XME Coding and Documentation Guidelines

Contents

1. Introduction

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

2. 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.

3. 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();

4. 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; }

5. 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

6. 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!
```

7. 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` `(` `....[...]` `);`

8. 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)! { ....[...] }

9. 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); }	

10. Documentation

YES	NO

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)	

YES	NO
/***/ /*** 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;

YES	NO
/***/ /*** 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!)	

11. 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!

12. 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>
```

13. Documenting group

13.1 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>
```

13.2 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>
```

13.3 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>
```

14. 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

```
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
```

NO

```
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!
```

15. 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
```