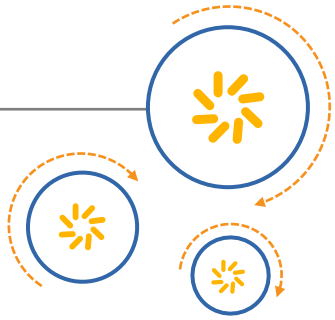
Qualcomm Technologies, Inc.



H2XML Build Tool

User Guide

80-xxxxx-x x

April 18, 2017

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
A	July 2016	Initial release
B	December 2016	Update for H2XML version 1.0.1.0
C	April 2017	Update for H2XML version 1.0.2.0

Note: There is no Rev. I, O, Q, S, X, or Z per Mil. standards.

Contents

1 Introduction 7

1.1 Purpose 7
1.2 Conventions 7
1.3 Technical assistance 7

2 Overview 8

3 Documenting the Code 9

3.1 Placing Annotations before the Identifier 9
3.2 Placing Annotations after the Identifier 10
3.3 Referencing an Identifier 11
3.4 Module Definition 12
3.5 Annotations 12
3.6 Coexistence of H2XML and Doxygen Annotations 13

4 Nested Modules, Structures and Inheritance 14

5 H2XML Usage 15

6 H2XML Configuration 16

6.1 Annotations Definition 17
6.1.1 Attributes 17
6.1.2 Tags 19
6.1.3 Aliases 20
6.1.4 Configuration Arguments 21

7 RTF Documentation 21

8 Master Configuration 22

8.2 Config_Master Global Annotations 23
8.3 Config_Master XML Annotations 23
8.4 Config_Master Module Annotations 24
8.4.1 Using global Parameters within a Module 26
8.5 Config_Master Parameter Annotations 27
8.6 Config_Master Element Annotations 28
8.6.1 Coding convention for dynamic annotation values 30

9 H2XML Internals 31

A References 32

A.1 Related documents 32
A.2 Acronyms and terms 32

Figures

Figure 2-1 H2XML overview 8
Figure 9-1 H2XML Internals 31

Tables

Table 5-1 H2XML command line options 16
Table 6-1 Configuration Atrributes 18
Table 6-2 Configuration Attribute Types 19
Table 6-3 Configuration Tags 20
Table 6-4 Configuration Aliases 20
Table 6-5 Configuration Arguments 21
Table 7-1 RTF template key words 21
Table 8-1 Config_Master Global Annotations 23
Table 8-2 Config_Master XML Annotations 24
Table 8-3 Config_Master Module Annotations 26
Table 8-4 MASTER Parameter Annotations 28
Table 8-5 Config_Master Parameter Annotations 30

1 Introduction

1.1 Purpose

This document describes the functionality and usage of the H2XML Build Tool. The document contains two parts:

- Part one is a general manual for the H2XML build tool
- Part two gives some practical examples by using the configuration for the WDSP

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, `<number>`.

Commands to be entered appear in a different font, for example, **`copy a:*.* b:`**.

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

Shading indicates content that has been added or changed in this revision of the document.

1.3 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://support.cdmatech.com/.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com.

2 Overview

H2XML is a generic tool for generating XML files from annotated C header files. Grammar and syntax of the annotations are similar to Doxygen. Thus the annotations can be included in Doxygen code documentation as well as in the generated XML output file.

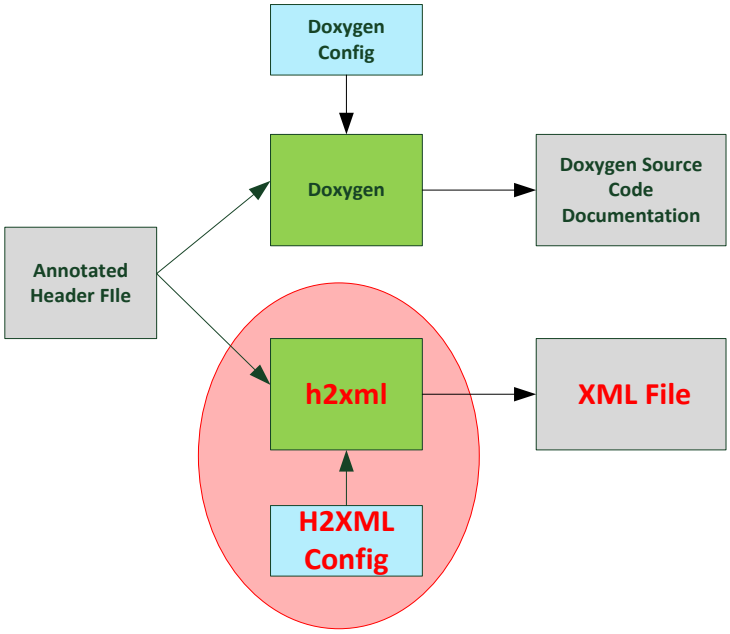


Figure 2-1 H2XML overview

Annotation can be added to:

- C-language Structure, Union or Enum identifiers
- C-language element identifiers, e.g. struct members and bit-fields
- Modules (which have no equivalent in the C-language), which allow to group structures and elements.

A set of annotations and their appearance in the generated XML output can be defined in a configuration file.

3 Documenting the Code

Documentation of annotation follows the Doxygen documentation style. All annotations must be within a Doxygen-style comment block. Annotations within the comment block are case insensitive.

Example:

comment_modes1.h

comment_modes1.xml

comment_modes2.h

comment_modes2.xml

comment_modes3.h

comment_modes3.xml

3.1 Placing Annotations before the Identifier

All annotations in this comment block will be added identifier following this block. In the below examples the annotations will be added to the variable `int a`.

1. C-style comment starting with two *'s, like:

```
/**
    …some annotation
*/
int a;
```

2. Or starting with a '!' after opening of a c-style comment, e.g.:

```
/*!
    …some annotation
*/
int a;
```

3. Or starting with a '#' after opening of a c-style comment. Note: This comment style will not be recognized by Doxygen. Comment blocks that should only be evaluated by H2XML but not by Doxygen can use this style, e.g.:

```
/*#
    …some annotation
*/
int a;
```

4. A single comment line starting with ‘///’, e.g.

```
/// …some annotation
int a;
```

5. Or a single comment line stating with ‘//!’, e.g.

```
//! …some annotation
int a;
```

6. Or a single comment line stating with ‘//#’. Note: This comment style will not be recognized by Doxygen. Comment blocks that should only be evaluated by H2XML but not by Doxygen can use this style, e.g.:

```
//# …some annotation
int a;
```

3.2 Placing Annotations after the Identifier

If you want to place the annotations after an identifier put an additional ‘<’ marker in the comment block. In the below examples the annotations will be added to the variable `int a`.

1. C-style comment followed by ‘*<’, like:

```
int a;
/**<
    …some annotation
*/
```

2. Or staring with a ‘!<’ after opening of a c-style comment, e.g.:

```
int a;
/*!<
    …some annotation
*/
```

3. Or staring with a ‘#<’ after opening of a c-style comment. Note: This comment style will not be recognized by Doxygen. Comment blocks that should only be evaluated by H2XML but not by Doxygen can use this style, e.g.:

```
int a;
/*#<
    …some annotation
*/
```

4. A single comment line starting with '///<', e.g.

```
int a;  ///< …some annotation
```

5. Or a single comment line stating with '//!<', e.g.

```
int a;  //!< …some annotation
```

6. Or a single comment line stating with '//#<' .Note: This comment style will not be recognized by Doxygen. Comment blocks that should only be evaluated by H2XML but not by Doxygen can use this style, e.g.:

```
int a;  //#< …some annotation
```

3.3 Referencing an Identifier

Annotations within a comment block as defined in Chapter 3.1 or Chapter 3.2 can be associated with any identifier that has been previously defined. The identifier is selected by structure name(s) and identifier name separated by '::'.

Example:

```
/*********************************************************************//**
* @file
* Example for h2xml_select
*********************************************************************/

typedef struct {
    int exa;            /**< @h2xmle_default    {0x1}*/
    int exb;            /**< @h2xmle_default    {0x2}*/
    int exc;            /**< @h2xmle_default    {0x3}*/
} sStruct1_t;

typedef struct {
    sStruct1_t s1;
    char exy;           /**< @h2xmle_default    {0x4}*/
    short exz;          /**< @h2xmle_default    {0x5}*/
}sStruct2_t;


/*-------------------------------------------------------------------*//**
    @h2xmlm_module                  {MODULE_1,0x11111111  }
    @{
--------------------------------------------------------------------*/

/** @h2xmlp_parameter {Parameter1,0x11111111} */
struct param_1{
    short   a1;     /**< @h2xmle_default        {0x88} */
    int     b1;     /**< @h2xmle_default        {0x99} */
    sStruct2_t c1;
};

/**
    @h2xml_select   {param_1::c1::s1::exa}
```

```
    @h2xmle_default {0x11}
    @h2xml_select   {sStruct2_t::exy}
    @h2xmle_default {0x44}
*/

/** @} */                              /* End of Module */
```

In the above example, param1.c1.s1.exa will have the default value ‘0x11’ (which replaces the original default value ‘1’, param1.c1.s1.exy will have the default value ‘0x44’.

Example:

h2xml_select.h

h2xml_select.xml

3.4 Module Definition

Since modules have no c-code equivalence, they must be defined explicitly. Similar to Doxygen modules, a module starts with ‘@{‘ inside a h2xml comment and ends with a ‘@}’ inside an h2xml comment:

```
/**
    @{          <!-- Start of module -->
*/

// some parameter definitions

/**@}           <!-- End of Module --> */
```

Modules can be nested.

Example:

h2xmlm_module.h

h2xmlm_module.xml

3.5 Annotations

Annotations follow the style of Doxygen aliases. They are defined by

```
@label{ArgumentList}
```

Where label can be any c-style label defined by the regular expression:

```
[a-zA-Z_][a-zA-Z0-9_.]*
```

Arguments can be separated by ‘;’, ‘,’, ‘..’ or ‘=’.

Comments that should not be considered by the parser can be put into XML-style comments:

```
<!--Some Comment-->
```

Please note that annotations, their arguments and appearance in the output XML are defined in the configuration file.

Examples:

```
#define MODULE_ID 0x1234
/**<
    @htxmlm_ID          { MODULE_ID }           <!-- MODULE_ID will be
                                                replaced by 0x1234 -->
    @htxmlm_InputPins   {Pin1=1;Pin2=2;Pin3=3}  <!-- 6 Arguments -->
    @h2xmle_Range       {1..27}                 <!-- 2 Arguments -->
*/
```

3.6 Coexistence of H2XML and Doxygen Annotations

Doxygen will ignore unknown aliases (i.e. annotations starting with '@'), but issue a warning. By adding an empty alias definition to the Doxygen configuration file the annotation will be ignored, e.g.

```
Alias+=@htxmlm_ID=""
```

In general, it will be advantageous to use the same annotations for H2XML and for Doxygen. Therefore the Doxygen configuration file should be updated accordingly. Please refer to the Doxygen documentation at http://www.**doxygen**.org.

Please note that Doxygen uses ',' as an argument separator. Thus Arguments like '1..27' will be considered by Doxygen as single string.

If annotations shall not be used by Doxygen, comment block styles '/*#' and '//#' can be used, which will not be ignored by Doxygen (see chapter 3).

4 Nested Modules, Structures and Inheritance

Annotations can be added to modules, structures and structure elements. In addition, global annotations (defined in the command line or in the configuration file) can be added to the file.

Accordingly there are four types of annotations:

- Global annotations
- Module annotations
- Structure annotations
- Element annotations

Modules and structures can be nested. Annotations will be inherited by the subordinate structure. I.e.

- Global annotations will be inherited by all Modules, structures and elements
- Module annotations will be inherited by subordinate modules, structures, and elements
- Structure annotations will be inherited by all subordinate structures and elements
- Element annotations will not be inherited, since elements are always terminal

Inheritance	Modules	Structures	Elements
Global annotations	Y	Y	Y
Module annotations	Y	Y	Y
Structure annotations		Y	Y
Element annotations			

Examples:

The following example assumes that an element annotation ‘h2xmle_visibility’ is defined in the configuration file.

```
/**
    @h2xmle_visibility          {show}
*/
typedef struct {
    int a;
    int b;
}sGain_t;
```

```
typedef struct {
    int     version;
    /**
        @h2xmle_visibility      {hide}
    */
    sGain_t gain1;
    sGain_t gain2;
}sParameter_t
```

The element annotation `'h2xmle_visibility = show'` will be applied to all elements of the sGain_t structure. Furthermore, all instances of sGain_t in sParameter_t will possess this annotation. The element version of sParamter_t gets the annotation `'h2xmle_visibility = hide.`

5 H2XML Usage

H2XML is a command line based utility. The command line is

```
h2xml.exe –conf <config_file> -o <output_file>  [optional parameters]
input_file
```

Mandatory Parameters	Description
-conf <config_file>	<config_file> must be a valid h2xml configuration file
<input_file> or – i <input_file>	A C-header input file with extension '.h'. Output file name will be <input_file>.xml. Must be last argument if '- i' is omitted.
Optional Parameters	
-f <commandFile>	Text file that contains command line options. Lines starting with '#' are ignored.
-o <output_directory>	The output file(s) will be written to this directory. If this parameter is omitted, output will be to the actual working directory.
-I <dir>	Add the directory <dir> to the list of directories to be searched for header files. Several include directories can be added by applying multiple –I parameters.
-D <name>	Predefine name. Several definitions can be added by applying multiple –D parameters. The definition __H2XML__ is pre-defined.
-rtf <template_file>	Generate rtf documentation, using the template file. Output is written to 'output_directory\<input_file>.rtf'
-ortf < output_directory>	The rtf output file(s) will be written to this directory. If this parameter is omitted, output is to the same directory as the XML files.
-v	Print version information
-h	Print a help file
-verbose	Print verbose information for debugging into 'output_directory\<input_file>_verbose.txt'

Mandatory Parameters	Description
-pointerSize16	This option must be set for 16-bit systems that store pointers in 16 bits. Default for pointers is 32 bits. The correct pointer size is needed to calculate size and offset of elements containing pointers.
-WPacked	Issue a warning if "__attribute__((packed))" is used and byte-offsets for packed and un-packed version are different.

Table 5-1 H2XML command line options

Note: The compiler front-end of the H2XML build tool defines the flag '__H2XML__'. Any code that should be ignored by the H2XML tool can be omitted by

```
#ifndef __H2XML__
    // code to be ignored
#endif
```

Vice versa, code specifically for the h2xml tool can be included by

```
#ifdef __H2XML__
    // code to be included
#endif
```

For an example of a configuration file please refer to test_compile_flag__H2XML__.h

6 H2XML Configuration

The syntax of the configuration file is subject to change, so this documentation is somehow preliminary.

The H2XML configuration file is case insensitive.

The configuration XML file consists of element H2XML_CONFIG and four sub-elements:

- XML_GENERATOR: properties of the generated XML output file
- MODULE_ANNOTATIONS: defines the annotations associated with modules
- STRUCTURE_ANNOTATIONS: defines the annotations associated with structures
- ELEMENT_ANNOTATIONS: defines the annotations associated with elements

Each of the elements can have a "prefix" attribute. This prefix will be added to the key words defined in the section. Example:

```
<H2XML_CONFIG prefix="h2xml">
    <MODULE_ATTRIBUTES prefix="m_">
        <ATTRIBUTE keyword="ID" Arguments="INTEGER"/>
    </MODULE_ANNOTATIONS>
</H2XML_CONFIG>
```

The module annotation “ID” has one integer argument and can be set from within an h2xml comment with the keyword ‘**@h2xmlm_id** {**value**}’.

For an example of a configuration file please refer to config_wcd.xml

6.1 Annotations Definition

Annotations can be defined for modules, structures and elements. The annotation contain three types:

- ATTRIBUTE: Defines an attribute that will be attached to an element in the generated XML. Attribute definitions can only have a single argument.
- TAGS: Defines an XML TAG that will be added to the generated XML. Tag definitions have no arguments
- ALIAS: Aliases define an easy-to-read syntax for the header annotations and can have multiple arguments or argument lists.

6.1.1 Attributes

Attribute will be attached to an element in the generated XML. Attribute definitions can only have a single argument.

Attribute	Argument	Description
keyword	LABEL	An unambiguous label for this attribute. Prefixes for Global, Module, Structure or Element will be added.
name	LABEL	With this name it will show up in the output XML. By default, name is set to the same as “keyword”
arguments	ARGUMENT	One argument. Please refer to the Arguments type table Configuration Arguments.
parent	LABEL	Attach this attribute to a parent element. By default, the attribute is attached to the last active element.
min	INT	Minimum value for integer arguments. An error will be issued, if the argument exceeds this range.
max	INT	Maximum value for integer arguments. An error will be issued, if the argument exceeds this range.
labelList	LABEL [; LABEL [; ..]]	A list of valid Label arguments. If the argument matches none of the list entries, an error will be issued.
show	BOOL	If true, show this attribute in XML output.

Attribute	Argument	Description
showInTag	LABEL [; LABEL [; ..]]	Show this attribute in XML output if it is inside a Tag contained in the LABEL list.
removeQuotes	BOOL	Remove quotes (“ or ‘) from argument.
default	STRING	If no argument is given, the default value is applied.
addToFront	BOOL	Add element to begin of list (instead of appending it to the end)
attributeType	ATTRTYPE	Some attributes have special functions that need be evaluated during XML generation (e.g. like “name”). For a list of valid types refer to table Table 6-2.

Table 6-1 Configuration Atrributes

Example:

```
<ATTRIBUTE keyword="visibility" Arguments="LABEL" labelList="show, hide"
  default="show"/>
```

6.1.1.1 Configuration Attribute Types

Some attributes have a special function that must be known by the h2xml tool. E.g. the element name will be set with the variable name if no name is specified. The following attribute types are available:

Attribute Type	Arguments	Description
GLOBAL_ANNOTATIONS	BOOL	Start of global annotations
GENERATE_OUTPUT	BOOL	Include this element in generated output
SELECT	LABEL	The following annotations refer to the selected identifier
INSERT_PARAM		insert external parameter to Module
CREATE_STRUCT	LABEL	create an identifier
XML_LINE_LENGTH	INT	Line length of the generated XML output
XML_TAB_SIZE	INT	Tab size of the generated XML output
XML_ATTRIBUTE_LENGTH	INT	Maximum length of an XML attribute value. If this value is exceeded, a new-line is inserted in the value string.
XML_NUMBER_FORMAT	‘INT’ | ‘HEX’	Number format in output XML, either integer or hexadecimal. Default is Integer.
EXPAND_STRUCTS	BOOL	Expand structure to its elements for the generated output
EXPAND_TYPEDEFS	BOOL	Display base type instead of the typedef name
EXPAND_ENUMS	BOOL	Display the value of enums and #defines in the generated output
EXPAND_ARRAY	BOOL	Display all array elements in the generated output
EXPAND_ARRAY_OFFSET	INT	Array element names will start with this value
NAME	LABEL	The element’s name
DEFAULT	INT	The element’s default value

Attribute Type	Arguments	Description
DEFAULTFILE	STRING	Specifies a binary file that is used to initialize an array or structure. The file will be searched for in the include directories specified with the –I command line option. The size of the file must match the array or structure size.
DEFAULTLIST	INT[;INT[…]]	Specifies a list of default values to initialize an array or structure. The number of elements must match the array or structure size.
DEFAULT_CONDITIONAL	INT	Same as 'DEFAULT', but value depends on conditions.
DEFAULTFILE_CONDITIONAL	STRING	Same as 'DEFAULTFILE', but value depends on conditions.
DEFAULTLIST_CONDITIONAL	INT[;INT[…]]	Same as 'DEFAULTLIST', but value depends on conditions.
BITMASK	UINT	Bitmask for bit field
BYTESIZE	INT	Element size in bytes
BITESIZE	INT	Element size in Bits
BYTEOFFSET	INT	Byte address of element within a structure
BITOFFSET	INT	Bit address of element within a structure
ELEMENTTYPE	LABEL	Type of element
ELEMENTTYPE_UNSIGNED	BOOL	True, if element is unsigned
ARRAYELEMENTS	INT	Number of array elements. Set for arrays if EXPAND_ARRAY==false
RANGE_MIN	INT	Element's minimum value
RANGE_MAX	INT	Element's maximum value
RANGE_LIST	LABEL; INT	Elelement's range list
RANGE_VALUE	INT; INT	Element's range value
ARRAY_DEFAULT	STRING	Default hex string for arrays or structures

Table 6-2 Configuration Attribute Types

6.1.2 Tags

A tag that will be added to the generated XML. Tag definitions have no arguments.

Attribute	Argument	Description
keyword	LABEL	An unambiguous label for this tag. Prefixes for Global, Module, Structure or Element will be added.
name	LABEL	With this name it will show up in the output XML. By default, name is set to the same as "keyword"
parent	LABEL	Attach this tag to a parent tag. By default, the attribute is attached to the last active tag.
newParent	BOOL	If true, this tag is the new active element. Subsequent attributes will attach to this tag.
addToFront	BOOL	Add element to begin of list (instead of appending it to the end)
show	BOOL	If true, show this attribute in XML output

Attribute	Argument	Description
maxOccurs	“unbounded” | INT	Maximum number of this Tag. If the maximum number of tags is reached, the latest Tag will be replaced by the new one
selectExisting	BOOL	If the maximum number of Tags is created (see maxOccurs), the last Tag will be selected and all subsequent attributes and sub-tags will be integrated into this tag.
removeRedundantTags	BOOL	Remove all sub-tags with similar content

Table 6-3 Configuration Tags

Example:

```
<TAG keyword="TAG_ModuleInfo" name="ModuleInfo" Parent="TAG_Block"/>
```

6.1.3 Aliases

Aliases define an easy-to-read syntax for the header annotations and can have multiple arguments or argument lists.

Attribute	Argument	Description
keyword	LABEL	An unambiguous label for this alias. Prefixes for Global, Module, Structure or Element will be added.
Arguments	ARGUMENTS	A number of argument types, separated by ‘;’. Please refer to the Arguments type table Configuration Arguments.
isList	BOOL	Argument list contains an unknown number of arguments defined by the ‘Arguments’ attribute.
action	STRING	Any attribute, element or alias commands. Arguments are referred by ‘\i’ (i being the i^{th} argument). The number of arguments can be referred by ‘\kn’
actionList	STRING	Same as attribute ‘action’, but will be repeated for each list entry
actionPost	STRING	Same as ‘action’ but will be executed after ‘actionList’

Table 6-4 Configuration Aliases

Example:

```
<ALIAS keyword="InputPins" Arguments="LABEL;INTEGER" isList="true"
   action="
    @h2xmlm_InputPinsMax={\kn}
    @h2xmlm_InputPinsIdSize={2}"
   actionList="@h2xmlm_InputPin{\1,\2}"
 />
```

This definition will allow to write the input pin list in the header file like this:

```
/**
    @h2xmlm InputPins                {In1=1;In2=2;In3=10}
*/
```

6.1.4 Configuration Arguments

Argument	Description
STRING	Any characters. A String argument cannot be mixed with any other argument types.
LABEL	Like a C-language variable, defined by regular expression `[a-zA-Z_][a-zA-Z0-9_.]*`. Note that labels will be replaced by the preprocessor if they have been defined by #define or within an enum statement. Arbitrary character strings that don't meet the former requirements can be used by placing the label into quotes (""). **Example:** test_labels.h test_labels.xml
INT, INTEGER	An integer number
UNSIGNED_INTEGER, UINT	An unsigned integer number
FLOAT	A floating point number
BOOL	Boolean, true or false

Table 6-5 Configuration Arguments

7 RTF Documentation

In addition to the xml output, an rtf (Rich Text Format) file can be generated using a template file. The template file must contain the following marks.

RTF Key Word	Description
[H2XML_MODULE_START]	Beginning of module
[H2XML_MODULE_END]	End of module
[H2XML_PARAMETER_START]	Beginning of parameter
[H2XML_ PARAMETER _END]	End of parameter
[H2XML_ELEMENT_START]	Beginning of element
[H2XML_ ELEMENT _END]	End of element

Table 7-1 RTF template key words

Annotation values are referred to by [@<keyword_name>]. Special formatting of the values can be applied by providing an optional formatting value after the keyword, separated by ':'. Formatting for each formatting value is hard coded in the h2xml source code.

A valid annotation reference would be for example:

```
[@h2xmle_default:33]
```

An example for a template file is given by rtf_template.rtf.

8 Master Configuration

An example configuration file is given in config_master.xml.

The header file annotations as well as the XML output format of config_master.xml is described in the following chapters. The syntax can easily be changed, new annotations added or the generated XML output be changed by modifying the config file.

8.2 Config_Master Global Annotations

Key Word	Arguments	Description
@h2xml_setVersion	LABEL dspType; INT versionMajor; INT versionMinor; INT version Branch; INT versionSubBranch	Adds version information to the XML file **Example:** h2xml_setVersion.h h2xml_setVersion.xml
@h2xml_titlex	STRING Title	Add an title attribute to the XML file, where x is in the range 1...10. this attribute can be used e.g. in the rtf file template. **Example:** h2xml_title.h h2xml_title.xml
@h2xml_select	LABEL Identifier	The following annotations refer to the selected identifier **Example:** test_external_parameters.h test_external_parameters.xml

Table 8-1 Config_Master Global Annotations

8.3 Config_Master XML Annotations

Key Word	Arguments	Description
@h2xmlx_xmlTabSize	INT tabSize	Tab size of the generated XML output. **Default:** 2 **Example:** h2xmlx_xmlTabSize.h h2xmlx_xmlTabSize.xml
@h2xmlx_xmlLineLength	INT length	Maximum line length of the generated XML output. **Default:** 160 **Example:** h2xmlx_xmlLineLength.h h2xmlx_xmlLineLength.xml
@h2xmlx_xmlAttributeLength	INT length	Maximum length of an XML attribute value. If this value is exceeded, a new-line is inserted in the value string. **Default:** 160 **Example:** h2xmlx_xmlAttributeLength.h h2xmlx_xmlAttributeLength.xml

Key Word	Arguments	Description
@h2xmlx_xmlNumberFormat	LABEL type={NORMAL, HEX, INT}	Number format used in XML file. **Default: HEX** ▪ NORMAL: for each number use same representation as in input header file. ▪ INT: all numbers in integer. ▪ HEX: all numbers in hexadecimal (with preceding 0x), negative numbers are 2th complement based on the byte size of the element. **Example:** h2xmlx_xmlNumberFormat_Normal.h h2xmlx_xmlNumberFormat_Int.h h2xmlx_xmlNumberFormat_Hex.h h2xmlx_xmlNumberFormat_Normal.xml h2xmlx_xmlNumberFormat_Int.xml h2xmlx_xmlNumberFormat_Hex.xml
@h2xmlx_expandarray	BOOL expand	Display all array elements in the generated output. **Default:** TRUE **Example:** h2xmlx_expandarray.h h2xmlx_expandarray.xml
@h2xmlx_expandStructs	BOOL expand	Expand structure to its elements for the generated output. **Default:** TRUE **Example:** h2xmlx_expandStructs.h h2xmlx_expandStructs.xml
@expandTypedefs	BOOL expand	Display base type instead of the typedef name **Default:** FALSE **Example:** h2xmlx_expandTypedefs.h h2xmlx_expandTypedefs.xml
@expandEnums	BOOL expand	Display the value of enums and #defines in the generated output. **Default:** TRUE **Example:**

Table 8-2 Config_Master XML Annotations

8.4 Config_Master Module Annotations

Key Word	Arguments	Description
@h2xmlm_module	LABEL name; INT id	Starts the beginning of a module with name='name' and ID='id' **Example:** h2xmlm_module.h h2xmlm_module.xml
@h2xmlm_displayName	STRING name	Desciptive short name of a module. **Example:** h2xmlm_displayName.h h2xmlm_displayName.xml
@h2xmlm_PinType	LABEL type={Static, Dynamic}	Pin Type. **Default:** Static **Example:** h2xmlm_PinType.h h2xmlm_PinType.xml
@h2xmlm_InputPins	LABEL name1; INT id1 [; LABEL name2; INT id2 [; …]]	Defines the input pin names and ids. Any number of name/id pairs can be specified. Note that for better readability the separator '=' can be used between name and id. **Example:** h2xmlm_InputPins.h h2xmlm_InputPins.xml
@h2xmlm_OuputPins	LABEL name1; INT id1 [; LABEL name2; INT id2 [; …]]	Defines the output pin names and ids. Any number of name/id pairs can be specified. Note that for better readability the separator '=' can be used between name and id. **Example:** h2xmlm_OutputPins.h h2xmlm_OutputPins.xml
@h2xmlm_toolPolicy	LABEL policy1 [; LABEL policy2 [; …]]	toolPolicy must be a combination of {CALIBRATION, RTC, RTM, NO_SUPPORT, RTC_READONLY} **Example:** h2xmlm_toolPolicy.h h2xmlm_toolPolicy.xml
@h2xmlm_description	STRING	Any description string. **Example:** h2xmlm_description.h h2xmlm_description.xml
@h2xmlm_replacedBy	LABEL newModule	Module was replaced by <newModule> **Example:** h2xmlm_replacedBy.h h2xmlm_replacedBy.xml
@h2xmlm_replaces	INT id1 [; INT id2 [; …]]	Specifies a list of Module IDs that are replaced by this module. **Example:** h2xmlm_replaces.h h2xmlm_replaces.xml

Key Word	Arguments	Description
@h2xmlm_deprecated	BOOL deprecated	Module has been deprecated **Example:** h2xmlm_deprecated.h h2xmlm_deprecated.xml
@h2xmlm_InsertParameter	None	Insert selected Parameter (which was globally defined outside the module) into this module. **Example:** test_external_parameters.h test_external_parameters.xml test_external_parameters2.h test_external_parameters2.xml
@h2xmlm_InsertStructure	None	Insert selected Structure (which was globally defined outside the module) into this module. This is the same as inserting a Parameter. **Example:** test_external_subStruct.h test_external_ subStruct.xml

Table 8-3 Config_Master Module Annotations

8.4.1 Using global Parameters within a Module

Parameters that are globally defined and will be used by various modules can be attached to a module using the '@h2xmlm_InsertParameter' keyword. Annotations of the global parameter can be overridden within the module.

Example:

```
//<!-- parameter defined outside of module   -->
/** @h2xmlp_parameter {ExternalParameter1, 0x12345678} */
typedef struct {
    int exa;        /**< @h2xmle_default        {0x1}*/
    int exb;        /**< @h2xmle_default        {0x2}*/
} extParam1;


/*-------------------------------------------------------------*//**
    @h2xmlm_module                  {MODULE_1,0x11111111  }
    @{
-------------------------------------------------------------*/
/**
    <!-- within the module select the parameter -->
    @h2xml_Select           {extParam1}
    <!-- add the parameter to the module        -->
    @h2xmlm_InsertParameter
    <!-- original annotations can be overridden -->
    @h2xml_Select           {extParam1::exa}
    @h2xmle_default         {11}
*/
/**@}          <!-- End of Module --> */
```

Examples:

test_external_parameters.h
test_external_parameters.xml
test_external_parameters2.h
test_external_parameters2.xml

8.5 Config_Master Parameter Annotations

Key Word	Arguments	Description
@h2xmlp_parameter	LABEL name; INT id	Defines the referenced structure as a Parameter with name='name' and ID='id' **Example:** h2xmlp_parameter.h h2xmlp_parameter.xml
@h2xmlp_emptyParameter	LABEL name; INT id	Defines a Parameter without payload, e.g. a message, with name='name' and ID='id' **Example:** h2xmlp_emptyParameter.h h2xmlp_emptyParameter.xml
@h2xmlp_subStruct		The referenced structure will be inserted int o the XML. This is the same as defining a Parameter, but without ID and name. In the XML, the isSubStruct flag is set to 'true'. **Example:** h2xmlp_subStruct.h h2xmlp_ subStruct.xml
@h2xmlp_version	INT version	Defines the parameter version **Example:** h2xmlp_version.h h2xmlp_version.xml
@h2xmlp_description	STRING	Any description string. **Example:** h2xmlp_description.h h2xmlp_description.xml
@h2xmlp_toolPolicy	LABEL policy1 [; LABEL policy2 [; …]]	toolPolicy must be a combination of {CALIBRATION, RTC, RTM, NO_SUPPORT, RTC_READONLY } **Example:** h2xmlp_toolPolicy.h h2xmlp_toolPolicy.xml
@h2xmlp_readOnly	BOOL read	Defines a read-only element. **Example:** h2xmlp_readOnly.h h2xmlp_readOnly.xml

Key Word	Arguments	Description
@h2xmlp_deprecated	BOOL deprecated	Module has been deprecated **Example:** h2xmlp_deprecated.h h2xmlp_deprecated.xml

Table 8-4 MASTER Parameter Annotations

8.6 Config_Master Element Annotations

Key Word	Arguments	Description
@h2xmle_name	LABEL name	Per default, the name of an element with which it will show up in the XML is taken from the code. The name can be redefined with 'name'. **Example:** h2xmle_name.h h2xmle_name.xml
@h2xmle_policy	LABEL policy={basic, advanced, internal }	Element policy must be one of {basic, advanced, internal}. **Example:** h2xmle_policy.h h2xmle_policy.xml
@h2xmle_default	INT value	Defines the default value. **Default:** 0 **Example:** h2xmle_default.h h2xmle_default.xml
@h2xmle_defaultFile	STRING file	Specifies a binary file that is used to initialize an array or structure. The file will be searched for in the include directories specified with the –I command line option. The size of the file must match the array or structure size. **Example:** h2xmle_defaultFile.h h2xmle_defaultFile.xml
@h2xmle_defaultList	INT[;INT[…]]	Specifies a list of default values to initialize an array or structure. The number of elements must match the array or structure size. **Example:** h2xmle_defaultList.h h2xmle_defaultList.xml
@h2xmle_visibility	LABEL visibility={show, hide}	Visibility must be one of {show, hide}. **Example:** h2xmle_visibility.h h2xmle_visibility.xml

Key Word	Arguments	Description
@h2xmle_readOnly	BOOL read	Defines a read-only element. **Example:** h2xmle_readOnly.h h2xmle_readOnly.xml
@h2xmle_isVersion	BOOL version	This element defines the parameter version. **Example:** h2xmle_isVersion.h h2xmle_isVersion.xml
@h2xmle_description	STRING	Any description string. **Example:** h2xmle_description.h h2xmle_description.xml
@h2xmle_dataFormat	LABEL format={Q20, Q18, Q8}	Format must be one of {Q20, Q18, Q8}. **Example:** h2xmle_dataFormat.h h2xmle_dataFormat.xml
@h2xmle_range	INT min; INT max	Defines a parameter range. Default is MIN and MAX of the element type. Note that for better readability the separator '..' can be used between min and max. **Example:** h2xmle_range.h h2xmle_range.xml
@h2xmle_rangeList	LABEL name1; INT id1 [; LABEL name2; INT id2 [; …]]	Defines discrete values, e.g. {disable=0; enable=1}. Note that for better readability the separator '=' can be used between name and id. **Example:** h2xmle_rangeList.h h2xmle_rangeList.xml
@h2xmle_bitfield	UINT mask	Starts a bit field with mask 'mask. All subsequent annotations will refer to this bit field. A bit field definition must be terminated with the @h2xmle_bitfieldEnd command. **Example:** h2xmle_bitfield.h h2xmle_bitfield.xml
@h2xmle_bitfieldEnd	NONE	Ends a bit field which has been defined with the @h2xmle_bitfield command. **Example:** h2xmle_bitfield.h h2xmle_bitfield.xml
@h2xmle_increment	INT increment	Defines an increment for this element value. **Example:** h2xmle_increment.h h2xmle_increment.xml

Key Word	Arguments	Description
@h2xmle_displayType	LABEL policy={dump; checkBox; slider; dropDown; bitfield; file}	DisplayType must be one of { dump; checkBox; slider; dropDown; bitfield; file }. **Example:** h2xmle_displayType.h h2xmle_displayType.xml
@h2xmle_rawData	BOOL rawData	Element contains raw data. **Example:** h2xmle_rawData.h h2xmle_rawData.xml
@h2xmle_ variableArraySize	LABEL size	Element policy must be one of {basic, advanced, internal}. **Example:** h2xmle_variableArraySize.h h2xmle_variableArraySize.xml
@h2xmle_ defaultDependency	LABEL list	Add dependencies for default values. Defaults may be single values, a default list (see h2xmle_defaultDependency_list.h) or a default file (see h2xmle_defaultDependency_file.h). **Example:** h2xmle_defaultDependency.h h2xmle_defaultDependency.xml h2xmle_defaultDependency_List.h h2xmle_defaultDependency_List.xml h2xmle_defaultDependency_File.h h2xmle_defaultDependency_File.xml

Table 8-5 Config_Master Parameter Annotations

8.6.1 Coding convention for dynamic annotation values

A value of an annotation may depend on the value of another element. In the following example, the size of w_struct::w depends on FSTV::N.

```
struct FSTV
{
      int16_t N;        //controls size of array inside a structure
}

struct w_struct
{
      int32_t idt;
      int32_t w[0];     /**@h2xmle_variableArraySize {2*FSTV::N} */
}
```

To allow QACT to parse the formula and resolve the relationship, the delimiter "**::**" is mandatory to indicate that "N" belongs to structure "FSTV".

9 H2XML Internals

The H2XML tool consists of

- Configuration file parser
- Preprocessor, which resolves #includes, #defines and conditional compilation (#if, #ifdef etc.)
- A Flex and Bison based Compiler, which parses the declarations and attaches annotations to them. The compiler is actually not limited to header files, but could also parse C source code. Since this feature is not used at the moment, it is not tested (and may therefore not work as expected)
- A XML generator that writes all annotations to an XML file according to the specifications in the configuration file.

Note: Instead of using the internal preprocessor, an external preprocessor (like GCC) could be used to resolve the preprocessing directives. An external preprocessor will resolve any #defines, so they cannot be used in the annotations. Enumerations will always be resolved by the compiler, so they can be used with the internal or an external preprocessor.

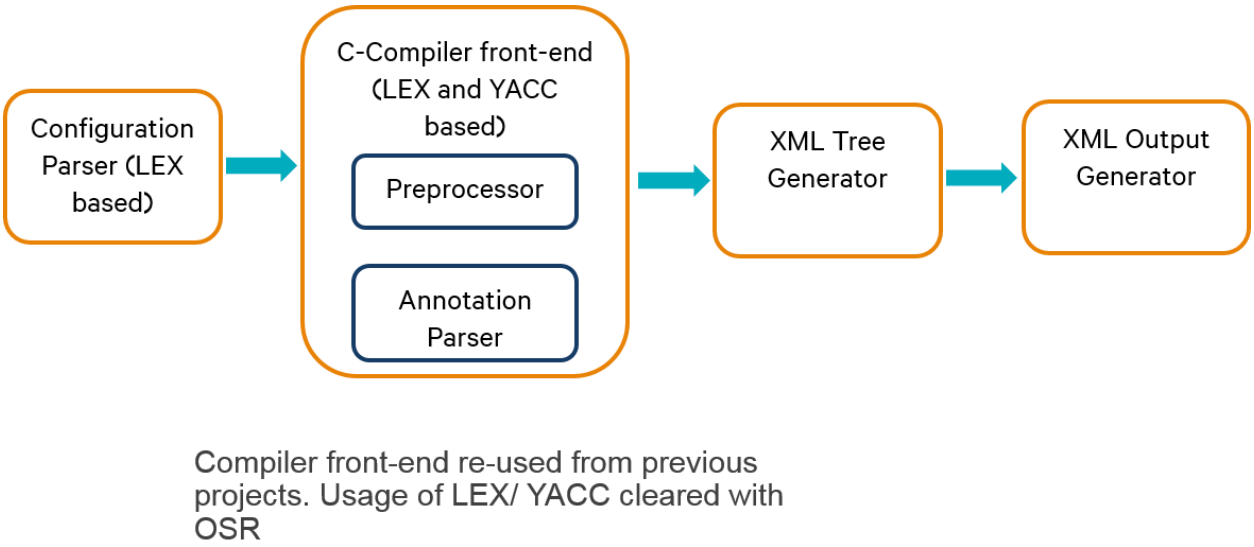


Figure 9-1 H2XML Internals

A References

A.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
Standards	
Resources	

A.2 Acronyms and terms

Acronym or term	Definition