XME Coding and  
Documentation Guidelines

Contents

[1. Introduction 1](#_Toc379211239)

[2. File History 1](#_Toc379211240)

[3. File and component naming conventions 2](#_Toc379211241)

[4. Error handling 3](#_Toc379211242)

[5. Preprocessor directives 3](#_Toc379211243)

[6. Variables and data structures 3](#_Toc379211244)

[7. Function signatures 4](#_Toc379211245)

[8. Curly brackets and statements 5](#_Toc379211246)

[9. Comments 5](#_Toc379211247)

[10. Documentation 5](#_Toc379211248)

[11. Includes and header files 9](#_Toc379211249)

[12. File structure (license, copyright and file description block) 10](#_Toc379211250)

[13. Documenting group 11](#_Toc379211251)

[13.1 Header files (.h) 11](#_Toc379211252)

[13.2 Implementation files (.c) 11](#_Toc379211253)

[13.3 Subgroups and different architectures (.h) 12](#_Toc379211254)

[14. Usage of bool type 12](#_Toc379211255)

[15. Structure of HAL modules and waypoints 13](#_Toc379211256)

# Introduction

This document lists a few coding guidelines that are used in XME. When source code contains dots (.), this indicates whitespace.

This file is part of CHROMOSOME. Use, modification and distribution are subject to the terms specified in the accompanying license file LICENSE.txt located at the root directory of this software distribution. A copy is available at http://chromosome.fortiss.org/.

# File History

|  |  |  |
| --- | --- | --- |
| **2012-10-08** | MG | Initial version |
| **2012-10-19** | MG | File history and file naming conventions section added |
| **2012-11-05** | MG | Added documentation groups (\ingroup), newline at EOF and section numbering |
| **2012-11-13** | MG | Renamed to “Additional XME Coding *and Documentation* Guidelines” |
| **2012-11-20** | MG | Removed obligation to add author name to file headers |
| **2012-12-07** | MG | Added “Error handling” section |
| **2012-12-20** | MG | Extended file naming conventions by component naming conventions. |
| **2012-12-20** | MG | Fixed specification of section headers. |
| **2013-01-30** | MG | Corrected syntax error in code example. |
| **2013-02-21** | FR | Added sections for preprocessor and variable and datatype definition, included required documentation. |
| **2013-02-21** | MG | Minor corrections to Doxygen documentation. |
| **2013-02-22** | FR | Updated section 3 with updated text of abbreviations. Added doxygen grouping tags in “File Structure” section. Completed “Documentation” section. |
| **2013-03-04** | FR | Updated with further Doxygen documentation. |
| **2013-03-04** | MG | Minor corrections to Doxygen documentation (e.g., use /\*! instead of /\*\*). |
| **2013-03-05** | FR | Corrected subgrouping documentation. Added typedef in variables and data structures section. |
| **2013-05-21** | MG | Removed references to fortiss-internal projects. |
| **2013-05-23** | DG | Added rule for typedef structs which are used recursively (like linked list \*next). |
| **2013-07-22** | DG | Adding guidelines for names which contains abbreviations like IP. |
| **2013-07-23** | MG | Revised rules w.r.t abbreviations. |
| **2013-09-03** | MG/FR | Large revision of coding guidelines w.r.t. Lint warnings and consistency. |
| **2013-09-17** | MG | Updated whitespace policy after Doxygen commands. |
| **2013-09-17** | BW | Fixed wrong XME\_EXTERN\_C\_\* macros ( \_C\_ was missing). |
| **2014-01-14** | BW | Added section about usage of bool type. |
| **2014-01-15** | MG | Minor clarifications to “usage of bool type” section. |
| **2014-01-15** | GK | Changed section “static variables” into “variables”. |
| **2014-01-22** | MG | Restricted usage of “bool” type in function signatures. |
| **2014-02-03** | MG | Added section about structure of HAL modules and waypoints |
| **2014-02-18** | FR | Updated enumeration definitions. |

# File and component naming conventions

|  |  |
| --- | --- |
| **YES** | **NO** |
| myComponentName.h  myComponentName.c  another.h  another.c | MyComponentName.h // No first upper-case letter!  Mycomponentname.c // Use camelCase!  Another.h // No first upper-case letter!  another.C // No upper-case extension! |
| // Directory structure: xme/core/plugAndPlay/src/logicalRouteManager.c  // Functions:  xme\_core\_pnp\_lrm\_doIt();  // Abbreviations such as “IP”, “LRM”, “LASER” are considered as single words, i.e., either all lowercase (if at the beginning of a namespace) or all caps  xme\_com\_interface\_ipv4ToGeneric();  xme\_com\_interface\_genericToIPv4();  xme\_core\_pnp\_lrm\_lrmDoStuff();  xme\_core\_pnp\_lrm\_triggerLRM();  Rules:   * Directory and file names and abbreviations used for component prefixes in functions should be at least 3 characters long. * Abbreviations should be used inside the source code (e.g. functions and global variable names) if the item is longer than 6 characters. * Abbreviations like IP when used in the names should maintain the similar case that is all lower or upper. | // Don’t abbreviate directory names (except for “src”) or source/header file names!  xme/core/pnp/src/lrm.c  // Use abbreviations for function names!  xme\_core\_plugAndPlay\_logicalRouteManager\_doIt();  // Abbreviations such as “IP”, “LRM”, “LASER” are considered as single words, i.e., either all lowercase (if at the beginning of a namespace) or all caps!  xme\_com\_interface\_iPv4ToGeneric();  xme\_com\_interface\_ipv4ToGeneric();  xme\_com\_interface\_genericToipv4(); |

# Error handling

|  |  |
| --- | --- |
| **YES** | **NO** |
| xme\_status\_t  xme\_adv\_compName\_doThis(void)  {  ....xme\_status\_t rval;  ....rval = xme\_adv\_compName\_otherFunc();  ....XME\_CHECK(XME\_CORE\_STATUS\_SUCCESS == rval,  ........XME\_STATUS\_INTERNAL\_ERROR);  ....return rval;  } | xme\_status\_t  xme\_adv\_compName\_doThis(void)  {  ....xme\_adv\_compName\_otherFunc(); // Always check  // return value!  ....return XME\_CORE\_STATUS\_SUCCESS;  } |

# Preprocessor directives

|  |  |
| --- | --- |
| **YES** | **NO** |
| #define XME\_CHECK\_RVAL\_VOID | #define xme\_check\_rval\_void // UPPERCASE |
| #define XME\_CORE\_RR\_DEFAULT\_REQUEST\_TIMEOUT 5000 |  |
| #define XME\_UNUSED\_PARAMETER(param) (void)(param) |  |
| #define XME\_CHECK(condition, rval) \  do { \  if (!(condition)) \  { \  return rval; \  } \  } while (0) | #define XME\_CHECK(condition, rval) // Missing  //Breakline(\)  do { \  if (!(condition)) { \ // Opening bracket  // in new line  return rval; \  } \  } while (0)  // For functions, check corresponding structures  // in following sections |

# Variables and data structures

|  |  |
| --- | --- |
| **YES** | **NO** |
| typedef struct  {  ....uint16\_t nodeId;  ....uvoid\* dataHandler;  } xme\_adv\_myComp\_configStruct\_t; | // Wrap curly brackets to next line!  typedef struct {  ....uint16\_t node2; // Use meaningful vars!  }  xme\_adv\_hmon\_configStruct\_t;  // Type name should not be placed in new line! |
| typedef enum xme\_hal\_myComp\_descriptiveId\_e  {  XME\_ADV\_TEST\_HEARTBEAT = 0,  XME\_ADV\_TEST\_CPU,  XME\_ADV\_TEST\_NOTEST  }; | // Wrap curly brackets to next line!  typedef enum {  // Should be named as \_e.  initial\_heartbeat = 0, // Always use CAPITALIZED identifiers!  TEST\_CPU, // Use chromosome prefixes!  XME\_ADV\_TEST\_NOTEST, // Avoid the last comma in the last member of enum!  } xme\_adv\_test\_type\_t;  // Should be an anonymous enumeration. |
| typedef uint64\_t xme\_hal\_myComp\_descriptiveId\_t; | // typedefs should be defined in the same line  typedef uint64\_t  xme\_hal\_myComp\_descriptiveId\_t; |
| typedef struct linkedList\_genericElement\_s  {  struct linkedList\_genericElement\_s\* next;  void\* item;  } linkedList\_genericElement\_t; | // Use \_s for structure names and not \_t  // \_t is reserved for typedef.  typedef struct linkedList\_genericElement\_t  {  struct linkedList\_genericElement\_t\* next;  void\* item;  }  linkedList\_genericElement\_t; // Type name should be placed in a new line! |

# Function signatures

|  |  |
| --- | --- |
| **YES** | **NO** |
| static void  xme\_adv\_compName\_doThis(void)  {  ....[...]  } | // Return value/modifiers on separate line!  static void xme\_adv\_compName\_doThis  (  ....void // Do not indent void! Do not omit it!  ) { // Break curly brackets!  ....[...]  } |
| void  xme\_adv\_compName\_doThat  (  ....uint16\_t count  )  {  } | // Wrap function/macro parameters!  // Wrap curly brackets to next line!  // No cryptic parameter names!  // Use specific data type size if possible!  void xme\_adv\_compName\_doThat(int i) {  🡪 [...] // 4 spaces instead of 1 tab!  } |
| bool  xme\_adv\_compName\_doThatAsWell  (  ....uint16\_t numBytes  )  {  ....xme\_hal\_table\_rowHandle\_t r1, r2;  ....int\* a;  ....int\* b;  ....[...]  ....return (\*a == \*b);  } | bool  xme\_adv\_compName\_doThatAsWell  (  ....uint16\_t uBytes // No Hungarian notation!  )  {  ....xme\_hal\_table\_rowHandle\_t r1, r2;  ....int\* a, \*b; // One pointer var.d. per line!  ....[...]  ....return \*a == \*b; // Brackets recommended  } |
| static void  xme\_adv\_compName\_internalFunc  (  ....[...]  ); | void // Use static for internal functions!  xme\_adv\_compName\_internalFunc  (  ....[...]  ); |

# Curly brackets and statements

|  |  |
| --- | --- |
| **YES** | **NO** |
| if (a && (b == c))  {  ....[...]  }  else if (d || (0x02 == e ^ f))  {  ....[...]  }  else  {  ....[...]  } | if (a && b == c) // Add brackets!  {  ....[...]  } else if (d || (e ^ f == 0x02)) { // Wrap {}!  ....[...] // Put constant (0x02) to left side!  } else { // Wrap {}!  ....[...]  } |
| if (someLongVariableName == someValue ||  ....someLongVariableName == someOtherValue)  {  ....[...]  } | if (someLongVariableName == someValue ||  someLongVariableName == someOtherValue) //Indent!  {  ....[...]  } |
| switch (var)  {  ....case 1:  ........doSomething();  ........// fallthrough  ....case 2:  ........doSomethingElse();  ........break;  ....default:  ........throwError();  } | switch (var) { // Wrap {}!  case 1: // Indent case labels!  ....doSomething();  ....// Comment fall-through semantics!  case 2:  ....doSomethingElse();  ....break;  // Always add default case!  } |
| for (;;) // No spaces around ";"  {  ....[...]  } | while (1) // Avoid while(1) (causes warnings)!  {  ....[...]  } |

# Comments

| **YES** | **NO** |
| --- | --- |
| #if 0  someLargeBlockOfCommentedOutCode();  #endif // #if 0 | // Comment large blocks of code with “#if 0” instead of “/\* ... \*/”  /\*  someLargeBlockOfCommentedOutCode();  \*/ |
| // Analyze the status in order to determine what to do.  // Since this block contains full sentences, each sentence  // should be terminated with a full stop.  if (XME\_STATUS\_SUCCESS == status)  {  // Process the login response, because the status is OK  doIt();  }  else  {  // Raise an error  XME\_LOG(XME\_LOG\_ERROR, "Error, status is %d!\n", status);  } |  |

# Documentation

| **YES** | **NO** |
| --- | --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Prototypes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \* \brief Description.  \*  \* \note Notes about the function.  \*  \* \param[in] param Input parameter.  \* \param[out] result Result of the function.  \*  \* \retval XME\_STATUS\_SUCCESS on success.  \* \retval XME\_STATUS\_INTERNAL\_ERROR on error.  \*/  xme\_status\_t  xme\_adv\_compName\_doThis  (  ....int param,  ....double\* result  ); |  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Defines \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \* \def XME\_CHECK  \*  \* \brief Description.  \*  \* \note Notes about the function.  \*  \* \param[in] condition Condition to check.  \* \param[in] rval Return value.  \*/  #define XME\_CHECK(condition, rval) \  do { \  if (!(condition)) \  { \  return rval; \  } \  } while (0) |  |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Type definitions \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \* \struct xme\_adv\_myComp\_configStruct\_t  \*  \* \brief Configuration struct for my component.  \*/  typedef struct  {  xme\_core\_node\_nodeID\_t nodeID; ///< Node ID.  xme\_core\_component\_t componentID; /\*!< Identifier  of component \*/  } xme\_adv\_myComp\_configStruct\_t;  /\*\*  \* \enum xme\_adv\_hmon\_status\_t  \*  \* \brief Enumeration type definition for health  \* monitoring status.  \*/  typedef enum  {  XME\_ADV\_HMON\_COMPONENT\_INVALID = 0, ///< Invalid status.  XME\_ADV\_HMON\_COMPONENT\_OK, ///< Component is OK.  XME\_ADV\_HMON\_STATUS\_UNKNOWN, ///< Unknown status.  } xme\_adv\_hmon\_status\_t;  /\*\*  \* \typedef xme\_hal\_myComp\_descriptiveID\_t  \*  \* \brief A descriptive brief description.  \*/  typedef uint64\_t xme\_hal\_myComp\_descriptiveID\_t; | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Type definitions \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  typedef struct { // missing header  // brackets in new line  xme\_core\_node\_nodeID\_t nodeID;  // the field description should start with ///<  xme\_core\_component\_t componentID; // Id  // Invalid comments in documenting field ^^  } xme\_adv\_hmon\_configStruct\_t;  // brackets in new line.  /\*!  \* \enum other name // same name as enum name  // without white spaces!  \*  \* \brief Enumeration type def for health  \* monitoring status // Add full stop  \*/  typedef enum {  XME\_ADV\_HMON\_COMPONENT\_OK = 0,  XME\_ADV\_HMON\_COMPONENT\_EXCEPTION,  XME\_ADV\_HMON\_STATUS\_UNKNOWN,  // enum items description needed (/\*!<... \*/)  } xme\_adv\_hmon\_status\_t;  /\*! \\ use /\*\* instead  \* \typedef \\ include typedef name  \*  \* \brief A descriptive brief description  \\ phases should be end with full stop (.)  \*/  typedef uint64\_t xme\_hal\_myComp\_ descriptiveID\_t; |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Prototypes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \* \brief.Does something.  \*  \* \details.A long description of the function. This  \* .........long description should be aligned with  \* .........the rest of the comments.  \*  \* \code{.c} // code highlighting  \* int16\_t var;  \* if (var == NULL)  \* {  \* var = 15;  \* }  \* \endcode // end of code highlighting  \*  \* \note.[...]  \*  \* \param[in].numBytes How many bytes to process.  \* \param[in,out].buf Buffer where to put read bytes.  \*  \* \return.Returns value that after performing some  \* operation.  \* \\ WHEN THE RETURN VALUES CAN BE DETERMINED  \* \\ BEFOREHAND, USE \retval instead:  \* \retval.XME\_CORE\_STATUS\_SUCCESS if [...]  \* \retval.XME\_CORE\_STATUS\_[...] if [...]  \*/  xme\_status\_t  doSomething  (  ....uint16\_t numBytes, ....char\* buf  );  [...]  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Implementation \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  xme\_status\_t  doSomething  (  ....uint16\_t numBytes  )  {  ....[...]  } | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Prototypes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  xme\_status\_t  doSomething  (  ....uint16\_t numBytes  );  [...]  // Missing section header!  /\*! // Document the prototype if available!  \* \brief..[...]  \* \param..numBytes [...]  \* \return.[...] // Spacing!  \*/  xme\_status\_t  doSomething  (  ....uint16\_t numBytes, ....char\* buf  )  {  ....[...]  } |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* <section-name> \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  Format:   * Slash + 78 stars + slash * Slash + 3 stars + 3 spaces + title + 3 stars + slash * Slash + 78 stars + slash   Do not omit section headers! Do not add sections headers with empty content!  Available names so far (preferably in this order):   * Includes * Defines * Type definitions * Variables * Prototypes * Implementation (only in source files!) * Platform-specific includes (headers only!) |  |

# Includes and header files

|  |  |
| --- | --- |
| **YES** | **NO** |
| Module xme/hal/myModule.c:  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Includes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include “xme/hal/myModule.h” // 1. Direct header  #include “xme/core/abc.h” // 2. Core headers in  #include “xme/core/something.h” // alphabet. order  #include “xme/adv/c1.h” // 3. Others grouped  #include “xme/adv/c2.h” // in alphabetical order  #include “xme/hal/hal1.h”  #include “xme/hal/hal2.h”  #include <stdint.h> // 4. System includes  #include <stdio.h> // in alphabetical order  [...]  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Platform-specific includes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "xme/hal/myModule\_arch.h" // Architecture  <EOF> // specific include must be last | Module xme/hal/myModule.c:  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Includes \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include <stdint.h> // No arbitrary order!  #include “xme/core/defines.h”  #include “xme/hal/myModule.h”  #include “xme/core/something.h”  #include “xme/adv/c1.h”  #include “xme/adv/c2.h”  #include <stbool.h> // No unneeded include!  #include “xme/hal/hal1.h”  #include “xme/hal/hal2.h”  #include <stdio.h>  #include "xme/hal/myModule\_arch.h" // Not too  // early!  [...]  <EOF> |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Includes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include “xme/defines.h” // If needed  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Prototypes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  XME\_EXTERN\_C\_BEGIN  <function prototypes>  XME\_EXTERN\_C\_END | // Do not forget XME\_EXTERN\_C\_\* in  // all header files for C++ compatibility!  // Externally visible variables also need  // to be enclosed in these macros! |
| Header xme/adv/component.h:  #ifndef XME\_ADV\_COMPONENT\_H  #define XME\_ADV\_COMPONENT\_H  [...]  #endif // #ifndef XME\_ADV\_COMPONENT\_H | Header xme/adv/component.h:  #ifndef COMPONENT\_H // Use full path in guard!  #define COMPONENT\_H  [...]  #endif // Repeat #ifdef arguments! |

# File structure (license, copyright and file description block)

|  |
| --- |
| **YES** |
| /\*  \* Copyright (c) 2011-2013, fortiss GmbH. // Always 2011-<current year>,  \* Licensed under the Apache License, Version 2.0. // update only when file is edited  \*  \* Use, modification and distribution are subject to the terms specified  \* in the accompanying license file LICENSE.txt located at the root directory  \* of this software distribution. A copy is available at  \* http://chromosome.fortiss.org/.  \*  \* This file is part of CHROMOSOME.  \*  \* $Id$ // For SVN keyword expansion  \*/  /\*\*  \* \file  .\*.\brief <My component> abstraction. // In this example, the file is called xme/hal/myComp.h  \*  \* An extended description // In case it is needed. Without white or tab spaces  #ifndef XME\_ADV\_COMPONENT\_H  #define XME\_ADV\_COMPONENT\_H  [...]  // Last line of code<EOL> // “A source file that is not empty shall end in a new-line character [...].” <EOF> |

# Documenting group

## Header files (.h)

|  |
| --- |
| **YES** |
| /\*\*  \* \file  \* [...]  \*/  #ifndef XME\_ADV\_MYCOMPONENT\_H  #define XME\_ADV\_MYCOMPONENT\_H  // Defining documentation groups is optional, but recommended.  // Group definition should be placed just after the component #define block  /\*\*  \* \defgroup hal\_myComp My HAL component // group\_name short description  \* @{ // opens the group definition .\*.\brief..HAL component for doing cool stuff. // brief description of the group  \*  \* Further optional long description  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Includes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  [...]  // at the end of the file, the group tag should be closed, just before the end of the file  /\*\*  @}  \*/ // The functions and file structure between @{ and @} generate automatic documentation  // based in available tags  #endif // #ifndef XME\_ADV\_COMPONENT\_H  // Last line of code<EOL> // “A source file that is not empty shall end in a new-line character [...].” <EOF> |

## Implementation files (.c)

|  |
| --- |
| **YES** |
| /\*\*  \* \file  \* [...]  \*/  // The group should be defined in the corresponding header file (see previous section)  // Group definition should be placed just after the \file block  /\*\*  \* \addtogroup hal\_myComp   \* @{ // opens the group inclusion  \*/  [...]  // at the end of the file, the group tag should be closed, just before the end of the file  /\*\*  @}  \*/ // The functions and file structure between @{ and @} generate automatic documentation  // based in available tags  // Last line of code<EOL> // “A source file that is not empty shall end in a new-line character [...].” <EOF> |

## Subgroups and different architectures (.h)

|  |
| --- |
| **YES** |
| /\*\*  \* \file  \* [...]  \*/  #ifndef XME\_ADV\_MYSUBCOMPONENT\_H  #define XME\_ADV\_MYSUBCOMPONENT\_H  // The subgroup should be defined inside an existing group  // Group definition should be placed just after the \file block  /\*  \* \ingroup hal\_myComp  \* @{  \*  \* \defgroup hal\_myComp\_x86 My HAL Component (x86 architecture)  \* @{  \*  \* \brief Subcomponent/architecture description.  \*/  [...]  // at the end of the file, both \ingroup and \defgroup tags shall be closed  /\*\*  \* @} // closing of hal\_myComp\_x86 \defgroup tag  \* @} // closing of hal\_myComp \ingroup tag  \*/  #endif // #ifndef XME\_ADV\_MYSUBCOMPONENT\_H  // Last line of code<EOL> // “A source file that is not empty shall end in a new-line character [...].” <EOF> |

# Usage of bool type

The size of the **bool** type is not defined in the C standard and may hence vary between compilers. This can lead to errors when C and C++ code is mixed or when parts of the same application are compiled using different compilers, such as shared libraries that are used as plug-ins.

Whenever data structures that are defined in CHROMOSOME might be used (i.e., directly accessed) from within code compiled with a different compiler or in C++ mode (e.g. unit tests, user components), use **char** instead of **bool**. The same applies to function signatures, where **bool** should be avoided as a parameter or return value. Use the **char** type instead, which is guaranteed to be 1 byte large and is most compatible across platforms (as opposed to, for example, **int8\_t**). Some care is required when doing so, however:

|  |  |
| --- | --- |
| **YES** | **NO** |
| char isDone; // Used as bool  isDone = 0; // false  isDone = 1; // true  // Testing (preferred):  if (isDone) { }  if (!isDone) { }  // Testing (alternative):  if (isDone != 0) { }  if (0 == isDone) { }  char doIt(char isVerbose); // Add "is" to indicate  // boolean nature of  // parameter | char isDone; // Used as bool  isDone = false; // May cause performance warning  isDone = true; // May cause performance warning  if (isDone == true) { } // Any non-zero value is  // “true”, but may not be  // equal to true!  if (isDone == false) { } // Both trigger a  //warning in C++  bool doIt(bool verbose); // Avoid bool in return  // value or parameter! |

# Structure of HAL modules and waypoints

**HAL modules** and **waypoints** typically offer similar programming interfaces, respectively. The following suggestions should be considered when defining them.

|  |
| --- |
| **YES** |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Prototypes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  XME\_EXTERN\_C\_BEGIN  /\*\*  \* \brief Initializes this abstraction.  \*  \* \retval XME\_STATUS\_SUCCESS on success.  \* \retval XME\_STATUS\_OUT\_OF\_RESOURCES if not enough resources were available.  \* \retval ...  \*/  xme\_status\_t // \_init() typically returns values of type xme\_status\_t  xme\_hal\_myAbstraction\_init(void); // and should return XME\_STATUS\_SUCCESS on success  // \_init() should internally increment a reference counter that is initially zero  // and only do the initialization if the counter has been zero before  // in order to allow components to call it agnostic of other components.  // Some HAL components may not offer an \_init() function;  // those components do not need to be initialized or finalized.  /\*\*  \* \brief Frees all resources occupied by this abstraction.  \*/  void // \_fini() typically returns void and handles errors gracefully  xme\_hal\_myAbstraction\_fini(void); // (an error code would not be useful to the caller)  // \_fini() should internally decrement the reference counter  // and only do the finalization if the counter is zero afterwards  // in order to allow components to call it agnostic of other components  // Some HAL components may not offer a \_fini() function;  // those components do not need to be initialized or finalized.  /\*\*  \* \brief ...  \*  \* \param[in] ...  \*/  returnValueType\_t  xme\_hal\_myAbstraction\_someFunc  (  ...  );  ...  XME\_EXTERN\_C\_END |

|  |
| --- |
| **YES** |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\* Prototypes \*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  XME\_EXTERN\_C\_BEGIN  /\*\*  \* \brief Initialize this waypoint class.  \*  \* \retval XME\_STATUS\_SUCCESS on success.  \* \retval XME\_STATUS\_OUT\_OF\_RESOURCES if not enough resources were available.  \*/  xme\_status\_t // \_init() typically returns values of type xme\_status\_t  xme\_wp\_myWaypoint\_init(void); // and should return XME\_STATUS\_SUCCESS on success  /\*\*  \* \brief Executes the given instance of this waypoint class.  \*  \* \param[in] instanceID Identifier of the configuration for which to execute  \* the waypoint, as returned by the respective call to  \* xme\_wp\_myWaypoint\_addConfig().  \*  \* \retval XME\_STATUS\_SUCCESS on success.  \* \retval XME\_STATUS\_INVALID\_HANDLE if the given instance identifier was invalid.  \* \retval ...  \*/  xme\_status\_t  xme\_wp\_myWaypoint\_run  (  xme\_wp\_waypoint\_instanceId\_t instanceID  );  /\*\*  \* \brief Add a new configuration to this waypoint class.  \*  \* \param[in,out] instanceID Address of a variable where the identifier for the  \* newly added configuration is written to. Only valid if the  \* function returns XME\_STATUS\_SUCCESS.  \* \param[in] ...  \*  \* \retval XME\_STATUS\_SUCCESS if the configuration has been successfully added.  \* \retval ...  \* \retval XME\_STATUS\_OUT\_OF\_RESOURCES if the configuration could not be added  \* due to resource constraints (e.g., not enough memory to store entry).  \*/  xme\_status\_t  xme\_wp\_myWaypoint\_addConfig  (  xme\_wp\_waypoint\_instanceId\_t\* instanceID,  ...  );  /\*\*  \* \brief Removes a configuration of this waypoint.  \*  \* \param[in] instanceID Instance identifier of the configuration to be removed.  \*  \* \retval XME\_STATUS\_SUCCESS if configuration was successfully removed.  \* \retval XME\_STATUS\_INVALID\_HANDLE if the given instance identifier was invalid.  \*/  xme\_status\_t  xme\_wp\_myWaypoint\_removeConfig  (  xme\_wp\_waypoint\_instanceId\_t instanceID  );  /\*\*  \* \brief Frees all resources occupied by this waypoint class.  \*/  Void // \_fini() typically returns void and handles errors gracefully  xme\_wp\_myWaypoint\_fini(void); // (an error code would not be useful to the caller)  XME\_EXTERN\_C\_END |