ASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

LIGO Laboratory / LIGO Scientific Collaboration

LIGO- E1200225-v1 *advanced LIGO* 2/20/2012

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 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.

1.2.1 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.

Items	Path	
Library files	slowcontrols\TwinCAT\Library	
Target files	slowcontrols\TwinCAT\Target	

Table 3: Organization of the archive.

1.2.2 Version Numbers

The production code is managed by 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 TimingMFO then new version would be called TimingMFOV2.

1.3 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).

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), and
- one or more function blocks containing the run-time code.

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
       (* 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 Function Block

```
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
…
```

2.2 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.

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

2.3 Program

The main program is particular 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 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

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 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:
      Als:        AlsStruct;
      Asc:        AscStruct;
      Lsc:        LscStruct;
STRUCT
END_STRUCT
END_TYPE;
…
TYPE IfoStruct:
STRUCT
      Isc:        IscStruct;
      Tcs:        TcsStruct;
END_STRUCT
END_TYPE;

H2:               IfoStruct;  (*~   (OPC : 1 : visible for OPC-Server) *)
```

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 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.

4.1 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

Table 7: Project Information.

4.2 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

4.3 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

4.4 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

4.5 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

4.6 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