**C Shape Coding Guidelines**

**Prepared For:**

**SW Development**

# Table of Content

[Table of Content 3](#_Toc491673538)

[Revision history 8](#_Toc491673539)

[1. Introduction 9](#_Toc491673540)

[1.1. Purpose of the document 9](#_Toc491673541)

[1.2. Reference 9](#_Toc491673542)

[2. Naming Convention 10](#_Toc491673543)

[2.1. File Naming 10](#_Toc491673544)

[2.1.1. Name\_File\_001 ([1] Clause 5.4.7 - table 1 - 1h) 10](#_Toc491673545)

[2.1.2. Name\_File\_002 ([1] Clause 5.4.7 - table 1 - 1h) 10](#_Toc491673546)

[2.1.3. Name\_File\_003 10](#_Toc491673547)

[2.1.4. Name\_File\_004 11](#_Toc491673549)

[2.1.5. Name\_File\_005 11](#_Toc491673550)

[2.2. Variable Naming 11](#_Toc491673551)

[2.2.1. Name\_Var\_001 ([1] Clause 5.4.7 - table 1 - 1h) 11](#_Toc491673552)

[2.2.2. Name\_Var\_002 ([1] Clause 5.4.7 - table 1 - 1h) 12](#_Toc491673553)

[2.2.3. Name\_Var\_003 ([1] Clause 5.4.7 - table 1 - 1h) 12](#_Toc491673554)

[2.2.4. Name\_Var\_004 12](#_Toc491673555)

[2.2.5. Name\_Var\_005 12](#_Toc491673556)

[2.3. Method Naming 13](#_Toc491673557)

[2.3.1. Name\_Method\_001 ([1] Clause 5.4.7 - table 1 - 1h) 13](#_Toc491673558)

[2.4. Type Naming 13](#_Toc491673559)

[2.4.1. Name\_Type\_001 ([1] Clause 5.4.7 - table 1 - 1h) 13](#_Toc491673560)

[2.4.2. Name\_Type\_002 ([1] Clause 5.4.7 - table 1 - 1h) 13](#_Toc491673561)

[2.4.3. Name\_Type\_003 13](#_Toc491673562)

[2.4.4. Name\_Type\_004 14](#_Toc491673563)

[2.4.5. Name\_Type\_005 15](#_Toc491673564)

[2.5. Macro Naming 16](#_Toc491673565)

[2.5.1. Name\_Macro\_001 ([1] Clause 5.4.7 - table 1 - 1h) 16](#_Toc491673566)

[2.6. Parameter Naming 16](#_Toc491673920)

[2.6.1. Name\_Param\_001 16](#_Toc491673921)

[2.6.2. Name\_Param\_002 16](#_Toc491673922)

[2.7. Properties Naming 17](#_Toc491673947)

[2.7.1. Name\_Prop\_001 17](#_Toc491673949)

[2.7.2. Name\_Prop\_002 17](#_Toc491673950)

[2.7.3. Name\_Prop\_003 17](#_Toc491673951)

[2.7.4. Name\_Prop\_004 18](#_Toc491673952)

[2.7.5. Name\_Prop\_005 18](#_Toc491673954)

[2.8. Capitalization 18](#_Toc491673955)

[2.8.1. Name\_Cap\_001 18](#_Toc491674040)

[2.9. Abbreviations 19](#_Toc491674042)

[2.9.1. Name\_Abbr\_001 19](#_Toc491674080)

[2.9.2. Name\_Abbr\_002 20](#_Toc491674082)

[2.9.3. Name\_Abbr\_003 20](#_Toc491674084)

[2.9.4. Name\_Abbr\_004 20](#_Toc491674085)

[2.9.5. Name\_Abbr\_005 20](#_Toc491674086)

[2.10. Namespace Naming 21](#_Toc491674089)

[2.10.1. Name\_NaSpc\_001 21](#_Toc491674127)

[2.10.2. Name\_NaSpc\_002 21](#_Toc491674128)

[2.10.3. Name\_NaSpc\_003 21](#_Toc491674129)

[2.10.4. Name\_NaSpc\_004 22](#_Toc491674131)

[2.10.5. Name\_NaSpc\_005 22](#_Toc491674132)

[2.10.6. Name\_NaSpc\_006 22](#_Toc491674135)

[2.10.7. Name\_NaSpc\_007 23](#_Toc491674137)

[2.11. Class Naming 23](#_Toc491674138)

[2.11.1. Name\_Class\_001 23](#_Toc491674170)

[2.12. Event Naming 23](#_Toc491674172)

[2.12.1. Name\_Event\_001 23](#_Toc491674174)

[2.13. Field Naming 24](#_Toc491674176)

[2.13.1. Name\_Field\_001 24](#_Toc491674197)

[2.14. Resources Naming 24](#_Toc491674198)

[2.14.1. Name\_Res\_001 24](#_Toc491674215)

[2.15. Assemblies and DLLs Naming 25](#_Toc491674216)

[2.15.1. Name\_AsmDLL\_001 25](#_Toc491674226)

[2.15.2. Name\_AsmDLL\_002 25](#_Toc491674228)

[2.16. General Naming 26](#_Toc491674229)

[2.16.1. Name\_Gen\_001 26](#_Toc491674230)

[2.16.2. Name\_Gen\_002 26](#_Toc491674231)

[3. Coding style 27](#_Toc491674235)

[3.1. Format 27](#_Toc491674236)

[3.1.1. Style\_Format\_001 ([1] Clause 5.4.7 - table 1 - 1g) 27](#_Toc491674237)

[3.1.2. Style\_Format\_002 ([1] Clause 5.4.7 - table 1 - 1g) 27](#_Toc491674239)

[3.1.3. Style\_Format\_003 ([1] Clause 5.4.7 - table 1 - 1g) 28](#_Toc491674241)

[3.1.4. Style\_Format\_004 ([1] Clause 5.4.7 - table 1 - 1g) 29](#_Toc491674242)

[3.1.5. Style\_Format\_005 ([1] Clause 5.4.7 - table 1 - 1g) 29](#_Toc491674243)

[3.1.6. Style\_Format\_006 ([1] Clause 5.4.7 - table 1 - 1g) 29](#_Toc491674244)

[3.1.7. Style\_Format\_007 ([1] Clause 5.4.7 - table 1 - 1g) 29](#_Toc491674245)

[3.1.8. Style\_Format\_008 ([1] Clause 5.4.7 - table 1 - 1g) 30](#_Toc491674246)

[3.1.9. Style\_Format\_009 ([1] Clause 5.4.7 - table 1 - 1g) 30](#_Toc491674247)

[3.1.10. Style\_Format\_010 ([1] Clause 5.4.7 - table 1 - 1g) 30](#_Toc491674248)

[3.1.11. Style\_Format\_011 ([1] Clause 5.4.7 - table 1 - 1g) 31](#_Toc491674249)

[3.1.12. Style\_Format\_012 ([1] Clause 5.4.7 - table 1 - 1g) 31](#_Toc491674250)

[3.1.13. Style\_Format\_013 ([1] Clause 5.4.7 - table 1 - 1g) 32](#_Toc491674251)

[3.1.14. Style\_Format\_014 ([1] Clause 5.4.7 - table 1 - 1g) 33](#_Toc491674252)

[3.1.15. Style\_Format\_015 ([1] Clause 5.4.7 - table 1 - 1g) 33](#_Toc491674253)

[3.1.16. Style\_Format\_016 34](#_Toc491674255)

[3.1.17. Style\_Format\_017 34](#_Toc491674256)

[3.1.18. Style\_Format\_018 34](#_Toc491674257)

[3.2. File Structure 34](#_Toc491674258)

[3.2.1. Style\_File\_001 34](#_Toc491674259)

[3.2.2. Style\_File\_002 ([1] Clause 5.4.7 - table 1 – 1a) 34](#_Toc491674260)

[3.2.3. Style\_File\_003 ([1] Clause 5.4.7 - table 1 - 1g) 35](#_Toc491674261)

[3.2.4. Style\_File\_004 35](#_Toc491674262)

[3.2.5. Style\_File\_005 35](#_Toc491674263)

[3.2.6. Style\_File\_006 36](#_Toc491674264)

[3.3. Comments 36](#_Toc491674265)

[3.3.1. Style\_Comment\_001 ([1] Clause 5.4.7 - table 1 - 1g) 36](#_Toc491674266)

[3.3.2. Style\_Comment\_002 ([1] Clause 5.4.7 - table 1 - 1g) 36](#_Toc491674268)

[3.3.3. Style\_Comment\_003 ([1] Clause 5.4.7 - table 1 - 1g) 36](#_Toc491674269)

[3.3.4. Style\_Comment\_004 ([1] Clause 5.4.7 - table 1 - 1g) 37](#_Toc491674270)

[3.3.5. Style\_Comment\_005 ([1] Clause 5.4.7 - table 1 - 1g) 37](#_Toc491674271)

[3.3.6. Style\_Comment\_006 ([1] Clause 5.4.7 - table 1 - 1g) 38](#_Toc491674272)

[3.3.7. Style\_Comment\_007 ([1] Clause 5.4.7 - table 1 - 1g) 38](#_Toc491674273)

[4. Coding Rules 39](#_Toc491674274)

[4.1. Conformance to external rules 39](#_Toc491674275)

[4.1.1. Rules\_Ext\_001 39](#_Toc491674276)

[4.1.2. Rules\_Ext\_002 39](#_Toc491674277)

[4.1.3. Rules\_Ext\_003 39](#_Toc491674286)

[4.2. Environment 39](#_Toc491674287)

[4.2.1. Rules\_Envr\_001 ([1] Clause 5.4.7 - table 1 - 1d) 39](#_Toc491674288)

[4.2.2. Rules\_Envr\_002 ([1] Clause 5.4.7 - table 1 - 1d) 39](#_Toc491674289)

[4.2.3. Rules\_Envr\_003 ([1] Clause 5.4.7 - table 1 - 1d) 39](#_Toc491674290)

[4.2.4. Rules\_Envr\_004 40](#_Toc491674291)

[4.2.5. Rules\_Envr\_005 ([1] Clause 5.4.7 - table 1 - 1d) 40](#_Toc491674292)

[4.2.6. Rules\_Envr\_006 40](#_Toc491674293)

[4.3. Comments/Documentation 40](#_Toc491674295)

[4.3.1. Rules\_Comment\_001 ([1] Clause 5.4.7 - table 1 - 1g) 40](#_Toc491674296)

[4.3.2. Rules\_Comment\_002 ([1] Clause 5.4.7 - table 1 - 1g) 40](#_Toc491674297)

[4.3.3. Rules\_Comment\_003 ([1] Clause 5.4.7 - table 1 - 1g) 41](#_Toc491674298)

[4.3.4. Rules\_Comment\_004 ([1] Clause 5.4.7 - table 1 - 1g) 41](#_Toc491674299)

[4.3.5. Rules\_Comment\_005 ([1] Clause 5.4.7 - table 1 - 1g) 41](#_Toc491674300)

[4.3.6. Rules\_Comment\_006 42](#_Toc491674301)

[4.3.7. Rules\_Comment\_007 42](#_Toc491674303)

[4.3.8. Rules\_Comment\_008 43](#_Toc491674304)

[4.4. Identifiers 44](#_Toc491674305)

[4.4.1. Rules\_Identifier\_001 ([1] Clause 5.4.7 - table 1 - 1b) 44](#_Toc491674306)

[4.5. Types 44](#_Toc491674307)

[4.5.1. Rules\_Types\_001 ([1] Clause 5.4.7 - table 1 - 1c) 44](#_Toc491674308)

[4.5.2. Rules\_Types\_002 44](#_Toc491674309)

[4.5.3. Rules\_Types\_003 ([1] Clause 5.4.7 - table 1 - 1h) 45](#_Toc491674310)

[4.5.4. Rules\_Types\_004 45](#_Toc491674312)

[4.5.5. Rules\_Types\_005 45](#_Toc491674313)

[4.6. Constants 45](#_Toc491674316)

[4.6.1. Rules\_Const\_001 ([1] Clause 5.4.7 - table 1 - 1g) 45](#_Toc491674317)

[4.6.2. Rules\_Const\_002 ([1] Clause 5.4.7 - table 1 - 1a) 46](#_Toc491674319)

[4.6.3. Rules\_Const\_003 46](#_Toc491674320)

[4.6.4. Rules\_Const\_004 47](#_Toc491674322)

[4.7. Declarations and Definitions 47](#_Toc491674347)

[4.7.1. Rules\_Defn\_Decl\_001 ([1] Clause 8.4.4 - table 8 - 1e) 47](#_Toc491674348)

[4.7.2. Rules\_Defn\_Decl\_002 47](#_Toc491674350)

[4.7.3. Rules\_Defn\_Decl\_003 ([1] Clause 5.4.7 - table 1 - 1a) 47](#_Toc491674352)

[4.7.4. Rules\_Defn\_Decl\_004 ([1] Clause 5.4.7 - table 1 - 1g) 47](#_Toc491674354)

[4.7.5. Rules\_Defn\_Decl\_005 ([1] Clause 5.4.7 - table 1 - 1g) 48](#_Toc491674355)

[4.7.6. Rules\_Defn\_Decl\_006 48](#_Toc491674357)

[4.7.7. Rules\_Defn\_Decl\_007 48](#_Toc491674359)

[4.7.8. Rules\_Defn\_Decl\_008 48](#_Toc491674361)

[4.7.9. Rules\_Defn\_Decl\_009 48](#_Toc491674363)

[4.7.10. Rules\_Defn\_Decl\_010 48](#_Toc491674365)

[4.7.11. Rules\_Defn\_Decl\_011 48](#_Toc491674366)

[4.7.12. Rules\_Defn\_Decl\_012 ([1] Clause 5.4.7 - table 1 – 1d) 48](#_Toc491674369)

[4.7.13. Rules\_Defn\_Decl\_013 ([1] Clause 5.4.7 - table 1 – 1d) 49](#_Toc491674370)

[4.7.14. Rules\_Defn\_Decl\_014 49](#_Toc491674371)

[4.7.15. Rules\_Defn\_Decl\_015 ([1] Clause 5.4.7 - table 1 - 1a) 49](#_Toc491674372)

[4.7.16. Rules\_Defn\_Decl\_016([1] Clause 5.4.7 - table 1 - 1a) 49](#_Toc491674374)

[4.7.17. Rules\_Defn\_Decl\_017 ([1] Clause 5.4.7 - table 1 - 1a) 50](#_Toc491674375)

[4.7.18. Rules\_Defn\_Decl\_018 ([1] Clause 5.4.7 - table 1 - 1c) 50](#_Toc491674376)

[4.7.19. Rules\_Defn\_Decl\_019 ([1] Clause 5.4.7 - table 1 - 1c) 50](#_Toc491674378)

[4.7.20. Rules\_Defn\_Decl\_020 ([1] Clause 8.4.4 - table 8 - 1d) 50](#_Toc491674379)

[4.7.21. Rules\_Defn\_Decl\_021 51](#_Toc491674380)

[4.7.22. Rules\_Defn\_Decl\_022 51](#_Toc491674382)

[4.7.23. Rules\_Defn\_Decl\_023 51](#_Toc491674383)

[4.7.24. Rules\_Defn\_Decl\_024 51](#_Toc491674384)

[4.7.25. Rules\_Defn\_Decl\_025 52](#_Toc491674385)

[4.7.26. Rules\_Defn\_Decl\_026 52](#_Toc491674386)

[4.7.27. Rules\_Defn\_Decl\_027 52](#_Toc491674387)

[4.7.28. Rules\_Defn\_Decl\_027 53](#_Toc491674389)

[4.8. Initialization 53](#_Toc491674390)

[4.8.1. Rules\_Init\_001 ([1] Clause 8.4.4 - table 8 - 1c) 53](#_Toc491674391)

[4.8.2. Rules\_Init\_002 53](#_Toc491674392)

[4.9. Control Statement Expressions 54](#_Toc491674394)

[4.9.1. Rules\_Expr\_001 ([1] Clause 5.4.7 - table 1 – 1d) 54](#_Toc491674395)

[4.9.2. Rules\_Expr\_002 ([1] Clause 5.4.7 - table 1 – 1a) 54](#_Toc491674397)

[4.9.3. Rules\_Expr\_003 ([1] Clause 5.4.7 - table 1 – 1d) 54](#_Toc491674398)

[4.9.4. Rules\_Expr\_004 ([1] Clause 5.4.7 - table 1 - 1d) 55](#_Toc491674399)

[4.9.5. Rules\_Expr\_005 ([1] Clause 5.4.7 - table 1 - 1e) 55](#_Toc491674400)

[4.9.6. Rules\_Expr\_006 55](#_Toc491674401)

[4.9.7. Rules\_Expr\_007 ([1] Clause 5.4.7 - table 1 - 1d) 55](#_Toc491674402)

[4.9.8. Rules\_Expr\_008 56](#_Toc491674403)

[4.9.9. Rules\_Expr\_009 56](#_Toc491674404)

[4.9.10. Rules\_Expr\_010 57](#_Toc491674405)

[4.9.11. Rules\_Expr\_011 ([1] Clause 8.4.4 - table 8 – 1b) 57](#_Toc491674406)

[4.9.12. Rules\_Expr\_012 ([1] Clause 5.4.7 - table 1 - 1c) 58](#_Toc491674407)

[4.9.13. Rules\_Expr\_013 58](#_Toc491674408)

[4.10. Control Flow 58](#_Toc491674410)

[4.10.1. Rules\_CtrlFlow\_001 ([1] Clause 5.4.7 - table 1 - 1d) 58](#_Toc491674411)

[4.10.2. Rules\_CtrlFlow\_002 ([1] Clause 5.4.7 - table 1 - 1d) 58](#_Toc491674413)

[4.10.3. Rules\_CtrlFlow\_003 ([1] Clause 5.4.7 - table 1 - 1a) 59](#_Toc491674414)

[4.10.4. Rules\_CtrlFlow\_004 ([1] Clause 5.4.7 - table 1 – 1a) 59](#_Toc491674415)

[4.10.5. Rules\_CtrlFlow\_005 ([1] Clause 5.4.7 - table 1 - 1d) 60](#_Toc491674416)

[4.10.6. Rules\_CtrlFlow\_006 ([1] Clause 5.4.7 - table 1 - 1d) 60](#_Toc491674417)

[4.10.7. Rules\_CtrlFlow\_007 61](#_Toc491674418)

[4.10.8. Rules\_CtrlFlow\_008 ([1] Clause 5.4.7 - table 1 - 1e) 61](#_Toc491674419)

[4.10.9. Rules\_CtrlFlow\_009 ([1] Clause 8.4.4 - table 8 - 1h) 61](#_Toc491674421)

[4.10.10. Rules\_CtrlFlow\_010 ([1] Clause 8.4.4 - table 8 - 1i) 62](#_Toc491674422)

[4.10.11. Rules\_CtrlFlow\_011 63](#_Toc491674423)

[4.10.12. Rules\_CtrlFlow\_012 63](#_Toc491674424)

[4.10.13. Rules\_CtrlFlow\_013 63](#_Toc491674425)

[4.10.14. Rules\_CtrlFlow\_014 64](#_Toc491674426)

[4.11. Functions 64](#_Toc491674458)

[4.11.1. Rules\_Func\_001 ([1] Clause 5.4.7 - table 1 - 1d) 64](#_Toc491674459)

[4.11.2. Rules\_Func\_002 ([1] Clause 5.4.7 - table 1 - 1d) 65](#_Toc491674460)

[4.11.3. Rules\_Func\_003 ([1] Clause 5.4.7 - table 1 - 1d) 65](#_Toc491674462)

[4.11.4. Rules\_Func\_004 ([1] Clause 8.4.4 - table 8 - 1a) 65](#_Toc491674463)

[4.11.5. Rules\_Func\_005 ([1] Clause 8.4.4 - table 8 - 1j) 66](#_Toc491674464)

[4.11.6. Rules\_Func\_006 66](#_Toc491674465)

[4.11.7. Rules\_Func\_007 66](#_Toc491674466)

[4.12. Pointers and Arrays 66](#_Toc491674467)

[4.12.1. Rules\_Ptr\_001 ([1] Clause 8.4.4 - table 8 - 1f) 66](#_Toc491674468)

[4.12.2. Rules\_Ptr\_002 ([1] Clause 5.4.7 - table 1 - 1d) 66](#_Toc491674470)

[4.12.3. Rules\_Ptr\_003 ([1] Clause 5.4.7 - table 1 - 1d) 66](#_Toc491674471)

[4.12.4. Rules\_Ptr\_004 ([1] Clause 5.4.7 - table 1 - 1d) 67](#_Toc491674472)

[4.12.5. Rules\_Ptr\_005 ([1] Clause 5.4.7 - table 1 - 1d) 67](#_Toc491674473)

[4.13. Structures and Unions 67](#_Toc491674474)

[4.13.1. Rules\_Struct\_001 ([1] Clause 5.4.7 - table 1 - 1b) 67](#_Toc491674475)

[4.14. Pre-processing Directives 67](#_Toc491674476)

[4.14.1. Rules\_PreProcess\_001 ([1] Clause 5.4.7 - table 1 - 1g) 67](#_Toc491674477)

[4.14.2. Rules\_PreProcess\_002 67](#_Toc491674478)

[4.14.3. Rules\_PreProcess\_003 67](#_Toc491674480)

[4.14.4. Rules\_PreProcess\_004 67](#_Toc491674482)

[4.14.5. Rules\_PreProcess\_005 ([1] Clause 5.4.7 - table 1 - 1g) 67](#_Toc491674484)

[4.14.6. Rules\_PreProcess\_006 ([1] Clause 5.4.7 - table 1 - 1a) 67](#_Toc491674485)

[4.14.7. Rules\_PreProcess\_007 ([1] Clause 5.4.7 - table 1 - 1a) 67](#_Toc491674487)

[4.14.8. Rules\_PreProcess\_008 ([1] Clause 5.4.7 - table 1 - 1a) 67](#_Toc491674488)

[4.14.9. Rules\_PreProcess\_009 67](#_Toc491674490)

[4.14.10. Rules\_PreProcess\_010 ([1] Clause 5.4.7 - table 1 - 1g) 68](#_Toc491674491)

[4.14.11. Rules\_PreProcess\_011 ([1] Clause 8.4.4 - table 8 - 1g) 68](#_Toc491674493)

[4.14.12. Rules\_PreProcess\_012 68](#_Toc491674494)

[4.15. Optimization 68](#_Toc491674495)

[4.15.1. Rules\_Opt\_001 ([1] Clause 5.4.7 - table 1 – 1a) 68](#_Toc491674496)

[4.15.2. Rules\_Opt\_002 ([1] Clause 5.4.7 - table 1 - 1g) 68](#_Toc491674497)

[4.15.3. Rules\_Opt\_003 ([1] Clause 5.4.7 - table 1 - 1g) 68](#_Toc491674498)

[4.15.4. Rules\_Opt\_004 ([1] Clause 5.4.7 - table 1 – 1a) 68](#_Toc491674499)

[4.15.5. Rules\_Opt\_005 69](#_Toc491674500)

[4.15.6. Rules\_Opt\_006 69](#_Toc491674502)

[4.15.7. Rules\_Opt\_007 69](#_Toc491674503)

[4.15.8. Rules\_Opt\_008 69](#_Toc491674518)

[4.16. Events 69](#_Toc491674519)

[4.16.1. Rules\_Event\_001 69](#_Toc491674520)

[4.16.2. Rules\_Event\_002 70](#_Toc491674521)

[4.16.3. ptRules\_Event\_003 70](#_Toc491674522)

[4.17. Object Oriented Programming 70](#_Toc491674523)

[4.17.1. Rules\_Oop\_001 70](#_Toc491674524)

[4.18. Except Handling 70](#_Toc491674525)

[4.18.1. Rules\_Expt\_001 70](#_Toc491674526)

[4.18.2. Rules\_Expt\_002 71](#_Toc491674527)

[4.18.3. Rules\_Expt\_002 71](#_Toc491674528)

[5. Prpoject Settings and Structure 72](#_Toc491674529)

[5.1.1. Rules\_Project\_001 72](#_Toc491674530)

[5.1.2. Rules\_Project\_002 72](#_Toc491674531)

[5.1.3. Rules\_Project\_003 72](#_Toc491674532)

[5.1.4. Rules\_Project\_003 72](#_Toc491674533)

[5.1.5. Rules\_Project\_004 73](#_Toc491674534)

[5.1.6. Rules\_Project\_005 73](#_Toc491674535)

[5.1.7. Rules\_Project\_006 73](#_Toc491674536)

# Revision history

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SI. No** | **Version** | **Date** | **Reason for change** | **Author** | **Approved by** |
| 1 | 0.0.1 | 14-Aug-2017 | Initial Version by revising JB0019 Software Coding Guideline v.4 (draft) | Viet Nguyen  Tu Nguyen |  |
| 2 | 1.0.0 | 28-Aug-2017 | Finished reviewing in RVC and revising guideline according to findings | Viet Nguyen  Tu Nguyen |  |

# Introduction

## Purpose of the document

The purpose of this document is to lay down the C sharp coding rules, style and naming convention which has to be followed for MCAL generator tool development during the software development in C Sharp language.

The C Sharp coding guidelines gives the following benefit:

* It improves the software quality, readability and maintainability.
* It minimizes the software bug.

## Reference

1. ISO26262 Standard First Edition 2011 – Part 6:

“Part 6\_Product development at the software level.pdf”

# Naming Convention

This section specifies the naming conventions to be followed for file, functions, variables, constants, macros and types.

## File Naming

### Name\_File\_001 ([1] Clause 5.4.7 - table 1 - 1h)

**Rule:**

File name shall use Pascal case and only consist of alphanumeric characters. But it shall not start with numeric.

**Example:**

// Compliant

Adc.cs

BswLinConfiguration.cs

// Not compliant

59BswWdgConfiguration.cs

**Rationale:**

Even though most operating systems nowadays can handle spaces and special characters in file names some of the tool used in the software development may not handle them properly.

### Name\_File\_002 ([1] Clause 5.4.7 - table 1 - 1h)

**Rule:**

File name shall not contain more than 64 characters.

**Example:**

// Compliant

LinGaaConfigurationStruct.cs (28 characters long)

**Rationale:**

Readability.

### Name\_File\_003

**Rule:**

In case of MCU Driver related Generation Tool, the naming of the main file which contains program entry point shall be in following format: ‘<Msn>.cs’

**Example:**

- DIO module: ‘Dio.cs’.

- ADC module: ‘Adc.cs’.

**Rational:**

The file belongs to which module is easily understood.

### Name\_File\_004

**Rule:**

All file name should follow below scheme:

1. Normal source files

*‘<Component><Sub Component>.cs’*

2. Source file for interface class should have prefix "I"

*‘I<Component><Sub Component>.cs’*

• Where < *Sub* Component> contains zero or more separated sub-components.

**Example:**

// Compliant

- DIO moduleinterface: ‘IDioIntermediate.cs’.

- ADCX2x interfacemodule: ‘IX2xIntermediate.cs’.

**Rational:**

The file belongs to which module is easily understood.

### Name\_File\_005

**Rule:**

Do not use abbreviation, contraction or acronyms as part of file names if it is not widely accepted.

**Example:**

No required.

**Rational:**

The file name shall be easily readable and understandable.

## Variable Naming

### Name\_Var\_001 ([1] Clause 5.4.7 - table 1 - 1h)

**Rule:**

Use Camel case for local variables.

**Example:**

// Compliant

List<Container> channelSet;

uint numberOfChannel;

// Not compliant

List<Container> ChannelSet;

**Rationale:**

The variable scope can be specified easier.

### Name\_Var\_002 ([1] Clause 5.4.7 - table 1 - 1h)

**Rule:**

Use Pascal case for global variables.

**Example:**

// Compliant

public unit DioChanelPortId;

// Not compliant

public unit dioChanelPortId;

**Rationale:**

The variable scope can be specified easier.

### Name\_Var\_003 ([1] Clause 5.4.7 - table 1 - 1h)

**Rule:**

A meaning full name (with respect to that context) should be given to the variables. Avoid giving name as Temp, Busy, etc.

**Example:**

// Compliant

uint8 tempBufferValue

// Not compliant

uint8 temp

**Rationale:**

Readability

### Name\_Var\_004

**Rule:**

Reserved identifiers, class, namespace and methods in the standard libraries, shall not be defined, redefined or undefined. The names of namespaces, classes and methods in the standard libraries shall not be reused.

**Example:**

Not required.

**Rationale:**

There are some specific reserved words and method names which are known to give rise to undefined behaviours if they are redefined or undefined.

### Name\_Var\_005

**Rule:**

The following rules should be applied for all variables naming:

* Use a noun or noun phrase to name properties that are easily readable and understandable.
* Use i, j, k for counting variables.
* Use UPPER CASE constant. An underscore can be used to separate terms when necessary.
* Do not use Hungarian notation (to indicate the type of variable).
* Do not use underscores.

**Example:**

// Compliant

string ContainerName;

const string MAX\_VALUE = “1.0.0”;

// Not compliant

// Hungarian notation

string stringName;

// Use underscore

string Container\_Name;

**Rationale:**

Consistent with the Microsoft's .NET Framework and easy to read.

## Method Naming

### Name\_Method\_001 ([1] Clause 5.4.7 - table 1 - 1h)

**Rule:**

* Using Pascal case for public/protected methods and Camel case for private methods.
* DO give methods names that are verbs or verb phrases.
* DO NOT use the underscore character (\_).

**Example:**

Not required

**Rationale:**

Readability

## Type Naming

### Name\_Type\_001 ([1] Clause 5.4.7 - table 1 - 1h)

Not applicable.

### Name\_Type\_002 ([1] Clause 5.4.7 - table 1 - 1h)

Not applicable**.**

### Name\_Type\_003

**Rule:**

Names of enumeration types (also called enums) in general should follow below rules:

* Using Pascal case for naming enumerations type.
* Using a singular type name for an enumeration unless its values are bit fields.
* Using a plural type name for an enumeration for masking bit fields or doing bitwise comparisons, also called flags enum.
* DO NOT use an "Enum" suffix in enum type names.
* DO NOT use "Flag" or "Flags" suffixes in enum type names.
* DO NOT use a prefix on enumeration value names.

**Example:**

// Compliant

// Using a singular type name

public enum Color

{

Red,

Green,

Blue

}

// Flag enumeration is used to indicate the bit field masks

// Using a plural type name

[Flags]

public enum Dockings

{

None = 0,

Top = 1,

Right = 2,

}

// Not compliant

// “Enum” suffix shouldn’t be used

public enum CoinEnum

{

Penny,

Nickel,

Dime

}

**Rationale:**

Makes the code more natural to read. Plural flags because enum can hold multiple values (using bitwise 'OR').

### Name\_Type\_004

**Rule:**

Names of Generic Type Parameter should be following below rules:

* **DO** name generic type parameters with descriptive names unless a single-letter name is completely self-explanatory and a descriptive name would not add value.
* **CONSIDER** using “T” as the type parameter name for types with one single-letter type parameter.
* **DO** prefix descriptive type parameter names with “T”.
* **CONSIDER** indicating constraints placed on a type parameter in the name of the parameter.

**Example:**

// Compliant

a) Using “T” as the type parameter name for types with one single-letter type parameter.

public int IComparer<T> { ... }

public delegate bool Predicate<T>(T item);

public struct Nullable<T> where T:struct { ... }

b) Prefix descriptive type parameter names with “T”

public interface ISessionChannel<TSession> where TSession : ISession

{

TSession Session { get; }

}

c) Indicating constraints placed on a type parameter in the name of the parameter.

Using a parameter constrained to “ISession” might be called “TSession”

**Rationale:**

Readability

### Name\_Type\_005

**Rule:**

Names of Common Type should be following the guidelines described in the following table:

|  |  |
| --- | --- |
| **Base Type** | **Derived/Implementing Type Guideline** |
| System.Attribute | **✓ DO** add the suffix "Attribute" to names of custom attribute classes. add the suffix "Attribute" to names of custom attribute classes. |
| System.Delegate | **✓ DO** add the suffix "EventHandler" to names of delegates that are used in events. |
|  |
| **✓ DO** add the suffix "Callback" to names of delegates other than those used as event handlers. |
|  |
| **X DO NOT** add the suffix "Delegate" to a delegate. |
| System.EventArgs | **✓ DO** add the suffix "EventArgs." |
| System.Enum | **X DO NOT** derive from this class; use the keyword supported by your language instead; for example, in C#, use the enum keyword. |
| System.Exception | **✓ DO** add the suffix "Exception." |
| IDictionary | **✓ DO** add the suffix "Dictionary." Note that IDictionary is a specific type of collection, but this guideline takes precedence over the more general collections guideline that follows. |
| IDictionary<TKey,TValue> |
| IEnumerable | **✓ DO** add the suffix "Collection." |
| ICollection |
| IList |
| IEnumerable<T> |
| ICollection<T> |
| IList<T> |
| System.IO.Stream | **✓ DO** add the suffix "Stream." |
| CodeAccessPermission IPermission | **✓ DO** add the suffix "Permission." |

**Example:**

Not required

**Rationale:**

Makes the code more natural to read.

## Macro Naming

### Name\_Macro\_001 ([1] Clause 5.4.7 - table 1 - 1h)

Not applicable.











## Parameter Naming

### Name\_Param\_001

**Rule:**

* DO use Camel case in parameter names.
* DO use meaning full parameter names.
* CONSIDER using names based on a parameter’s meaning rather than the parameter’s type.

**Example:**

Not required

**Rationale:**

Readability

### Name\_Param\_002

**Rule:**

Naming Operator Overload Parameters should be followed below rules:

* DO use “left” and “right” for binary operator overload parameter names if there is no meaning to the parameters.
* DO use value for unary operator overload parameter names if there is no meaning to the parameters.
* CONSIDER meaningful names for operator overload parameters if doing so adds significant value.
* DO NOT use abbreviations or numeric indices for operator overload parameter names.

**Example:**

Not required

**Rationale:**

Readability

## Properties Naming

### Name\_Prop\_001

**Rule:**

* Using Pascal case for property names.
* DO name properties using a noun, noun phrase, or adjective.

**Example:**

// Compliant

Not required

// Not compliant

// This pattern typically indicates that the property should really be a method.

public string TextWriter { get {...} set {...} }

public string GetTextWriter(int value) { ... }

**Rationale:**

Readability

### Name\_Prop\_002

**Rule:**

* DO NOT have properties that match the name of "Get" methods.

**Example:**

// Not compliant

// This pattern typically indicates that the property should really be a method.

public string TextWriter { get {...} set {...} }

public string GetTextWriter(int value) { ... }

**Rationale:**

Readability

### Name\_Prop\_003

**Rule:**

* DO name collection properties with a plural phrase describing the items in the collection instead of using a singular phrase followed by "List" or "Collection."

**Example:**

// Compliant

List<string> EmployeeNames;

// Not compliant

List<string> EmployeeName;

**Rationale:**

Readability

### Name\_Prop\_004

**Rule:**

* DO name Boolean properties with an affirmative phrase. Optionally, you can also prefix Boolean properties with "Is," "Can," or "Has," but only where it adds value.

**Example:**

// Compliant

public bool CanSeek {...}

public bool IsPath {...}

public bool HasChild {...}

// Not compliant

public bool Seek {...}

public bool CannotSeek {...}

**Rationale:**

Readability

### Name\_Prop\_005

**Rule:**

CONSIDER giving a property the same name as its type.

**Example:**

// Compliant

// The following property correctly gets and sets an enum value named Color, so the property is named Color

public enum Color {...}

public class Control

{

public Color Color { get {...} set {...} }

}

**Rationale:**







Readability

## Capitalization



### Name\_Cap\_001

**Rule:**

The following table summarizes the capitalization rules should be used for the different types of identifiers.

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Case** | **Example** |
| Class | Pascal(1\*) | **AppDomain** |
| Enum type | Pascal | **ErrorLevel** |
| Enum values | Pascal | **FatalError** |
| Event | Pascal | **ValueChange** |
| Exception class | Pascal | **WebException** |
| **Note**   Always ends with the suffix **Exception**. |
| Read-only Static field | Pascal | **RedValue** |
| Interface | Pascal | **IDisposable** |
| **Note**   Always begins with the prefix I. |
| Method | Pascal | **ToString** |
| Namespace | Pascal | **System.Drawing** |
| Parameter | Camel(2\*) | **typeName** |
| Property | Pascal | **BackColor** |
| Protected instance field | Camel | **redValue** |
| **Note**   Rarely used. A property is preferable to using a protected instance field. |
| Public instance field | Pascal | **RedValue** |
| **Note**   Rarely used. A property is preferable to using a public instance field. |

*(1\*), (2\*) Refer to Appendix. A1*

Table 2.2 Capitalization Naming

**Example:** (Refer to “Table 2.2 Capitalization Naming”)

**Rationale:**

To avoid confusion and keep consistence within the source code.

## Abbreviations

### Name\_Abbr\_001

**Rule:**

* Do not use abbreviations or contractions as parts of identifier names.

**Example:**

// Compliant

GetWindow()

// Not compliant

GetWin()

**Rationale:**

To avoid confusion and guarantee cross-language interoperation.

### Name\_Abbr\_002

**Rule:**

Do not use acronyms that are not generally accepted in the computing field.

**Example:**

Not required

**Rationale:**

To avoid confusion and guarantee cross-language interoperation.

### Name\_Abbr\_003

**Rule:**

Where appropriate, use well-known acronyms to replace lengthy phrase names.

**Example:**

// Compliant

use UI for User Interface

use UUID for Universally Unique Identifier

// Not compliant

use LCC for Lin Channel Configure (LCC is not a well-known acronyms in software development)

**Rationale:**

To avoid confusion and guarantee cross-language interoperation.

### Name\_Abbr\_004

**Rule:**

* When using acronyms, use Pascal case or camel case for acronyms more than two characters long. However, you should capitalize acronyms that consist of only two characters.

**Example:**

// Compliant

HtmlButton or htmlButton

System.IO (Capitalize acronyms that consist of only two characters)

// Not compliant

HTMLButton or htmlbutton

System.Io

**Rationale:**

To avoid confusion and guarantee cross-language interoperation.

### Name\_Abbr\_005

**Rule:**

Do not use abbreviations in identifiers or parameter names. If you must use abbreviations, use Camel Case for abbreviations that consist of more than two characters,even if this contradicts the standard abbreviation of the word.

**Example:**

Not required

**Rationale:**

To avoid confusion and guarantee cross-language interoperation.

## Namespace Naming

### Name\_NaSpc\_001

**Rule:**

The general rule for naming namespaces is to use the company name followed by the technology name and optionally the feature and design as “CompanyName.Technology[.Feature][.Design]”

**Example:**

// Compliant

Renesas.McalGenerator.Generic.DataHelper

// Not compliant

Generator.Generic.DataHelper

**Rationale:**

To avoid confusion.

### Name\_NaSpc\_002

**Rule:**

Use a stable, recognized technology name at the second level of a hierarchical name. Use organizational hierarchies as the basis for namespace hierarchies. Name a namespace that contains types that provide design-time functionality for a base namespace with the “.Design” suffix.

**Example:**

// Compliant

Renesas.McalGenerator.Generic.UI.Design

// Not compliant

Not required

**Rationale:**

To avoid confusion.

### Name\_NaSpc\_003

**Rule:**

A nested namespace should have a dependency on types in the containing namespace.

**Example:**

// Compliant

The classes in the System.Web.UI.Design depend on the classes in System.Web.UI. However, the classes in System.Web.UI do not depend on the classes in System.Web.UI.Design.

// Not compliant

Not required

**Rationale:**

To avoid confusion.

### Name\_NaSpc\_004

**Rule:**

You should use Pascal case for namespaces, and separate logical components with periods.

**Example:**

// Compliant

Renesas.McalGenerator.Generic

Renesas.McalGenerator.Specific

// Not compliant

Not Required

**Rationale:**

To avoid confusion.

### Name\_NaSpc\_005

**Rule:**

Use plural namespace names if it is semantically appropriate.

**Example:**

// Compliant

EcuConfiguration.Collections

// Not compliant

EcuConfiguration.Collection

**Rationale:**

To avoid confusion.

### Name\_NaSpc\_006

**Rule:**

Do not use the same name for a namespace and a class.

**Example:**

Not required.

**Rationale:**

To avoid confusion.

### Name\_NaSpc\_007

**Rule:**

A namespace name does not have to parallel an assembly name.

**Example:**

// Compliant

If project’s assembly is named MyCompany.MyTechnology.dll, it does not have to contain a MyCompany.MyTechnology namespace.

// Not compliant

Not required

**Rationale:**

To avoid confusion and guarantee cross-language interoperation.

## Class Naming

### Name\_Class\_001

**Rule:**

The following rules outline the guidelines for naming classes:

* Use a noun or noun phrase to name a class.
* Use Pascal case.
* Use abbreviations sparingly.
* Do not use a type prefix on a class name.
* Do not use the underscore character (\_).
* Where appropriate, use a compound word to name a derived class. The second part of the derived class's name should be the name of the base class.

**Example:**

// Compliant

Using “public class FileStream”

Using “public class StringUtils”

Using “ApplicationException” is an appropriate name for a class derived from a class named “Exception”.

// Not compliant

Using “public class CFileStream” (using prefix)

Using “public class File\_Stream” (using underscore)

**Rationale:**

To avoid confusion.

## Event Naming

### Name\_Event\_001

**Rule:**

Events naming should be followed below rules:

* DO name events with a verb or a verb phrase.
* DO give events names with a concept of before and after, using the present and past tenses.
* DO NOT use "Before" or "After" prefixes or postfixes to indicate pre- and post-events. Use present and past tenses as just described.
* DO name event handlers (delegates used as types of events) with the "EventHandler" suffix.
* DO use two parameters named sender and e in event handlers.
* DO name event argument classes with the "EventArgs" suffix.

**Example:**

// Compliant

public delegate void ClickedEventHandler(object sender, ClickedEventArgs e);

// A close event that is raised before a window is closed would be called “Closing”, and one that is raised after

// the window is closed would be called “Closed”.

public void ClosingWindows(object sender, XxxEventArgs e);

public void ClosedWindows(object sender, XxxEventArgs e);

// Not compliant

Not required

**Rationale:**

Readability.

## Field Naming

### Name\_Field\_001

**Rule:**

Field naming should be followed below rules:

* DO use Pascal case in public and protected field names.
* DO use Camel case in private field names.
* DO name fields using a noun, noun phrase, or adjective.
* DO NOT use a prefix for field names.

**Example:**

Not required

**Rationale:**

Readability.

## Resources Naming

### Name\_Res\_001

**Rule:**

Resources naming should be followed below rules:

* DO use Pascal case in resource keys.
* DO provide descriptive rather than short identifiers.
* DO NOT use language-specific keywords of the main CLR languages.
* DO use only alphanumeric characters and underscores in naming resources.

Note: Because localizable resources can be referenced via certain objects as if they were properties, the naming guidelines for resources are similar to property guidelines.

**Example:**

Not required.

**Rationale:**

Readability.

## Assemblies and DLLs Naming

### Name\_AsmDLL\_001

**Rule:**

DO choose names for your assembly DLLs that suggest large chunks of functionality.

Note:

Assembly and DLL names don’t have to correspond to namespace names, but it is reasonable to follow the namespace name when naming assemblies. A good rule of thumb is to name the DLL based on the common prefix of the assemblies contained in the assembly

**Example:**

// Compliant

Using an assembly with two namespaces, “MyCompany.MyTechnology.FirstFeature” and “MyCompany.MyTechnology.SecondFeature”, could be called “MyCompany.MyTechnology.dll”

// Not compliant

Not required

**Rationale:**

Easy to map DLL with assembly for managing code program.

### Name\_AsmDLL\_002

**Rule:**

CONSIDER naming DLLs according to the following pattern:

<Company>.<Component>.dll

* Where <Component> contains one or more dot-separated clauses.

**Example:**

// Compliant

Renesas.McalGenerator.Generic.dll

// Not compliant

Not required

**Rationale:**

Easy to map DLL with assembly for managing code program.

## General Naming

### Name\_Gen\_001

**Rule:**

Choosing words for naming convention should follow below rules:

* DO choose easily readable identifier names.
* DO favor readability over brevity.
* DO NOT use underscores, hyphens, or any other nonalphanumeric characters.
* DO NOT use Hungarian notation.
* AVOID using identifiers that conflict with keywords of widely used programming languages.
* DO use a common name, such as value or item, rather than repeating the type name, in the rare cases when an identifier has no semantic meaning and the type of the parameter is not important.

**Example:**

Not required.

**Rationale:**

Readability

### Name\_Gen\_002

**Rule:**

Naming new version of existing APIs should follow below rules:

* DO use a name similar to the old API when creating new versions of an existing API.
* DO prefer adding a suffix rather than a prefix to indicate a new version of an existing API.
* CONSIDER using a brand new, but meaningful identifier, instead of adding a suffix or a prefix.
* DO use a numeric suffix to indicate a new version of an existing API, particularly if the existing name of the API is the only name that makes sense (i.e., if it is an industry standard) and if adding any meaningful suffix (or changing the name) is not an appropriate option.
* DO NOT use the "Ex" (or a similar) suffix for an identifier to distinguish it from an earlier version of the same API.
* DO use the "64" suffix when introducing versions of APIs that operate on a 64-bit integer (a long integer) instead of a 32-bit integer. You only need to take this approach when the existing 32-bit API exists; don’t do it for brand new APIs with only a 64-bit version.

**Example:**

Not required.

**Rationale:**

Readabilit

# Coding style

## Format

### Style\_Format\_001 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Each file shall have a file template and file description banner agreed for each project. The template should be defined at the starting of development.

**Example:**

Not required.

**Rationale:**

This helps users determine which platforms this code had been developed on and also which version they are currently using.

### Style\_Format\_002 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Items of similar purpose such as namespace, class, type of member, attributes, declaring global variables, methods, etc. shall be grouped and ordered together. The result of ordering is the combination of all the lists provided with descending priority.

The code elements within a file should follow a standard ordering scheme:

a). Elements at the file root level or within a namespace must be positioned in the following order:

1. Extern Alias Directives
2. Using Directives
3. Namespaces
4. Delegates
5. Enums
6. Interfaces
7. Structs
8. Classes

b). Within a class, struct, or interface, elements must be positioned in the following order:

1. Constant Fields
2. Fields
3. Constructors
4. Finalizers (Destructors)
5. Delegates
6. Events
7. Enums
8. Interfaces
9. Properties
10. Indexers
11. Methods
12. Structs
13. Classes

c). Within each of these groups order by access:

1. public
2. internal
3. protected internal
4. protected
5. private

d). Within each of the access groups, order by static, then non-static:

1. static
2. non- static

e). Within each of the static/non-static groups of fields, order by readonly, then non-readonly:

1. readonly
2. non-readonly

f). Within each methods:

1. public static methods
2. public methods
3. internal static methods
4. internal methods
5. protected internal static methods
6. protected internal methods
7. protected static methods
8. protected methods
9. private static methods
10. private methods

**Example:**

Not required

**Rationale:**

Readability and maintainability of the file and encourage code reuse.

### Style\_Format\_003 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

All file headers shall have a revision history which includes correct information about:

1. Version.

2. Date of release/publish.

3. Change description.

\* Note: Revision history should contain appropriate change description whenever a change request is implemented.

**Example:**

Not required.

**Rationale:**

To record the changes of a file during project.

### Style\_Format\_004 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

All method headers shall have a information about:

1. Function name.

2. Purpose.

3. Parameters.

4. Return value.

**Example:**

Not required.

**Rationale:**

Clarification.

### Style\_Format\_005 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Each line in source files shall be limited to 120 columns. If the statement exceeds 120 characters, use a ‘carriage return’ and indent 4 spaces to continue the statement on the next line.

**Example:**

Not required.

**Rationale:**

Readability and compatibility

### Style\_Format\_006 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Tabs shall not be used instead spaces shall be used. Maintain 4 spaces difference between the levels of indentation.

**Example:**

Not required.

**Rationale:**

Different text editors/printers expand tabs to different number of spaces which modifies the original layout of the file.

### Style\_Format\_007 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

A space shall be used between ‘*if’, ‘else if’, ‘while’, ‘for’, ‘switch’* and the opening parenthesis.

**Example:**

// Compliant

if (SpiVar > 10)

while (SpiVar != 0)

// Not compliant

if(SpiVar>10)

while(SpiVar!=0)

**Rationale:**

Readability

### Style\_Format\_008 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Spaces shall not be used between unary operators and operands. Unary operators: ‘++’, ‘--’, ‘~’, ‘!’.

**Example:**

// Compliant

SpiRxCntr++;

// Not compliant

SpiRxCntr ++;

**Rationale:**

Methods to prevent implausible values, execution errors, and errors in the data flow and control flow.

### Style\_Format\_009 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

A space shall be used before and after a binary operator.

**Example:**

// Compliant

Sum = Data + Data2;

// Not compliant

Sum=Data+Data2;

**Rationale:**

Readability

### Style\_Format\_010 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

A space shall be used after each comma in the argument or initialization list.

**Example:**

// Compliant

Sum = UtlSum(Num1, Num2, Num3);

// Not compliant

Sum = UtlSum(Num1,Num2,Num3);

**Rationale:**

Readability

### Style\_Format\_011 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Each expression statement shall be on a separate line.

**Example:**

// Not compliant

// It has two statements in one line

uc\_Sum = uc\_num1 + uc\_num2; uc\_Avg = uc\_Sum / 2;

// Compliant

// It has two statements in two lines

uc\_Sum = uc\_num1 + uc\_num2;

uc\_Avg = uc\_Sum / 2;

**Rationale:**

Simplicity and readability

### Style\_Format\_012 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Braces which enclose a block shall be placed in the same column, on separate lines directly before and after the block. There shall not be anything else on these lines. The block of code inside the opening and closing braces shall be indented 4 spaces. An opening brace is always at the same level as the line above.

Exception:

1. The *‘while’* in the *‘do…while’* statement may be placed one space character after the brace closing the block.

2. The name of the new user type (*‘struct’/’union’/’enum’* type) may be placed one space after the brace closing the element definitions.

3. Array initialization where the opening and closing braces in same line.

**Example:**

// Compliant

a)

if (TRUE == RAM\_CopyData)

{

// if block

}

else

{

// else block

}

b)

void EepCancel()

{

// Function block

}

c)

do

{

// do-while block

} while (Cntr != 0)

// Not compliant

a)

if (TRUE == RAMCopyData)

{

// if block

}

else

{

// else block

}

b)

void EepCancel() {

}

**Rationale:**

Simplicity and readability

### Style\_Format\_013 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

*‘#if’* statements (*‘#if’ – ‘#else’ – ‘#endif’*) should be indented according to their nesting depth.

**Example:**

// Compliant

if (TRUE == RAM\_CopyData)

{

#if ( )

// #if block

#endif

}

// Not compliant

if (TRUE == RAM\_CopyData)

{

#if ( )

// #if block

#endif

}

**Rationale:**

Simplicity and readability

### Style\_Format\_014 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Tabs and unwanted spaces shall not be present at the end of the line in make files.

**Example:**

At the end of line, need remove the space, and tab.

**Rationale:**

To avoid compilation error because of tabs and unwanted spaces at the end of the line in make files.

### Style\_Format\_015 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

The constants shall be place on the left of equality comparisons.

**Example:**

public static readonly uint CONSTANT = 10

uint8 Variable = 0;

// Compliant

if (CONSTANT == Variable)

{

//

}

// Not compliant

if (Variable == CONSTANT)

{

//

}

**Rationale:**

Prefer the use of *“if (CONSTANT == Variable)”* over *“if (Variable == CONSTANT)”* in order to enable the automatic check of a quite common error of *“if (Variable = CONSTANT)”*. The latter represents unwanted assignment and produces unexpected results.

### Style\_Format\_016

**Rule:**

Check that all condition checks have variables of same type on RHS (right hand side) and LHS (left hand side) or there is explicit type casting.

**Example:**

Not required.

**Rationale:**

To avoid illegal access of variables and avoid compile warning.

### Style\_Format\_017

Not applicable.

### Style\_Format\_018

**Rule:**

One blank line should always be used in the following cases:

- Between local variables in a method and its first statement.

- After the closing brace of a code block that is not followed by another closing brace.

- Between logical sections inside a method.

**Example:**

Not required.

**Rationale:**

To improve the readability.

## File Structure

### Style\_File\_001

**Rule:**

The interface file shall have the prefix “I”.

**Example:**

Not required

**Rationale:**

Apply praxis

### Style\_File\_002 ([1] Clause 5.4.7 - table 1 – 1a)

**Rule:**

Code duplication shall be avoided. The use of methods and classes shall be implemented to increase maintainability and portability of source code.

\* Note: Methods and classes that are common shall be placed in a separate file.

**Example:**

Not required

**Rationale:**

- To avoid memory wastage.

- Improve portability and maintainability.

- Readability and understandability.

### Style\_File\_003 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

The source file should end with a blank line.

**Example:**

Not required

**Rationale:**

Avoid errors if the last line of the included file is a comment.

### Style\_File\_004

**Rule:**

If a module provides several functions and processes additional module source files “<Module><Sub>.cs” shall be used. Meaningful Name can be chosen freely for “<Sub>”.

**Example:**

Not required

**Rationale:**

Sub module source files can help to structure a module.

A linker locates only complete objects derived from a file and such an object cannot be subdivided. If more sub files exists linker can work more efficient.

### Style\_File\_005

**Rule:**

Using namespace should not be used back to back. One source file use namespace of another source file which is used back in the first source file.

**Example:**

Not required

**Rationale:**

Not required

### Style\_File\_006

Not applicable.

## Comments

### Style\_Comment\_001 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Multi-line comment blocks shall have the opening and closing character sequences in separate lines. There shall not be anything else on these lines, but optional star frames.

**Example:**

//

\* This is a multi-line comment block line 1

\* This is a multi-line comment block line 2

\*/

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

\* This is a multi-line comment block with a star frame \*

\* This is a multi-line comment block with a star frame \*

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

**Rationale:**

Readability and uniformity

### Style\_Comment\_002 ([1] Clause 5.4.7 - table 1 - 1g)

Not applicable.

### Style\_Comment\_003 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

For the following elements, suitable comments shall be added:

- Namespace (for what is it needed?).

- Definition of class.

- Elements of classes, structures and unions.

- Definition of variables.

- Definition of constants.

- Definition of methods.

**Example:**

Not required

**Rationale:**

Uniformity and readability

### Style\_Comment\_004 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

The following C Sharp constructs shall be commented after the concluding bracket if they are longer than 10 lines:

*‘switch-case’*

*‘if-then-else’*

*‘do-while’*

*‘for’*

**Example:**

if (a < b)

{

//

More than 10 lines

} // end of if (a < b)

**Rationale:**

Readability

### Style\_Comment\_005 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Constructs and compiler switches shall be commented at the end of block (e.g. after the concluding bracket, ‘*#endif’*, etc) if they are longer than 10 lines.

**Example:**

1)

// Compliant

if (a < b)

{

//

More than 10 lines

} // end of if (a < b)

2)

// Compliant

#if (CONDITION == TRUE)

//

more than 10 lines

\*/

#endif // (CONDITION == TRUE)

**Rationale:**

Readability

### Style\_Comment\_006 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Comments shall not be nested.

**Example:**

// Not compliant

// Comment // this is a Nested comment //another nested comment

**Rationale:**

Nested comments may create problems with some compilers.

### Style\_Comment\_007 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

A comment should be placed immediately above and indented same as the line/block of code which it is complementary.

**Example:**

// Compliant

// Check CONDITION

if (CONDITION)

{

……

**Rationale:**

Readability and understandability.

# Coding Rules

## Conformance to external rules

### Rules\_Ext\_001

**Rule:**

If a Coverity issue has occurred, analyse the issue to know if that error can be avoided by modifications in the code. Or that violation of Coverity rule shall be commented and reasoned at the corresponding code line.

**Example:**

Not required

**Rationale:**

Not required

### Rules\_Ext\_002

Not applicable.

### Rules\_Ext\_003

**Rule:**

In general, use *int* rather than unsigned types.

**Example:**

Not required

**Rationale:**

The use of *int* is common throughout C#, and it is easier to interact with other libraries when you use *int*.

## Environment

### Rules\_Envr\_001 ([1] Clause 5.4.7 - table 1 - 1d)

Not applicable.

### Rules\_Envr\_002 ([1] Clause 5.4.7 - table 1 - 1d)

Not applicable.

### Rules\_Envr\_003 ([1] Clause 5.4.7 - table 1 - 1d)

**Rule:**

While using object/library files (from third party, customer, etc.) it shall be confirmed that it was produced with same compilers and version which is used in the project.

**Example:**

Not required

**Rationale:**

Correct integration

### Rules\_Envr\_004

**Rule:**

There should not be any compiler and linker warnings (of-course errors also) while building the software.

**Example:**

Not required

**Rationale:**

Warnings are meant to be corrected.

### Rules\_Envr\_005 ([1] Clause 5.4.7 - table 1 - 1d)

Not applicable.

### Rules\_Envr\_006

Not applicable.

## Comments/Documentation

### Rules\_Comment\_001 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

All comments shall be in English and spell checked.

**Example:**

Not required

**Rationale:**

Readability

### Rules\_Comment\_002 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

The purpose of every line of executable code should be explained by a comment, although one comment may describe more than one source line of code.

**Example:**

Not required

**Rationale:**

Readability

### Rules\_Comment\_003 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

One should avoid stating in comments what is better stated in code.

**Example:**

// Not compliant

// Add ‘CAN\_Data2’ to ‘CAN\_Data1’

CAN\_Data1 += CAN\_Data2;

**Rationale:**

Redundant comments are unnecessary and increase the maintenance effort.

### Rules\_Comment\_004 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Commented out code shall not be left permanently in source files. Any lines or blocks of code that exists specifically for debugging purpose shall be put in method or class with Conditional attribute.

**Example:**

// Code for debugging purpose shall be put in conditional methods as following

[Conditional("DEBUG")]

public void PrintDebugInformation()

{

}

**Rationale:**

Readability. Redundant comments are unnecessary and increase the maintenance effort.

### Rules\_Comment\_005 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

File/Methods/Classes header format should be consistent across the code.

**Example:**

a) Methods comment

/// <name>MyMethod</name>

/// <summary>

/// Short Description, including preconditions

/// </summary>

/// <param name="VariableA">

/// Parameter Description

/// </param>

/// <returns>None</returns>

/// <remarks>

/// Global variables used, side effects

/// </remarks>

/// <ref>TSDD001, TSDD002</ref>

public void MyMethod(uint X)

{

}

b) Class comment

/// <name>MyClass</name>

/// <summary>

/// Short Description, including preconditions

/// </summary>

/// <remarks>

/// Global variables used, side effects

/// </remarks>

/// <ref>TSDD001, TSDD002</ref>

public class myClass

{

...

}

**Rationale:**

Readability and understandability.

### Rules\_Comment\_006

**Rule:**

Each cast statement shall have an associated comment describing the reason and behaviour.

**Example:**

// Compliant

// Method prototype

sbyte ExampleFooBar(sbyte arg);

ushort num;

sbyte result;

// Only the first byte of variable ‘num’ is necessary

result = ExampleFooBar((sbyte)num);

**Rationale:**

Readability and understandability

### Rules\_Comment\_007

**Rule:**

Block comments are used to provide descriptions of files, methods, data structures and algorithms.

Block comments may be used at the beginning of each file. They can also be used in other places, such as within methods.

Block comments inside a function or method should be indented to the same level as the code they describe.

A blank line to set it apart from the rest of the code should precede a block comment.

A block comment shall be described as following template.

///

// Description 1

// Description 2

// Description 3

///

**Example:**

// Compliant

///

// Read the data from the container

// then update to database.

///

**Rationale:**

Readability and understandability

### Rules\_Comment\_008

**Rule:**

Short comments can appear on a single line indented to the level of the code that follows.

If a comment can't be written in a single line, it should follow the block comment format.

A single-line comment should be preceded by a blank line.

Single line comments must not use documentation style slashes.

Single line comments must not be followed by blank line.

Trailing comments must not be used.

**Example:**

// Compliant

if (condition)

{

// Handle the condition.

...

}

// Not Compliant

a) An illegal comment beginning with three slashes:

/// Trim the name.

fullName = fullName.Trim();

b) A line of commented-out code beginning with four slashes:

////fullName = asfd;

return fullName

c). Trailing comments must not be used

return isPrime(a); // Works only for odd a

**Rationale:**

Readability and understandability

## Identifiers

### Rules\_Identifier\_001 ([1] Clause 5.4.7 - table 1 - 1b)

Not applicable.

## Types

### Rules\_Types\_001 ([1] Clause 5.4.7 - table 1 - 1c)

**Rule:**

Enumeration types must be used instead of integer types and constants to select from a limited series of choices.

**Example:**

// Compliant

public enum CanHwFaultStatus

{

CAN\_HW\_NO\_FAULT,

CAN\_HW\_SHORT\_TO\_GND,

CAN\_HW\_SHORT\_TO\_VCC,

};

// Not compliant

public const sbyte CAN\_HW\_NO\_FAULT = (sbyte)0

public const sbyte CAN\_HW\_SHORT\_TO\_GND = (sbyte )1

public const sbyte CAN\_HW\_SHORT\_TO\_VCC = (sbyte )2

**Rationale:**

Enhances debugging, readability and maintenance.

Compiler or static check tool can be set to generate a warning when the *‘enum’* type variable is used in *‘switch’* statement and all enumerators are not used as case.

### Rules\_Types\_002

**Rule:**

Integer values of the enumeration elements must not be used in calculations.

**Example:**

typedef enum Element

{

ELEMENT\_1,

ELEMENT\_2,

ELEMENT\_3

};

// Not compliant

Variable = Element .ELEMENT\_1 + 3;

**Rationale:**

The integer value of an enumeration element can be changed when another element is added at later point of time during development.

### Rules\_Types\_003 ([1] Clause 5.4.7 - table 1 - 1h)

Not applicable.

### Rules\_Types\_004

**Rule:**

Variables for loop counters must be declared in the generic type (e.g. *‘int’*).

**Example:**

Not required

**Rationale:**

The variables declared in generic type could be handled best by all of the compilers. Therefore loop counters, in view of reuse of software, have to be declared in the generic type.

### Rules\_Types\_005

**Rule:**

Do not cast types where a loss of precision is possible. Only implement casts that operate on the complete object from a base type to a derived type.

**Example:**

// Compliant

// Create a new derived type

Dog dog = new Dog();

// A base type can keep a derived type safely

Animal animal = dog;

// Cast back to derived type

Dog dog2 = (Dog)animal;

// Not compliant

double d = 3.234;

int number = (int)d;

**Rationale:**

Not required.

## Constants

### Rules\_Const\_001 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Hexadecimal constants shall be represented using all uppercase letters following 0x.

**Example:**

// Compliant

public string RamStartAddr 0x7AFEu

// Not compliant

public string RamStartAddr 0x7afeu

**Rationale:**

Readability

### Rules\_Const\_002 ([1] Clause 5.4.7 - table 1 - 1a)

**Rule:**

Numeric values shall not be used in code; symbolic values shall be used instead.

Exception:

1. 0 in initialization of variables.

2. 1 in bit operations.

3. Constants used in generic math expressions like 2 in calculating the average value of two numbers.

**Example:**

// Compliant

int average = 0;

average = (a + b) / 2;

**Rationale:**

Readability and maintenance

### Rules\_Const\_003

**Rule:**

All SOFTWARE Modules shall indicate all global data with read-only purposes by explicitly assigning the *‘const’* or “readonly” keyword depend on specific cases.

* Using “const” if the data has to be initialized at declaration.
* Using “readonly” if the data can be initialized at declaration or constructor.

**Example:**

// Compliant

class ConfigurationA

{

public const uint MAX\_NUMBER = 100;

public readonly string ModuleName;

// In constructor method, ModuleName is initialized

ConfigurationA()

{

}

}

**Rationale:**

The variable shall be initialized one time only and should not be changeable after the initialization.

### Rules\_Const\_004

**Rule:**

Use all uppercase when naming constant. An underscore can be used to separate terms when necessary.

**Example:**

const MIN\_QUAL = 25

**Rationale:**

To clearly distinguish constants from other elements

## Declarations and Definitions

### Rules\_Defn\_Decl\_001 ([1] Clause 8.4.4 - table 8 - 1e)

**Rule:**

Global variables must be avoided or its usage needs to be justified.

**Example:**

// Compliant

class Program

{

/// <summary>

/// This function update the global variable “GlobalVar.GlobalValue” with value of parameter.

/// </summary>

/// <param name="parameter">The parameter</param>

void DoSomething(int parameter)

{

// Set global integer.

GlobalVar.GlobalValue = parameter;

}

}

**Rationale:**

The module which uses the global variable can update mistakenly and it is difficult to debug.

### Rules\_Defn\_Decl\_002

Not applicable.

### Rules\_Defn\_Decl\_003 ([1] Clause 5.4.7 - table 1 - 1a)

Not applicable.

### Rules\_Defn\_Decl\_004 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Multiple variable declarations shall not be allowed on the same line.

**Example:**

// Compliant

uint CanRxData;

uint CanTxData;

// Not compliant

uint CanRxData, CanTxData;

**Rationale:**

Readability

### Rules\_Defn\_Decl\_005 ([1] Clause 5.4.7 - table 1 - 1g)

Not applicable.

### Rules\_Defn\_Decl\_006

Not applicable.

### Rules\_Defn\_Decl\_007

Not applicable.

### Rules\_Defn\_Decl\_008

Not applicable.

### Rules\_Defn\_Decl\_009

Not applicable.

### Rules\_Defn\_Decl\_010

Not applicable.

### Rules\_Defn\_Decl\_011

**Rule:**

Declaration and definition of local methods shall have *‘private’*.

Local function means function with internal usage inside a class only.

**Example:**

private void MyLocalFunction(void);

**Rationale:**

Limit visibility of local functions.

### Rules\_Defn\_Decl\_012 ([1] Clause 5.4.7 - table 1 – 1d)

**Rule:**

Variables used in the code need to be defined; moreover, unused variables should not be contained.

**Example:**

Not required

**Rationale:**

In order to prevent redundancy and compiler warnings.

### Rules\_Defn\_Decl\_013 ([1] Clause 5.4.7 - table 1 – 1d)

**Rule:**

Unused input parameter(s) of function shall not be declared.

**Example:**

Not required

**Rationale:**

To avoid redundancy and compiler warnings.

### Rules\_Defn\_Decl\_014

Not applicable.

### Rules\_Defn\_Decl\_015 ([1] Clause 5.4.7 - table 1 - 1a)

Not applicable.

### Rules\_Defn\_Decl\_016([1] Clause 5.4.7 - table 1 - 1a)

**Rule:**

The common implementations across the modules such as initialization of status of the driver should be consistent across all the modules.

**Example:**

// Defined values for compile switch conditions in all modules are “STD\_ON” and “STD\_OFF”.

// Compliant

// Module DIO

if (DioChannelConfigured == Constant.STD\_ON)

if (DioChannelGroupConfigured == Constant.STD\_ON)

// Module MCU

if (McuErrorDetected == Constant.STD\_ON)

if (McuClockEnabled == Constant .STD\_OFF)

**Rationale:**

Uniformity and easy understanding.

### Rules\_Defn\_Decl\_017 ([1] Clause 5.4.7 - table 1 - 1a)

**Rule:**

The variables and macros (used in different modules but similar context), shall be named similarly. Consistency should be maintained across modules.

**Example:**

Name of the status variable can be consistent across all modules, e.g.

- ADC module: ‘AdcGblDriverStatus’.

- GPT module: ‘GptGblDriverStatus’.

- PWM module: ‘PwmGblDriverStatus’.

**Rationale:**

Uniformity and easy understanding.

### Rules\_Defn\_Decl\_018 ([1] Clause 5.4.7 - table 1 - 1c)

**Rule:**

The type of variable should be appropriately selected depending on the values that the variable may hold.

**Example:**

For example, if the variable can hold 'INIT', 'UNINIT' and 'BUSY', then enum or integer needs to be given as type instead of boolean.

**Rationale:**

To avoid unexpected behaviour.

### Rules\_Defn\_Decl\_019 ([1] Clause 5.4.7 - table 1 - 1c)

**Rule:**

Typecasts and the values assigned to variables shall be correctly defined and shall be consistent across the modules.

**Example:**

Not required

**Rationale:**

Uniformity

### Rules\_Defn\_Decl\_020 ([1] Clause 8.4.4 - table 8 - 1d)

**Rule:**

No multiple uses of variable names or similar names but different purpose and usage.

**Example:**

// Not compliant

// Use the “Counter” and “counter” for two variable but different usage.

class Program

{

private int Counter;

void DoSomething()

{

int counter = 0;

…

for (counter = 0; counter < X; counter++)

}

}

**Rationale:**

To avoid errors in implementation.

### Rules\_Defn\_Decl\_021

Not applicable.

### Rules\_Defn\_Decl\_022

Not applicable.

### Rules\_Defn\_Decl\_023

Not applicable.

### Rules\_Defn\_Decl\_024

**Rule:**

Variables have been defined in as small scope as possible.

If it can be defined locally in a method, shouldn’t define it as a field of class.

If it can be defined as a private field of class, shouldn’t define it as a public field.

**Example:**

// Compliant

// variable “comparedValue” is used in DoSometing() only.

class Program

{

void DoSomething()

{

int comparedValue = 0;

…

}

}

// Not Compliant

// variable “ComparedValue” is used in DoSometing() only.

class Program

{

private int ComparedValue;

void DoSomething()

{

…

}

}

**Rationale:**

Performance and to avoid illegal access.

### Rules\_Defn\_Decl\_025

**Rule:**

The variables should have only one purpose. Should not have multiple purpose.

**Example:**

// Not compliant

void DoSomething(int a, int b)

{

int sumValue = a + b;

// “sumValue” is used for a second purpose

sumValue = GetCurrentIndex();

}

**Rationale:**

Readability and to avoid illegal access.

### Rules\_Defn\_Decl\_026

**Rule:**

The Type and size of the variables are suited to its intended use.

**Example:**

// Not compliant

double MiliSecond;

// Compliant

ulong MiliSecond;

**Rationale:**

Readability and to avoid data casting, invalid values.

### Rules\_Defn\_Decl\_027

**Rule:**

Put declarations only at the beginning of blocks. (A block is any code surrounded by curly braces "{" and "}".) Don't wait to declare variables until their first use.

**Example:**

// Compliant

public void SomeMethod()

{

// Beginning of method block.

int int1 = 0;

…

if (condition)

{

// Beginning of "if" block.

int int2 = 0;

}

}

**Rationale:**

To avoid confusing the unwary programmer and hamper code portability within the scope.

### Rules\_Defn\_Decl\_027

**Rule:**

Use a public static read-only field to define predefined object instances.

**Example:**

public struct Color

{

public static readonly Color Red = new Color(0xFF0000);

public Color(int rgb)

{

// implementation

}

}

**Rationale:**

To avoid confusing the unwary programmer and hamper code portability within the scope.

## Initialization

### Rules\_Init\_001 ([1] Clause 8.4.4 - table 8 - 1c)

Not applicable.

### Rules\_Init\_002

**Rule:**

Try to initialize local variables where they're declared. The only reason not to initialize a variable where it's declared is if the initial value depends on some computation occurring first.

**Example:**

// Compliant

int int1 = 0;

string EmployeeName = string.Empty;

**Rationale:**

To avoid initialisation of variables with garbage values and for readability.

## Control Statement Expressions

### Rules\_Expr\_001 ([1] Clause 5.4.7 - table 1 – 1d)

**Rule:**

Brackets shall always be used in complex expressions even if the C Sharp priority rules do not necessarily demand this for operators. This also applies to expressions evaluated by the pre-processor.

**Example:**

// Compliant

if ((counter1 > 0) || (counter2 < 0))

#if ((SecSecurity == C) || (SecSecurity Class== CCC))

**Rationale:**

Readability

### Rules\_Expr\_002 ([1] Clause 5.4.7 - table 1 – 1a)

**Rule:**

Control structure nesting should be minimized and not exceed 6 levels. The number of levels shall be customizable. That means that it shall be possible to reduce the allowed number of levels per file.

**Example:**

Not required

**Rationale:**

Less complex and easy understanding.

### Rules\_Expr\_003 ([1] Clause 5.4.7 - table 1 – 1d)

**Rule:**

Multiple assignments shall not be done.

**Example:**

// Declarations

int AdcVarX;

int AdcVarY;

int AdcVarZ;

// Compliant

AdcVarX = AdcVarY;

AdcVarY = AdcVarZ;

// Not compliant

AdcVarX = AdcVarY = AdcVarZ;

**Rationale:**

Readability

### Rules\_Expr\_004 ([1] Clause 5.4.7 - table 1 - 1d)

**Rule:**

The use of ‘++’ and ‘--‘ should be limited to simple cases. They shall not be used in statements where other operators occur. The prefix use is always forbidden.

**Example:**

// Variable declaration

int LusVarX;

int AdcGusVarY;

// Not compliant

LusVarX -= AdcGusVarY++;

**Rationale:**

Readability and maintenance

### Rules\_Expr\_005 ([1] Clause 5.4.7 - table 1 - 1e)

**Rule:**

While performing masking operation, double check the mask value to be used and its impact for other functionalities.

**Example:**

Not required

**Rationale:**

To avoid bugs in implementation.

### Rules\_Expr\_006

**Rule:**

Check for wrong usage of “&&", "||" while performing mask operations in the CODE.

**Example:**

Not required

**Rationale:**

To avoid bugs in implementation.

### Rules\_Expr\_007 ([1] Clause 5.4.7 - table 1 - 1d)

**Rule:**

If the division operation is being performed in the embedded code then make sure to avoid condition of illegal division by zero.

**Example:**

// Compliant

if (c != 0)

{

a = b / c;

}

**Rationale:**

To avoid hanging of code while execution.

### Rules\_Expr\_008

**Rule:**

During updating for similar part of code (e.g. timer unit/channel interrupt handlers), check the code (e.g. unit/channel numbers) carefully, because there can be typo mistakes in updating correct values of those numbers.

**Example:**

Not required

**Rationale:**

To avoid bug in implementation.

### Rules\_Expr\_009

**Rule:**

Try to avoid comparison of RAM variable in the loops to reduce the CPU load.

**Example:**

// Not compliant

for (int loopCounter = 0; loopCounter < (uint8\_least)LpRunTimeData->ucChannelCount; loopCounter++)

{

}

In the above example, the comparison is made exactly the same number of time as there are channels configured. But RAM access is slower than register access, so use a local variable that should hold the channel count. Channel count should be read once before loop execution (a single memory access). Using a local variable the compiler will assign an internal register to the variable. This way the *‘for’* loop comparison operator will be translated as a register to register comparison - avoiding undesired and slow RAM access.

// Compliant

int maxLoop = LpRunTimeData->ucChannelCount;

for (int loopCounter = 0; loopCounter < maxLoop; loopCounter++)

{

}

**Rationale:**

To reduce the execution time.

### Rules\_Expr\_010

**Rule:**

Considering integer boundaries (overflows & underflows) while handling arithmetic operation.

**Example:**

- Overflow of unsigned integers:

32 bits integer types can hold certain ranges of values.

So if we have two unsigned integer types each with the value of 2147483648 (a & b):

a + b = 4294967296

which is larger than the maximum value that can be represented in an unsigned integer type. This is called an integer overflow.

- Underflow of unsigned integers:

unsigned int a, b;

a = 0

b = a - 1

The value of b is -1, which is below than the minimum possible value that can be stored this is called an integer underflow.

- Overflow/Underflow of Signed Integers:

The signed integer two’s compliment representation in binary will have value, padding, & sign bits. The sign bit represents the sign of the integer 0 for positive and 1 if the number is negative. When an overflow or underflow condition occurs on signed integers the result will wrap around the sign and causes a change in sign.

For example:

A 32 bit number 2147483647 = 0x7FFFFFFF in hex.

If we add 1 to this number it will be 0x80000000 which is equivalent to -2147483648 decimal.

With signed addition or subtraction, you can overflow the sign boundary by causing a positive number to wrap around 0x80000000 and become a negative number. You can also underflow the sign boundary by causing a negative number to wrap below 0x80000000 and become a positive number.

**Note: When result of arithmetic operation is stored in HW registers, the maximum value which register can hold to be considered before loading result of arithmetic operation into a register. If result value exceeds the limit which register can hold, work around to be carried out before loading result value into a register.**

**Rational:**

A numeric overflow or underflow that occurs early in a block of code can lead to a subtle series of cascading faults; not only is the result of a single arithmetic operation tainted, but every subsequent operation using that tainted result introduces a point where an attacker might have unexpected influence.

### Rules\_Expr\_011 ([1] Clause 8.4.4 - table 8 – 1b)

Not applicable.

### Rules\_Expr\_012 ([1] Clause 5.4.7 - table 1 - 1c)

**Rule:**

Type conversions of the variables shall be done explicitly and correctly. It shall not be implicit.

**Example:**

1)

// Function definition of foo

public void foo(uint val);

void func(void)

{

uint uiValue;

short shrtValue;

int intValue;

// Compilant

foo(uiValue);

// Not compliant

foo(shrtValue);

foo(intValue);

}

2)

// Compliant

(short)0xFFFF

(int)0x10000

// Not compliant

(short)0x10000

**Rationale:**

Implicit type conversion may lead to data loss and incorrect type casting of variables may cause bugs in implementation.

### Rules\_Expr\_013

Not applicable.

## Control Flow

### Rules\_CtrlFlow\_001 ([1] Clause 5.4.7 - table 1 - 1d)

Not applicable

### Rules\_CtrlFlow\_002 ([1] Clause 5.4.7 - table 1 - 1d)

**Rule:**

Assignments shall not be made within an *‘if’* or *‘else if’* or loop controlling expression (except for a single loop counter initialization in the first clause of a *‘for’* statement).

**Example:**

1)

// Compliant

ret = FooBar(arg1);

if (ZERO == ret)

// Not compliant

if (ZERO == (ret = FooBar(arg1)))

2)

// Compliant

// loop counter initialization for ‘for’ statement

for (counter = 0; counter < MAX\_COUNT; counter++)

**Rationale:**

To avoid unexpected control flows.

### Rules\_CtrlFlow\_003 ([1] Clause 5.4.7 - table 1 - 1a)

**Rule:**

Cyclomatic complexity numbers of all functions shall be minimized and be limited for each project (e.g. according McCabe, the cyclomatic complexity shouldn’t be higher than 15).

**Example:**

Not required

**Rationale:**

Too complex functions are difficult to read, test and maintain.

### Rules\_CtrlFlow\_004 ([1] Clause 5.4.7 - table 1 – 1a)

**Rule:**

*‘break’* and *‘continue’* commands should never be used in loops except under very rare circumstances when there is no other option.

**Example:**

// Comliant

while (nX < 10)

{

nX++;

}

// Not compliant

while (TRUE)

{

nX++;

if (nX == 10)

{

break;

}

}

**Rationale:**

- To avoid endless loops.

- To avoid bugs in implementation.

### Rules\_CtrlFlow\_005 ([1] Clause 5.4.7 - table 1 - 1d)

**Rule:**

Any *‘if’* statement shall end with an *‘else’* clause.

**Example:**

// Not compliant

if (MOD\_GetIndex() > 42u)

{

// Do something

}

// Compliant

if (MOD\_GetIndex() > 42u)

{

// Do something

}

else

{

// No action required

}

**Rationale:**

Understandably

### Rules\_CtrlFlow\_006 ([1] Clause 5.4.7 - table 1 - 1d)

**Rule:**

Avoid endless loop conditions in the source code. Any loop should also have a condition to exit at some points of time by using some count values which are appropriate for the functionality of that loop.

**Example:**

// Not compliant

while (true)

{

// …

}

**Rationale:**

The code may hang if the break statement isn’t executed.

### Rules\_CtrlFlow\_007

**Rule:**

The *‘break’* should be between each *‘case’* statement, unless the *‘case’* statements need to be executed in sequence in switch statements (mentioned exception is allowed with regard to faster processing speed, higher code efficiency, etc.).

\* Note: All switch statements shall have the default case implemented, even if it is left empty apart from the break statement.

**Example:**

Not required

**Rationale:**

Unnecessary code execution may lead to wrong functionality.

### Rules\_CtrlFlow\_008 ([1] Clause 5.4.7 - table 1 - 1e)

Not applicable.

### Rules\_CtrlFlow\_009 ([1] Clause 8.4.4 - table 8 - 1h)

**Rule:**

No hidden data flow or control flow except for call back function defined by SOFTWARE. “No hidden data flow or control flow” are meant to prohibit a control flow from processing multiple meanings. This is the rule to control the increase in risk of occurrence of problems by implementing a control which is not recognized by a designer.

**Example:**

Example of hidden control flow is shown below:

Variable ‘X’ in a function, created by an individual A, is used as a condition for assigning a variable ‘Z’ with no dependency within a function B, created by an individual B.

Preconditions:

- Global variables: X, Y.

- Function A and function B may be executed alternately.

- There is no dependency between variable "Z" in function B and global variable "X".

Function A created by an individual A:　　　　　　　 Function B created by an individual B:

(Hidden control flow)

Function A

Y++

**X** > 0xFF

**X**++

Exit

Yes

No

Function B

Z = 0x00

**X** > 0x10

Exit

Yes

No

**Rationale:**

To avoid errors in implementation.

### Rules\_CtrlFlow\_010 ([1] Clause 8.4.4 - table 8 - 1i)

**Rule:**

No unconditional jumps. Unconditional jumps like *‘goto’* shall not be used.

**Example:**

// Not compliant

do

{

if (a == 15)

{

// skip the iteration

a = a + 1;

goto LOOP;

}

a++;

}

while (a < 20);

// Compliant

do

{

if (a == 15)

{

// skip the iteration

a = a + 1;

}

else

{

a++;

}

}

while (a < 20);

**Rationale:**

To avoid errors in implementation.

### Rules\_CtrlFlow\_011

**Rule:**

When you use "switch", "while", "do-while", "for" statement, even if you can write one sentence, writing enclosed within the brace { }.

**Example:**

// Not compliant

if(a == 15)

a = a + 1;

// Compliant

if(a == 15)

{

a = a + 1;

}

**Rationale:**

To maintain the description of the consistency.

### Rules\_CtrlFlow\_012

**Rule:**

When use “if-else” or “switch”. All conditions should be handled.

**Example:**

Not required.

**Rationale:**

To avoid errors in implementation.

### Rules\_CtrlFlow\_013

**Rule:**

There shall be no unreachable code.

e.g. delete the branch code by impossible conditions

**Example:**

Not required.

**Rationale:**

To avoid errors in implementation.

### Rules\_CtrlFlow\_014

**Rule:**

Use a try-catch statement for most exception handling.

**Example:**

// Compliant

static string GetValueFromArray(string[] array, int index)

{

try

{

return array[index];

}

catch (System.IndexOutOfRangeException ex)

{

Console.WriteLine("Index is out of range: {0}", index);

}

}

// Not Compliant

Not required.

**Rationale:**

To avoid errors in implementation

## Functions

### Rules\_Func\_001 ([1] Clause 5.4.7 - table 1 - 1d)

**Rule:**

Functions that do not return a parameter (or return one fixed value only) shall be declared as *‘void’* type.

**Example:**

// Compliant

void CtlSystemCheck()

{

}

// Not compliant

int CtlSystemCheck()

{

// only “1” is returned

return 1;

}

**Rationale:**

Eliminates functions with the implicit return type of *‘int’* and improves readability.

### Rules\_Func\_002 ([1] Clause 5.4.7 - table 1 - 1d)

Not applicable.

### Rules\_Func\_003 ([1] Clause 5.4.7 - table 1 - 1d)

**Rule:**

If a function returns a value, the calling code should check for an absence of errors.

**Example:**

Check that if the called function returns data of appropriate type or not.

**Rationale:**

To ensure the called functionality is executed correctly.

### Rules\_Func\_004 ([1] Clause 8.4.4 - table 8 - 1a)

**Rule:**

One entry and one exit point in subprograms and functions. The way to invoke one function should be same and consistent. The return for exit should be one in a subprogram and a function.

**Example:**

// Not compliant

uint Compare(uint uiVal1, uint uiVal2)

{

if(uiVal1 > uiVal2)

{

return uiVal1;

}

else

{

return uiVal2;

}

}

// Compliant

uint Compare(uint uiVal1, uint uiVal2)

{

uint uiReturnValue = 0;

if(uiVal1 > uiVal2)

{

uiReturnValue = uiVal1;

}

else

{

uiReturnValue = uiVal2;

}

return uiReturnValue;

}

**Rationale:**

Readability, maintainability and avoidance of invalid operations.

### Rules\_Func\_005 ([1] Clause 8.4.4 - table 8 - 1j)

**Rule:**

No recursions. Recursive functions shall not be used.

**Example:**

// Not compiliant

void recurse()

{

recurse();

}

**Rationale:**

There can be stack overflow, in case of recursive functions.

### Rules\_Func\_006

Not applicable.

### Rules\_Func\_007

Not applicable.

## Pointers and Arrays

### Rules\_Ptr\_001 ([1] Clause 8.4.4 - table 8 - 1f)

Not applicable.

### Rules\_Ptr\_002 ([1] Clause 5.4.7 - table 1 - 1d)

Not applicable.

### Rules\_Ptr\_003 ([1] Clause 5.4.7 - table 1 - 1d)

Not applicable.

### Rules\_Ptr\_004 ([1] Clause 5.4.7 - table 1 - 1d)

Not applicable.

### Rules\_Ptr\_005 ([1] Clause 5.4.7 - table 1 - 1d)

Not applicable.

## Structures and Unions

### Rules\_Struct\_001 ([1] Clause 5.4.7 - table 1 - 1b)

Not applicable.

## Pre-processing Directives

### Rules\_PreProcess\_001 ([1] Clause 5.4.7 - table 1 - 1g)

Not applicable.

### Rules\_PreProcess\_002

Not applicable.

### Rules\_PreProcess\_003

Not applicable.

### Rules\_PreProcess\_004

Not applicable.

### Rules\_PreProcess\_005 ([1] Clause 5.4.7 - table 1 - 1g)

Not applicable.

### Rules\_PreProcess\_006 ([1] Clause 5.4.7 - table 1 - 1a)

Not applicable.

### Rules\_PreProcess\_007 ([1] Clause 5.4.7 - table 1 - 1a)

Not applicable.

### Rules\_PreProcess\_008 ([1] Clause 5.4.7 - table 1 - 1a)

Not applicable.

### Rules\_PreProcess\_009

Not applicable.

### Rules\_PreProcess\_010 ([1] Clause 5.4.7 - table 1 - 1g)

Not applicable.

### Rules\_PreProcess\_011 ([1] Clause 8.4.4 - table 8 - 1g)

Not applicable.

### Rules\_PreProcess\_012

Not applicable.

## Optimization

### Rules\_Opt\_001 ([1] Clause 5.4.7 - table 1 – 1a)

Not applicable.

### Rules\_Opt\_002 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

Combine equivalent constants, however, this is at the cost of readability. Appropriate comments should be provided where necessary.

**Example:**

// Compliant

A = B | CONST1;

A = A & CONST2;

// Can be

A = ((B | CONST1) & CONST2)

**Rationale:**

Performance

### Rules\_Opt\_003 ([1] Clause 5.4.7 - table 1 - 1g)

**Rule:**

The condition of most probable occurrence should be placed in the *‘if’* rather than in the *‘else’*.

**Example:**

Not required

**Rationale:**

Performance

### Rules\_Opt\_004 ([1] Clause 5.4.7 - table 1 – 1a)

**Rule:**

Optimize ‘|’ (or) and ‘&’ (and) expressions: An ‘|’ or ’&’ query should be held as short as possible:

- At an ‘|’ query, put the most probable case in the first statement.

- At an ‘&’ query, put the most improbable case in the first statement.

- For the query of the same variables use a mask.

**Example:**

// Given that variable ‘stNxt’ and all macros in below example, each has a value with either all bits turned off or only 1 bit turned on.

// Compliant

// Optimization with a mask

private bool SyncAllBit = (SyncTimeout | SyncWaitInc | SyncResyncOffset | SyncPhasePlausChk | SyncPhasePlausSecondChk)

if ((stNxt & SyncAllBit ) != 0)

{

...

}

// Not compliant

// CPU has to go through all conditions to check if any of them is true

if ((stNxt == SyncTimeOut) || (stNxt == SyncWaitInc ) || (stNxt == SyncResyncOffset ) || (stNxt == SyncPhasePlausChk ) || (stNxt SyncPhasePlausSecondChk ))

{

...

}

**Rationale:**

Performance

### Rules\_Opt\_005

Not applicable.

### Rules\_Opt\_006

Not applicable.

### Rules\_Opt\_007

Not applicable.

### Rules\_Opt\_008

Not applicable.

## Events

### Rules\_Event\_001

**Rule:**

Do not make assumptions on the application state after raising an event.

**Example:**

No required.

**Rationale:**

The even handler can change the application state or call another method that changes application state.

### Rules\_Event\_002

**Rule:**

Always document from which thread an event handler is called. And the event handler must synchronize (ensure thread-safety) access to shared data.

**Example:**

No required.

**Rationale:**

No required.

### ptRules\_Event\_003

**Rule:**

Raise events through a protected virtual method.

**Example:**

No required.

**Rationale:**

If a derived class wants to intercept an event, it can override such a virtual method, do its own work, and then decide whether or not to call the base class version.

## Object Oriented Programming

### Rules\_Oop\_001

**Rule:**

Provide a default private constructor if there are only static methods and properties on a class.

Explicitly define a protected constructor on an abstract base class.

**Example:**

No required.

**Rationale:**

No required.

## Except Handling

### Rules\_Expt\_001

**Rule:**

Only throw exceptions in exceptional situations. In general, try to design classes that do not throw exceptions in the normal flow of control.

**Example:**

No required.

**Rationale:**

No required.

### Rules\_Expt\_002

**Rule:**

Only re-throw exceptions when you want to specialization the exception.

**Example:**

No required.

**Rationale:**

No required.

### Rules\_Expt\_002

**Rule:**

Never do a catch exception and do nothing.

**Example:**

// Not compliant

string ReadFromFile (string fileName)

{

try

{

// Read from file.

}

catch (Exception ex)

{

// Catching general exception is bad... we will never know whether it

// was a file error or some other error.

// Here you are hiding an exception.

// In this case no one will ever know that an exception happened.

return null;

}

}

**Rationale:**

If hide an exception, we will never know if the exception happened or not.

# Project Settings and Structure

## Project Setting

### Rules\_Project\_001

**Rule:**

Allways build the project with warning level 4 and release project without warning or errors in build.

**Example:**

No required.

**Rationale:**

All potential issues have to be raised and resolved.

### Rules\_Project\_002

**Rule:**

Never suppress specific compiler warnings in project setting.

**Example:**

No required.

**Rationale:**

Don’t hide any potential issues.

### Rules\_Project\_003

**Rule:**

Always explicitly state your supported runtime versions in the application configuration file.

All .NET Framework v4.x versions specify the v4.0 for "runtime version” values.

**Example:**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

</startup>

</configuration>

**Rationale:**

Don’t hide any potential issues.

### Rules\_Project\_003

**Rule:**

Avoid explicit preprocessor definitions (#define). Use the project settings for defining conditional compilation constants.

**Example:**

No required.

**Rationale:**

No required.

### Rules\_Project\_004

**Rule:**

Do not put any logic inside AssemblyInfo.cs.

Do not put any assembly attributes in any file besides AssemblyInfo.cs.

Populate all fields in AssemblyInfo.cs such as company name, description, copyright notice.

**Example:**

No required.

**Rationale:**

No required.

### Rules\_Project\_005

**Rule:**

Avoid explicit code exclusion of method calls (#if…#endif). Use conditional methods instead.

**Example:**

public class MyClass

{

[Conditional("MySpecialCondition")]

public void MyMethod()

{

}

}

**Rationale:**

No required.

## Project Structure

### 1

# APPENDIX

## A1 - Capitalization Styles Conventions

We will use the three following conventions for capitalizing identifiers.

**Pascal case**  
The first letter in the identifier and the first letter of each subsequent concatenated word  
are capitalized. You can use Pascal case for identifiers of three or more characters. For  
Example:

**B**ack**C**olor

**Camel case**  
The first letter of an identifier is lowercase and the first letter of each subsequent  
concatenated word is capitalized. For example:  
**b**ack**C**olor

**Uppercase**  
All letters in the identifier are capitalized. Use this convention only for identifiers that  
consist of two or fewer letters.

Example:  
System.**IO;**  
System.Web.**UI;**

You might also have to capitalize identifiers to maintain compatibility with existing,  
unmanaged symbol schemes, where all uppercase characters are often used for  
enumerations and constant values. In general, these symbols should not be visible outside  
of the assembly that uses them.