[0..DatLog.cMAX\_NR\_OF\_VAR] OF STRING(DatLog.cMAX\_VAR\_NAME\_LEN); // complete VarName: p.e.:MAIN.INIT\_TASK.VARINT
* In\_uinStartId: UINT; // Start ident for item index -> cfg file config has to match with this number!
* In\_Port : T\_AmsPort; // on plc the first runtime system is the 801, 811 ...
* Out\_udiStatus: UDINT; // 0 => successful done, <>0 => error
* Out\_bolDone: BOOL; // Done

### Constants

#### Error numbers

cERR\_MEM\_ALLOC : UINT := 53001; // error on allocating memory

cERR\_NO\_HANDLE : UINT := 53002; // no handle received => fbGetHandle()

cERR\_NO\_DATE : UINT := 53003; // no valid data pointer received

cERR\_INIT\_FAILED : UINT := 53006; //

cERR\_NO\_ITEMS : UINT := 53004; //

cERR\_UNSUPPORTED\_TYPE : UINT := 53005; // the system support this datatyp not

cERR\_UNSUPPORTED\_TYPE\_STRING\_TOO\_LONG : UINT := 53006; // defined string in base log shorter then log string

**Limits**

cMAX\_SYS\_NAME\_LENGTH : USINT := 30; //max name length of system data log name

cMAX\_VAR\_NAME\_LEN : USINT := 50; // max len of variable name

cMAX\_NR\_OF\_VAR : UINT := 20; // max number of variables definable to log

## FB\_FileLc

### General

The file client function block subscribes itself in the base log system to read out configured log messages. He interprets the messages and writes it in the template part of the configuration file defined format to a file. The file name, extension, write cycle and the all other necessary parameters are defined in a configuration file. So the user can free define if he wish to have a \*.csv, \*.html or xml file.

The file client can at the moment the following standard log types interpret:

* Base - Log Einträge (For State-, Debug- and/or Tracelogs)
* Data - Log Records (FB\_DatLog)
* Open - Log Records (User defined records)

The records are data sets of base log messages with a start and end tag marked. For details take a look in the chapter oft he data log.

To be sure that only one client initialization is active at the same time and a unique client name is used, a central file client structure is required. It is recommended the use only one instance per application. The writing and reading of files need a lot of ressources so ist important to call the file client in a fast as possibel task with low priority and no requirement for real time.

### System overview

PV-Log: [Var1, Var2, Var3, Var4]

BaseLogSystem

Base-Log: [LogType, CtrlId, String1, String2, Par1..4]

Data-Log: [value1, value2, value3]

The FileClient notify itself in BaseLog system as client.

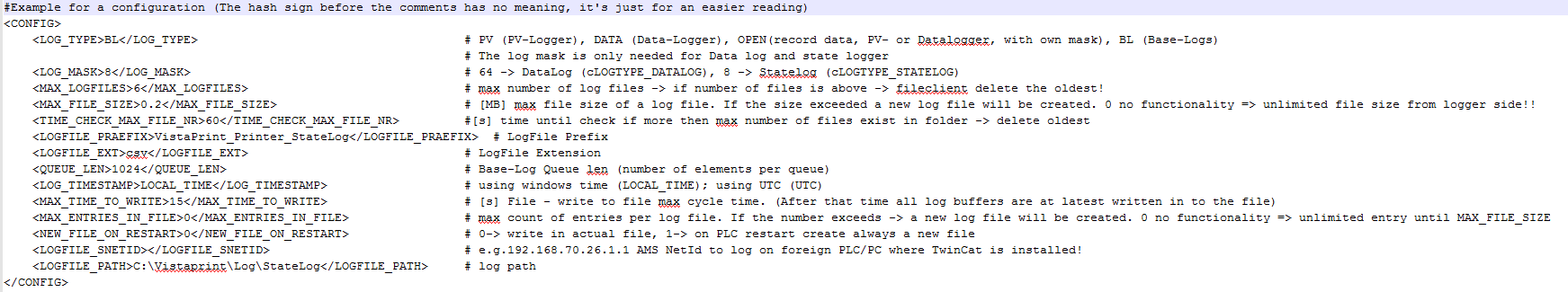
Configuratio/template file:  
<CONFIG>  
</CONFIG>

<TEMPLATE>  
</TEMPLATE>

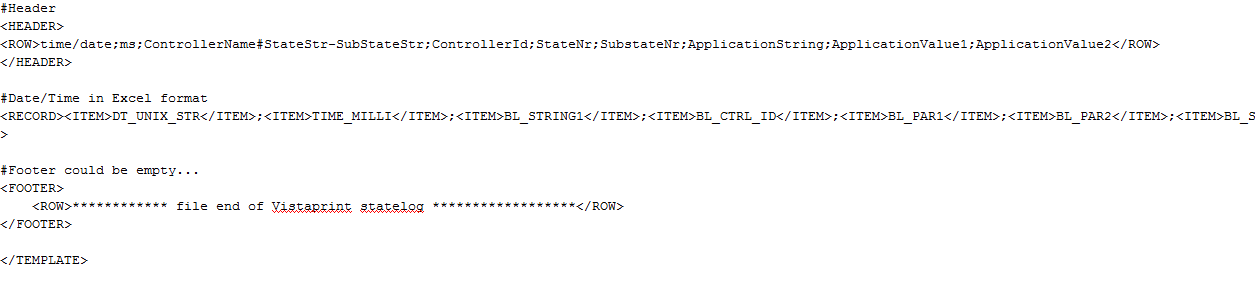
LogFiles

### Configuration and Template File

In this file are in the first part the parameters of the file client defined:



In the second part of the file the log output file format and value order is defined:



Under the logger information folder some examples should be available. Behind every parameter a comment should make clear the possible values and the behaviour.

### Public Methods

#### M\_Init

Initialization of file client based on the information read out from the configuration file. Because of the usage of the Beckhoff file io FUB’s it needs some PLC task cycles to read out. A cyclic call of the M\_Init method is required until Out\_udiStatus<>0 or Out\_bolDone=TRUE. The M\_Init() method subscribes the client as log message reader to the log system.

Interface:

* In\_strCfgFileName: STRING(FileLc.cFILENAME\_LENGTH); // name of the config/template file
* In\_strPath: STRING(FileLc.cPATH\_LENGTH); // path of the config/template file
* In\_p\_fbBasLog: POINTER TO FB\_BasLog; // only one central FB\_BasLog instance allowed per PLC!!!
* In\_bolExecute : BOOL;
* In\_p\_uinFLcIdx : POINTER TO UINT; // actual file client id
* Out\_pHandle: POINTER TO T\_FileLc\_SysInfo; // handle for cyclic call
* Out\_udiStatus: UDINT; // 0 => successful done, <>0 => error
* Out\_bolDone: BOOL; // Done

#### M\_Cyclic

The Cyclic call of the file client writes in required time periods or if buffer is full the configured log information to the log file. Sub function blocks for file handling (FB\_FileHdl) and folder supervision (FB\_FolderHdl) are also called in this cycle.

Interface:

* In\_pHandle: POINTER TO T\_FileLc\_SysInfo; // handle for cyclic call

### Constants

#### Error numbers

cERR\_NBR\_OF\_FC\_EXEEDED : UINT := 50400; (\*Max number of file clients reached\*)

cERR\_FILENAME\_MISSING : UINT := 50401; (\*Filename missing\*)

cERR\_PATH\_MISSING : UINT := 50402; (\*Path definition missing\*)

cERR\_CONFIG\_TAG : UINT := 50403; (\*Error in <CONFIG></CONFIG> tag\*)

cERR\_TEMPLATE\_TAG : UINT := 50404; (\*Error in <TEMPLATE></TEMPLATE> tag\*)

cERR\_INVALID\_LOGTYP : UINT := 50405; (\*Unknown or missing log type -> template/config - file\*)

cERR\_HEADER\_TAG : UINT := 50406; (\*Error in <HEADER></HEADER> tag\*)

cERR\_RECORD\_TAG : UINT := 50407; (\*Error in <RECORD></RECORD> tag\*)

cERR\_ROW\_TAG : UINT := 50408; (\*Error in <ROW></ROW> tag\*)

cERR\_ITEM\_TAG : UINT := 50409; (\*Error in <ITEM></ITEM> tag\*)

cERR\_FOOTER\_TAG : UINT := 50410; (\*Error in <FOOTER></FOOTER> tag\*)

cERR\_HANDLE : UINT := 50411; (\*no handle reached\*)

cERR\_INVALID\_DEVICE : UINT := 50412; (\*File destination not correct\*)

cERR\_ALLOC\_BUFFER\_FILE\_CONFIG : UINT := 50413; (\*No memory available for config read out\*)

cERR\_ALLOC\_MEMORY : UINT := 50414; (\*No memory available\*)

cERR\_NO\_BASELOG : UINT := 50415; (\*BaseLog pointer invalid\*)

cERR\_NO\_FLCIDX : UINT := 50416; (\*INIT not possibel no valid act client nr index reference / pointer invalid\*)

**Limits**

cPRE\_POST\_TEXTLEN : UINT := 256; // max text len for pre and post text of a value

cMAX\_RECORDS\_IN\_FILE\_BUF : UINT := 4; (\*factor for calculation of file buffer\*)

cMAX\_NBR\_OF\_FILECLIENTS : UINT := 10; (\*Max number of file clients\*)

cFILENAME\_LENGTH : UINT := 260; // config file name len

cPATH\_LENGTH : UINT := 260; // folderpath string len

Attention for using UDINT data types! At the moment only the DINT range is supported!

**Config/Template**

**//<ITEM>\*\*\*</ITEM> predefined items of base log**

cITEM\_UNIX\_DATE\_TIME\_STR : STRING[80] := 'DT\_UNIX\_STR'; (\*Item UNIX Datum/Zeit als String\*)

cITEM\_UNIX\_DATE\_TIME : STRING[80] := 'DT\_UNIX'; (\*Item UNIX Datum/Zeit als Integer\*)

cITEM\_TIME\_MILLI : STRING[80] := 'TIME\_MILLI'; (\*Item Millisekunden\*)

cITEM\_EXCEL\_DATE\_TIME : STRING[80] := 'DT\_EXCEL' (\*Item date/time in Excel format. Days as real since

1.1.1970\*)

cITEM\_DATE\_SLASH\_TIME\_COLON : STRING[80] := 'DATE\_SLASH\_TIME\_COLON'; //english date format 2014/09/23 07:10:24

cITEM\_BL\_LOGTYP : STRING[80] := 'BL\_LOGTYP'; (\*Base Log Item LogTyp\*)

cITEM\_BL\_CTRL\_ID : STRING[80] := 'BL\_CTRL\_ID'; (\*Base Log Item Controller Id\*)

cITEM\_BL\_STRING1 : STRING[80] := 'BL\_STRING1'; (\*Base Log Item STRING 1\*)

cITEM\_BL\_STRING2 : STRING[80] := 'BL\_STRING2'; (\*Base Log Item STRING 2\*)

cITEM\_BL\_PAR1 : STRING[80] := 'BL\_PAR1'; (\*Base Log Item PAR1 \*)

cITEM\_BL\_PAR2 : STRING[80] := 'BL\_PAR2'; (\*Base Log Item PAR2\*)

cITEM\_BL\_PAR3 : STRING[80] := 'BL\_PAR3'; (\*Base Log Item PAR3\*)

cITEM\_BL\_PAR4 : STRING[80] := 'BL\_PAR4'; (\*Base Log Item PAR4\*)

**// <LOG\_TYPE>\*\*\*\*</LOG\_TYPE> predefined values**

cLOGTYP\_BL\_STR : STRING[80] := 'BL'; (\*BaseLog\*)

cLOGTYP\_PV\_STR : STRING[80] := 'PV'; (\*PVLog\*)

cLOGTYP\_DATA\_STR : STRING[80] := 'DATA'; (\*DataLog\*)

cLOGTYP\_OPEN\_STR : STRING[80] := 'OPEN'; (\*General (Attention-> Mask has to be defined)\*)

## Other used system functionalities

### Timestamp

In the log system the FB\_TimeStamp (FB\_TimeStamp\_Description\_V0.1.docx) is used. Because the local windows time is not immediately and exact accessible at every PLC cycle.

### Task information

The function GETCURTASKINDEXEX finds the task index of the task from which it is called.

**GETCURTASKINDEXEX** : Returns the current task index of the calling task. The following return value are possible:  
-1 : Function is called by Windows context.  
0 : Function is called by realtime context but not from a cyclic PLC task.  
1 bis n : Function is called by a cyclic PLC task. The return value is the task index.