Coding Rules for *OpENer* — Open Source EtherNet/IP^{TM} Adapter Stack

Version 1.0

Alois Zoitl*

2009-10-29

Contents

1 Comments **1**
1.1 Fileheaders 1
1.2 Revision History 2
1.3 Keywords 2

2 Datatypes **2**

3 Naming of Identifiers **2**
3.1 Variables 2
3.2 Prefixes 3
3.3 Constants 3

4 Code Formatting **3**

1 Comments

A sufficient amount of comments has to be written. There are never too many comments, whereas invalid comments are worse than none — thus invalid comments have to be removed from the source code. Comments have to be written in English.

> Comments for function, structure, ... definitions have to follow the conventions of *Doxygen* to allow the automated generation of documentation for the sourcecode.

Comments have to be meaningful, to describe to program and to be up to date.

1.1 Fileheaders

Every source-file must contain a fileheader as follows:

```
/*******************************************************************************
 * Copyright (c) 2009, Rockwell Automation, Inc.
 * All rights reserved.
 *
 * Contributors:
 *     <date>: <author>, <author email> - changes
 ******************************************************************************/
```

Each author needs to explain his changes in the code.

*zoitl@acin.tuwien.ac.at

1.2 Revision History

The revision history has to be done in a style usable by Doxygen. This means that the history is independent of the files, but all classes are documented.

1.3 Keywords

The following Keywords should be used in the source code to mark special comments:

- **TODO:** For comments about possible or needed extensions
- **FIXME:** To be used for comments about potential (or known) bugs

2 Datatypes

The following table contains the definitions of important standard datatypes. This is done to ensure a machine independant defintion of the bit-width of the standard data types. For *OpENer*-development these definitions are in the file: `src/typedefs.h`

defined data type	bit-width / description	used C-datatype
EIP_BYTE	8 bit unsigned	char
EIP_INT8	8 bit signed	signed char
EIP_INT16	16 bit signed	short
EIP_INT32	32 bit signed	long
EIP_UINT8	8 bit unsigned	char
EIP_UINT16	16 bit unsigned	unsigned short
EIP_UINT32	32 bit unsigned	unsigned long
EIP_FLOAT	single precission IEEE float (32 bit)	float
EIP_DFLOAT	double precission IEEE float (64 bit)	double
EIP_BOOL8	byte variable as boolean value	bool

These data types shall only be used when the bit size is important for the correct operation of the code. If not we advice to use the type `int` or `unsigned int` for most variables, as this is the most efficent data type and can lead on some platforms (e.g., ARM) even to smaller code size.

3 Naming of Identifiers

Every identifier has to be named in English. The first character of an identifier must not contain underscores (there are some compiler directives which start with underscores and this could lead to conflicts). Mixed case letters has to be used and the appropriate prefixes have to be inserted where necessary.

3.1 Variables

Variables have to be named self explanatory. The names have to be provided with the appropriate prefixes and they have to start with an uppercase letter. In case of combining prefixes, the use of ranges, arrays, pointer, enumerations, or structures is at first, followed by basic data types or object prefixes. The only exception are loop variables (thereby the use of i, j, k is allowed). Only one variable declaration per line is allowed. Pointer operators at the declaration have to be located in front of the variable (not after the type identifier). If possible initializations have to be done directly at the declaration.

3.2 Prefixes

The following prefixes have to be applied to identifiers:

Type Definitions

S	for structures
E	for enum
T_	for types (e.g. typedef in C++)

Variable Types

a	for arrays
p	for pointers
e	for enumerations
st	for structures

Ranges

m_	for member variables of classes
g_	for global variables
s_	for static variables
pa_	for function parameters

Basic Data Types

c	for characters
b	for booleans
n	for integers
f	for all floating point numbers

Examples

```
struct SCIPObject;
int nNumber;
int *pnNumber = &nNumber;
char cKey;
bool g_bIsInitialized;
float m_fPi = 3.1415;
int anNumbers[10];
```

3.3 Constants

Constants have to be named with block letters (only upper case letters). If a name consists of more words, underscores for separation are allowed. Avoid the using "magic numbers" (e.g. `if (x == 3){...}`). Instead use constants.

4 Code Formatting

In order to have consitent code formating the rules of GNU shall apply. When using Eclipse as development environment this format is alread set as preset. By pressing ¡ctrl¿¡shift¿f the formater will format the code according to these rules.