LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO- E1200225-v3 | Advanced LIGO | 4/20/2012

Coding Standard for TwinCAT Slow Controls Software

Daniel Sigg

Distribution of this document:
LIGO Scientific Collaboration

This is an internal working note
of the LIGO Laboratory.

California Institute of Technology
LIGO Project – MS 18-34
1200 E. California Blvd.
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project – NW22-295
185 Albany St
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
P.O. Box 159
Richland WA 99352
Phone 509-372-8106
Fax 509-372-8137

LIGO Livingston Observatory
P.O. Box 940
Livingston, LA 70754
Phone 225-686-3100
Fax 225-686-7189

http://www.ligo.caltech.edu/

1 Introduction

The purpose of this document is to facilitate a single coding standard among the slow controls software written for the TwinCAT system. TwinCAT contains an embedded IEC 61131-3 software PLC which is the main focus here. The document gives guidance how to build a reusable programming structure, how to name objects like variable, structures and function blocks, and how to document a library module.

1.1 Programming Languages

The IEC 61131-3 programming standard supports 5 different languages: structured text (ST), function block diagram (FBD), ladder diagram (LD), instruction list (IL) and sequential function chart (SFC). TwinCAT 3 also supports C/C++ and Matlab/Simulink. For the advanced LIGO slow control systems only structured text shall be used with TwinCAT 2.11. For TwinCAT 3 advanced LIGO also supports C/C++ for integrating already written modules.

Programming language	Description	TwinCAT version
Structured Text	One of the IEC 61131-3 programming languages, Pascal like	2.11 and 3
C/C++	For integrating previously written modules	3

Table 1: Supported languages.

1.2 Project Directories

The project directories on a front-end or development machine are organized in a development area under version control and a target area where the run-times reside.

Items	Path	Owner
TwinCAT	C:\TwinCAT	Beckhoff
Target	C:\Target	Run-time
Development	C:\SlowControls	Subversion

1.2.1 Target Area

The target area contains the files associated with a specific run-time. In particular, the TwinCAT libraries, the TwinCAT system configuration files, the TwinCAT PLC programs, and the EPICS interface files. The files associated with a specific run-time are copied to the target directory using an installation script.

Items	**Path**	
Target Area	C:\Target	
TwinCAT related files	C:\Target\TwinCAT	
TwinCAT boot files	C:\Target\TwinCAT\Boot	
TwinCAT system configuration	C:\Target\TwinCAT\Configuration	
TwinCAT PLC 1 code	C:\Target\TwinCAT\PLC1	
TwinCAT PLC 2 code	C:\Target\TwinCAT\PLC2	
TwinCAT PLC 3 code	C:\Target\TwinCAT\PLC3	
TwinCAT PLC 4 code	C:\Target\TwinCAT\PLC4	
TwinCAT libraries (no subdirectories)	C:\Target\TwinCAT\Library	
EPICS related files	C:\Target\EPICS	
EPICS boot files	C:\Target\EPICS\Boot	
EPICS database files	C:\Target\EPICS\Database	
EPICS screen files	C:\Target\EPICS\Screen	

1.3 Project Archive

All project files are stored in a subversion (SVN) archive on redoubt.ligo-wa.caltech.edu.

Item	**Link**	**Type**
Server	redoubt.ligo-wa.caltech.edu	web
Archive	/slowcontrols	web
Full path	https://redoubt.ligo-wa.caltech.edu/svn/slowcontrols/trunk	checkout

Table 2: Subversion archive.

1.3.1 Organization

The slow controls archive contains the folder TwinCAT for storing all files related to TwinCAT. There are currently two sub folders TwinCAT\Library for storing libraries and TwinCAT\target for the storing project files and the system configuration associated with single real-time computer. There are up to 4 PLCs allowed in TwinCAT 2.11. The individual PLC projects are stored in subdirectories PCL1, PLC2, PLC3 and PLC4.

Items	**Path**	
System documents	SlowControls\Documents	
Network documents	SlowControls\Documents\Network	
TwinCAT files	SlowControls\TwinCAT	
TwinCAT documents	SlowControls\TwinCAT\Documents	
TwinCAT coding standard	SlowControls\TwinCAT\Documents\CodingStandard	
TwinCAT target files	SlowControls\TwinCAT\Target	
Individual TwinCAT target	SlowControls\TwinCAT\Target\H1ECATC1	
Individual TwinCAT PCLs	SlowControls\TwinCAT\Target\H1ECATC1\PLC1	
…	…	
TwinCAT library files	SlowControls\TwinCAT\Library	
Individual TwinCAT library	SlowControls\TwinCAT\Library\CommonModeServo	
…	…	
EPICS related files	SlowControls\EPICS	
EPICS target files	SlowControls\EPICS\Target	
Individual EPICS target	SlowControls\EPICS\Target\H1ECATC1	
…	…	
Modbus related files	SlowControls\Modbus	
Modbus target files	SlowControls\Modbus\Target	
Individual Modbus target	SlowControls\Modbus\Target\H1ModbusC1	
…	…	

Table 3: Organization of the archive.

1.3.2 Version Numbers

The production code is managed by subversion release numbers.

When significant changes to a library are made that require supporting both the old and new versions, a new library project has to be created. If the original library was called TimingMasterFanout then new version would be called TimingMasterFanoutV2.

1.4 Cycle Time

An IEC 61131-3 system consists of system task and at least one programmable logic controller (PLC). The system task is responsible for interfacing the hardware and starting the PLC tasks. The field bus of choice in advance LIGO is EtherCAT. The system task transfers data between a shared memory region and hardware at a fixed cycle time. TwinCAT 2.11 supports up to four different update rates. For advanced LIGO the standard update rate is 10 ms. For a limited number of channels a faster update rate of 1 ms is supported.

Task	Description	Rate
Standard	All non time critical software and supervisory tasks	10 ms
Fast	Time critical functions such as RS422 support at 115kbaud	1 ms

Table 4: Supported update rates.

The tasks with the fast update rate are running at a higher priority (lower number).

1.5 Data Tags (Channels)

1.5.1 Input/Output Convention

From the perspective of the TwinCAT program and configuration input channels refer to inputs from the EtherCAT terminals, e.g., analog-to-digital converters and binary inputs, whereas output channels refer to outputs to the EtherCAT terminals, e.g., digital-to-analog converters and binary outputs. The same is true for user inputs which are inputs into TwinCAT and readbacks which are outputs from TwinCAT.

1.5.2 Interface Variables

All external tags (channels) are declared PERSISTENT and are retained upon power failure and loading a new code. Any initialization that is required, when the PLC is started or when a new version is loaded, needs to be dealt with in software.

1.6 OPC Interface

We are using the TwinCAT OPC comments denoted by (*~ ... *) to make global variables accessible to the OPC server. Variable names in TwinCAT are translated one-to-one into OPC tag names, which in turn are translated into EPICS channels using a conversion rule. OPC properties are used to describe additional information such as limits, precision and state names. These OPC properties are translated into corresponding EPICS database fields.

2 Program Organization

The development blocks for the advanced LIGO slow controls software are individual libraries. Each of the basic libraries is tailored to control a single electronics chassis or controller.

A typically library consists of

- one or more type describing the hardware inputs,
- one or more type describing the hardware outputs,
- a type describing the user interface channels or tags (input and output),
- one or more function blocks containing the run-time code, and
- a set of visual templates that can be used for diagnostics.

The main program then consists of a global variable list and a series of function block calls.

2.1 Library

This section gives an example of the structures and the function block defined for the LowNoiseVco library.

2.1.1 Hardware Input Structure

```
TYPE LowNoiseVcoInStruct :
STRUCT
       PowerOk:             BOOL;  (* Voltage monitor readback *)
       TuneMon:             INT;   (* Monitor for the frequency offset  *)
       ReferenceMon:        INT;   (* RF power at the reference input *)
       DividerMon:          INT;   (* RF power at the divider input *)
       OutputMon:           INT;   (* RF power at the output amp *)
       ReferenceTemp:       INT;   (* Temperature of the reference RF detector *)
       DividerTemp:         INT;   (* Temperature of the divider RF detector *)
       OutputTemp:          INT;   (* Temperature of the output RF detector *)
       Excitation:          BOOL;  (* Monitors the excitation input enable *)
       Frequency:           LREAL; (* Measured frequency *)
       FrequencyLive:       BOOL;  (* Keep alive for frequency measurement *)
END_STRUCT
END_TYPE;
```

2.1.2 Hardware Output Structure

```
TYPE LowNoiseVcoOutStruct :
STRUCT
       TuneOfs:             INT;   (* Setpoint for the frequency offset  *)
       ExcitationEn:        BOOL;  (* Enables the excitation input *)
END_STRUCT
END_TYPE;
```

2.1.3 Interface Structure

All elements of an interface structure are getting exported with read and write permission. To prevent output tags from showing an invalid value each output parameter has to overwritten at each cycle. Output parameters in the interface structure should never be read.

```
TYPE LowNoiseVcoStruct :
STRUCT
       (* error handling *)
       Error:              BOOL;  (* Error flag *)
       ErrorCode:          DWORD; (* Error code *)
       ErrorMessage:       STRING;(* Error message *)
       (* output tags *)
       PowerOk:            BOOL;  (* Voltage monitor readback *)
       TuneMon:            LREAL; (* Monitor for the frequency offset in V *)
       ReferenceMon:       LREAL; (* RF power at the reference input in dBm *)
       DividerMon:         LREAL; (* RF power at the divider  input in dBm *)
       OutputMon:          LREAL; (* RF power after the output amplifier dBm *)
       ReferenceTemp:      LREAL; (* Temperature of the reference RF detector *)
       DividerTemp:        LREAL; (* Temperature of the divider RF detector *)
       OutputTemp:         LREAL; (* Temperature of the output RF detector in C *)
       ExcitationSwitch:   BOOL;  (* Monitor the excitation input enable *)
       Frequency:          LREAL; (* Frequency of the VCO output *)
       FrequncyServoFault: BOOL;  (* Indicates a fault in the frequency servo *)
       (* input tags *)
       TuneOfs:            LREAL; (* Setpoint for the frequency offset in V *)
       ExcitationEn:       BOOL;  (* Enables the excitation input *)
       FrequencySet:       LREAL; (* Setpoint for the VCO frequency output *)
       FrequencyServoEn:   BOOL;  (* Enables the frequency PID *)
END_STRUCT
END_TYPE;
```

2.1.4 Error Handling

Each main function block needs to provide error handling using three variables defined in the interface structure: Error, ErrorCode and ErrorMessage. The error flag is set true to indicate an error condition. The error code is a bit encoded value listing the error conditions with zero indicating no error. The error code number can be used to flag multiple errors by setting corresponding bits. Error conditions are described in the documentation associated with the library. The error message is a human readable string describing the error condition. It can contain up to 80 characters. When required the definition STRING(255) can be used to support up to 255 characters. If multiple errors are flagged, the error message needs to reflect this. A simple library without error conditions needs to set the error flag to false, the error code to zero and the error message to an empty string.

2.1.5 Function Block

A function block has to declare input and output variables. In the simplest case the input parameter is the hardware input structure, the hardware output structure is the output parameter and the interface structure is the in/out parameter.

```
FUNCTION_BLOCK LowNoiseVcoFB
VAR_INPUT
       LowNoiseVcoIn:      LowNoiseVcoInStruct;      (* Input structure *)
END_VAR
VAR_OUTPUT
       LowNoiseVcoOut:     LowNoiseVcoOutStruct;     (* Output structure *)
END_VAR
VAR_IN_OUT
       LowNoiseVco:        LowNoiseVcoStruct;        (* Interface structure *)
END_VAR
…
```

2.1.6 Initialization

Since all interface variables are persistent, they will not lose their value between reboots—or when a newly recompiled program is loaded. Using the example below, it is possible to execute proper initialization code. Values of hardware channels are not retained and are initialized to their default value when the program is started. If a safe condition is required after a restart, it should be reflected in the default values for the hardware channels as well as in the initialization code of the function block.

```
VAR
       InitBoot:           BOOL := TRUE;
END_VAR
VAR RETAIN
       InitProgram:        BOOL := TRUE;
END_VAR

(* Code *)
IF InitBoot THEN
       InitBoot := FALSE;
       (* Executed everytime the system is restarted or booted *)
END_IF;
IF InitProgram THEN
       InitProgram := FALSE;
       (* Executed everytime the code is recompiled *)
END_IF;
…
```

2.1.7 Visual Screen Templates

Either one or a set of visual screen templates are associated with a library. The top-level screen template should be a representation of the hardware controlled by the library. It should interface the interface structure, and display all its input and output parameters. Input parameters should be modifiable by the user. Since the library only knows abstract data types, the visual screen template shall deploy placeholder variables to represent actual data. For example, the VCO template screen might reference “vco.OutputMon” in the numeric field describing the output RF power. vco is the placeholder parameter that will be replaced with the actual data of type LowNoiseVcoStruct, when the visual template is embedded into a master screen. In most cases the visual template screens should leave their background transparent, so that it can be set by the master screen.

2.2 Global Variables

The global variable for the interface structure is for test purpose only. On a production system the hierarchical type structure outlined in section 3.4 has to be implemented. The interface variables are declared as persistent and are retained between reboots and restarts of the program.

```
VAR_GLOBAL
       LowNoiseVcoTestIn    AT %IB0:     LowNoiseVcoInStruct;       (* Input *)
       LowNoiseVcoTestOut   AT %QB0:     LowNoiseVcoOutStruct;      (* Output *)
END_VAR
VAR_GLOBAL PERSISTENT
       LowNoiseVcoTest:                  LowNoiseVcoStruct;         (* Interface *)
END_VAR
```

2.3 Program

Typically, the main program is simple with single a call to the function block. The program needs to be attached to the standard task, which updates at the 10 ms rate.

```
PROGRAM MAIN
VAR
       LowNoiseVco:        LowNoiseVcoFB;      (* function block for VCO *)
END_VAR

LowNoiseVco (LowNoiseVcoIn := LowNoiseVcoTestIn,
             LowNoiseVcoOut => LowNoiseVcoTestOut,
             LowNoiseVco := LowNoiseVcoTest);
END_PROGRAM;
```

3 Naming Scheme

3.1 Names

Generally, verbose and descriptive names are preferred to short and abbreviated ones. This will make the code more readable and help in maintenance and support. For example, Index is preferred over I and TimingMasterFanout is preferred over Tmfo.

3.1.1 Variable Names

The naming of variables preferably should be unique in all libraries, following the camel case notation: For each variable a meaningful, preferably short, English name should be used, the base name. Always the first letter of a word of the base name is to be written uppercase, the remaining letters lowercase; example: FastGain or InputOffset. Abbreviations are written starting with an uppercase and then all lower case; example: VcoGain or TimingMasterFanout. Pointer variables shall use the suffix **Ptr**, whereas constant variables may use the suffix **Const**.

3.1.2 Type Names

Type names follow the same rule as variable names. A complex type shall incorporate a suffix to denote is derivation: **Enum** for ENUM, **Struct** for STRUCT and **Array** for ARRAY.

Structure members follow the rules of variables.

3.1.3 Function and Method Names

Function and method names follow the same rules as variables but with the suffix **Fun**. Internal helper functions such as conversion routines can also use a lowercase name, so that they look more in line with mathematical notation.

3.1.4 Function Block Names

The names of function blocks follow the same rules as variables but with the suffix **FB**. Interfaces in TwinCAT 3 use the suffix **I**.

3.1.5 Names of Visuals

Visual interfaces have the suffix **Vis**.

3.1.6 Suffix Summary

Element	Description	suffix
Constant	Constant value (optional, may be clear from context)	Const
Pointer	Pointer to a variable	Ptr
ENUM	Enumerated type	Enum
STRUCT	Record type	Struct
ARRAY	Array type	Array
Function	Function or Method declaration	Fun
Function block	Function block declaration	FB
Interface	Abstract function block or interface	I
Visual	Screen interface for diagnostics	Vis

Table 5: Required suffix notation.

3.2 Hardware Channels

Variables that are connected to hardware channels are separated into input variables and output variables. They must be located in the input and output shared memory regions, respectively. A variable describing a list of input channels must have the suffix **In**. The corresponding structure must have the suffix **InStruct**. An output channel list uses the suffix **Out**, whereas the output structure uses **OutStruct**. Channels with different cycle time must be placed into different structures. The above names are for the standard cycle time of 10 ms. Channels that need to be updated at the fast rate need to prepend **Fast** to the above suffixes.

Element	Description	suffix
Input variable	Input variable with standard update rate	In
Output variable	Output variable with standard update rate	Out
Input variable	Input variable with fast update rate	FastIn
Output variable	Output variable with fast update rate	FastOut
Input STRUCT	Input channel structure with standard update rate	InStruct
Output STRUCT	Output channel structure with standard update rate	OutStruct
Input STRUCT	Input channel structure with fast update rate	FastInStruct
Output STRUCT	Output channel structure with fast update rate	FastOutStruct

Table 6: Input and output channel notation.

A code fragment declaring input and output channels in the global variable space:

```
PicoMotorFastIn    AT %IB0100: PicoMotorFastInStruct;
PicoMotorFastOut   AT %QB0200: PicoMotorFastOutStruct;
PicoMotorIn        AT %IB0102: PicoMotorInStruct;
PicoMotorOut       AT %QB0204: PicoMotorOutStruct;
```

3.3 Library Objects

3.3.1 Name Space

Libraries can optionally choose a name space following the variable name notation. This name space is then used to prefix all exported objects. For example: the library TimingMasterFanout has the name space prefix Timing. Within this library TimingSlaveDuoToneStructure, TimingReadSlaveFun and TimingMasterFanoutFB are a valid structure, function and function block, respectively.

Simple libraries that consist of an input structure, an output structure, an interface structure and a function block are not required to choose an explicit name space, but are expected to use the library name as the base for all four objects. Hence, they are defining an implicit name space with the same name as the library name. For example: the library CommonMode may contain the structures CommonModeInStruct, CommonModeOutStruct and CommonModeStruct as well as the function block CommonModeFB.

3.3.2 Folder Names

Program object units (POUs) and data types are organized in folders. These folders are purely organizational and are intended to help grouping items together for easier maintenance. In a library all exported types, functions and function blocks are typically located at the top level. If there are many objects, it may make sense to group them into folders. In any case, internal objects should always be moved into a folder named Internal.

3.4 External Tags

External tags (channels) are organized in a hierarchical structure. Each system defines its own structure. This continues with structures for subsystems that are contained in the system structures.

```
TYPE AlsStruct:
STRUCT
      Vco:        LowNoiseVcoStruct;
      FiberServo: CommonModeStruct;
      LaserServo: CommonModeStruct;
      …
END_STRUCT
END_TYPE;
…
TYPE IscStruct:
STRUCT
      Als:        AlsStruct;
      Asc:        AscStruct;
      Lsc:        LscStruct;
END_STRUCT
END_TYPE;
…
TYPE IfoStruct:
STRUCT
      Isc:        IscStruct;
      Tcs:        TcsStruct;
END_STRUCT
END_TYPE;

VAR_GLOBAL PERSISTENT
      I1:         IfoStruct;  (*~   (OPC : 1 : visible for OPC-Server) *)
END_VAR;
```

This allows for exporting the entire interferometer interface structure at once and it allows for generating tag names automatically while preserving the hierarchical organization.

4 OPC Access and Properties

4.1 OPC Access

The global variable describing the interface structure of the interferometer is made accessible to the OPC server by using the OPC comments. Meaning,

```
H2:          IfoStruct;  (*~   (OPC : 1 : visible for OPC-Server)
                               (OPC_PROP[8610] : h2ecatc1 : server name) *)
```

will make the entire h2 variable with all its sub elements will be visible through the OPC interface. In turn, it can be interfaced to EPICS. Individual tags such as the FastGain of the LaserServo will be available from the OPC server as “H2.Isc.Als.LaserServo.FastGain”. The default EPICS channel name constructed from this tag will then become “H2:Isc-Als_LaserServo_FastGain”. Be aware that IEC 61131-3 names are not case sensitive. The same is true for the corresponding TwinCAT OPC names, whereas EPICS channel names are case sensitive.

4.2 OPC Properties

OPC properties are used to further describe the external tags. These properties are also used to fill in the EPICS database fields. The properties have to be attached to the elements at the end of the hierarchical structure. These are variables with a basic type like INT or LREAL. Due to the program organization most of these variables are defined in libraries through structures. Therefore, the OPC properties are written after the structure elements using the OPC comment structure. For example:

```
TYPE LowNoiseVcoStruct :
STRUCT
       (* output tags *)
       PowerOk:      BOOL;  (*~
                            (OPC_PROP[0005] : 1 : read-only)
                            (OPC_PROP[0101] : Voltage monitor readback : DESC)
                            (OPC_PROP[0106] : OK : ONAM)
                            (OPC_PROP[0107] : OOR : ZNAM) *)
       TuneMon:      LREAL; (*~
                            (OPC_PROP[0005] : 1 : read-only)
                            (OPC_PROP[0101] : Frequency offset monitor : DESC)
                            (OPC_PROP[0100] : V : EGU)
                            (OPC_PROP[0103] : -10 : LOPR)
                            (OPC_PROP[0102] : +10 : HOPR)
                            (OPC_PROP[8500] : 3 : PREC) *)
       ...
END_STRUCT
END_TYPE;
```

Only a small subset of EPICS database fields are supported. In general, fields associated with conversion and calculations are not supported, since all processing should be done within the PLC program. At the present time alarms are also not supported. The following general properties are supported:

Property ID	Description	Record
5	Access control: 1 – read-only, 3- read/write	all
100	EGU: Engineering units	numeric
101	DESC: Description	all
102	HOPR: High operations value	numeric
103	LOPR: Low operation value	numeric
104	DRVH: Maximum instrument range	numeric
105	DRVL: Minimum instrument range	numeric
106	ONAM: Label for closed (one) state	binary
107	ZNAM: Label for open (zero) state	binary
8500	PREC: Display precision	numeric
8510 to 8525	ZRST, ONST, ... FFST: Zero string, one string, ... fifteen string	multi-bit binary
8600	EPICS data type (bi, bo, ai, ao, longin, longout, stringin, stringout, mbbi, mbbo, mbbiDirect, and mbboDirect)	all
8601	Input or output: overwrites the default behavior	all
8602	TSE: Time stamp; default is -2	all
8603	PINI: default 1 for input and 0 for output records	all
8604	DTYP: default is opc; can be overwritten with opcRaw	all
8610	Default OPC server name; default is opc	top level
8700 to 8799	FIELD: Any database field can be specified in the comment string; does not perform any checks; use only when truly desperate	don't use

Table 7: Supported OPC properties.

If a property is specified for a structure, it is used as the default value for all its elements. It can be overwritten by each element, so.

4.3 Automatic Type Support

By default all variables that are read-only will be represented by EPICS input records, whereas all variables that have read/write access will be represented by EPICS output records. This behavior can be overwritten, but there should never be a reason to.

The table below shows the default EPICS type selected for the database depending on the TwinCAT datatype.

Type	Description	
longin/longout	SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, BYTE, WORD, DWORD, LWORD	
bi/bo	BOOL	
mbbi/mbbo	Enumerated data type with 16 or fewer labels	
stringin/stringout	STRING	
ai/ao	REAL, LREAL, any other	

Table 8: Automatic type support.

An enumerated type will be converted into a multi-bit binary record, if there are 16 or fewer labels and if all numeric representations are between 0 and 15. There is no conversion possible. The numeric value of the enum type has to be the same as its EPICS representation, i.e., The zero value will be set to 0, etc. The string values of the multi-bit binary record are automatically set to the labels of the enumerated type.

4.4 Array Variables

Array variables are supported by IEC 61131-3 and can be exported through OPC as well. They will also be accessible through EPICS, but require an extension to the LIGO channel naming convention. For example, if the structure "L1.Io.Wfs1" contains the members:

```
TYPE DemodComplex:
STRUCT
       I:     LREAL;
       Q:     LREAL;
END_STUCT
END_TYPE;

Gain:         ARRAY [1..4] OF LREAL;
Rotation:     ARRAY [1..4,1..4] OF LREAL;
Signal:       ARRAY [1..4] OF DemodComplex;
```

The corresponding OPC and EPICS variables are (with *m* and *n* ranging from 1 to 4):

Type	OPC name	EPICS name
LREAL	L1.Io.Wfs1.Gain[*m*]	L1:Io-Wfs1_Gain[*m*]
LREAL	L1.Io.Wfs1.Rotation[*m*][*n*]	L1:Io-Wfs1_Rotation[*m*][*n*]
LREAL	L1.Io.Wfs1.Signal[*m*].I	L1:Io-Wfs1_Signal[*m*]_I
LREAL	L1.Io.Wfs1.Signal[*m*].Q	L1:Io-Wfs1_Signal[*m*]_Q

Table 9: Array variables with OPC and EPICS.

5 Documentation

A template for documenting a TwinCAT library exists in the DCC, F1200003. It contains the project information, a description of the function blocks as well as detailed listing of the input and output types. Some specialized libraries may require additional information for functions, interfaces or global variables. An example can be found in E1200226.

5.1 Project Information

The following project information is required: title, version, name space, author and a short description.

Field	Description	Mandatory
Title	Name of the library, usually in camel case, e.g., LowNoiseVco	Yes
Version	Library version number, usually 1, 2, etc.	Yes
TwinCAT	Version of TwinCAT for which the library was developed	Yes
Name space	Name space of the library	Yes, if exists
Author	Name of the programmer	Yes
Description	Short description of the purpose of the library	Yes
Error code	Lists the available error codes	Yes

Table 10: Project Information.

5.2 Type Information

Each external type of a library require the following information: name, definition and short description. For a complex type each element should contain a short description as well.

Field	Description	Mandatory
Type name	Name of the type, e.g., LowNoiseVcoStruct	Yes
Definition	Type definition used by the library	Yes
Description	Short description of the purpose of the type	Yes
Elements	For complex types a list of elements	Yes, if exist

Table 11: Type Information.

5.3 Global Variables

Generally, there should be no need for global variables in a library. If they exist, the following information is required: name, type, a possible initialization value and a short description.

Field	Description	Mandatory
Variable name	Name of the global variable	Yes
Type	Type of the global variable	Yes
Initialization	Initialization value of the variable	Yes, if exist
Description	Short description of the purpose of the variable	Yes

Table 12: Global variables.

5.4 Interfaces

In TwinCAT 3 abstract classes are called interfaces. They contain a list of abstract methods. Each interface definition requires name, list of methods and a short description.

Field	Description	Mandatory
Interface name	Name of the type, e.g., LowNoiseVcoStruct	Yes
Methods	List of methods used by the interface	Yes
Arguments	Each method can have a list of arguments	Yes, if exist
Description	Short description of the purpose of the interface	Yes

Table 13: Interfaces.

5.5 Functions

Each function requires the following information: name, return type, list of input parameters, list of output parameters, list of in/out parameters and a short description.

Field	Description	Mandatory
Name	Name of the, e.g., TimingSlaveDuoToneReadFunc	Yes
Return	Return type	Yes
Inputs	List of input parameters	Yes, if exist
Outputs	List of output parameters	Yes, if exist
In/Outs	List of in/out parameters	Yes, if exist
Description	Short description of the purpose of the function or function block	Yes

Table 14: Functions.

5.6 Function Blocks

Each function and function block requires the following information: name, list of input parameters, list of output parameters, list of in/out parameters and a short description. In TwinCAT 3 function block are treated as classes and can extend a base class, inherit from an interface definition and contain methods. If used, the information of all class elements are required.

Field	Description	Mandatory
Name	Name of the function or function block, e.g., LowNoiseVcoFB	Yes
Parent	For classes that extend a parent function block	Yes, if exist
Interfaces	For classes that implement an interface	Yes, if exist
Inputs	List of input parameters	Yes, if exist
Outputs	List of output parameters	Yes, if exist
In/Outs	List of in/out parameters	Yes, if exist
Methods	List of methods used by the function block	Yes, if exist
Description	Short description of the purpose of the function or function block	Yes

Table 15: Function blocks.

5.7 Visuals

Each visual screen element requires the following information: screen snapshot, name, a short description and a list of placeholders. Placeholders are parameters denoted by $paramter_name$ in the visuals that are required to be defined when the visual is embedded. Since the visual of a library usually represents an interface structure, there should be at least one placeholder parameter denoting a variable of this type.

Field	Description	Mandatory
Name	Name of the function or function block, e.g., IscWhiteningVis	Yes
Description	Short description of the purpose of the function or function block	Yes
Placeholder	Parameters used for variable substitution	Yes, if exist

Table 16: Visuals.