Chapter 1

Requirements

This chapter describes an utility that creates Wireshark dissectors from C header files. The dissectors must interpret binary representations of C structs. In section 1.1 we give a high level overview of the utility and its requirements, section 1.2 provides use cases for the utility, section 1.3 explains how we prioritizes the requirements, section 1.4 explains how we estimate their complexity, section 1.5 lists all the functional and non-function requirements, and section 1.6 contains the complete product backlog.

1.1 Overview

We are to create an utility that allows Wireshark to interpret the binary representations of C-language structs. While C structs seldom are exchanged across networks, they are sometimes used in inter-process communication. The purpose of the utility described here is to provide Wireshark with the capability of automatically dissecting the binary representation of a C struct, as long as its definition is known.

The expected work flow for the utility is to read one or more C header files, which contain struct definitions, and output Wireshark dissectors, implemented in Lua scripts. A configuration file or source code annotations in the header files may be used when additional configuration is required.

1.1.1 List of requirements

Table 1.1 is an overview of all the requirements. See Table 1.2 and Table 1.3 for more detailed description of the requirements.

Table 1.1: Requirements overview

ID	Description
FR1	Read basic C struct definitions
FR1-A	Support data types: int, float, char and boolean
FR1-B	Support members of type enums
FR1-C	Support members of type structs
FR1-D	Support members of type unions
FR1-E	Support member of type array
FR2	Generate Wireshark dissectors in Lua
FR2-A	Display simple structs
FR2-B	Support display of structs within structs
FR2-C	Support Wireshark filter and search on attributes
FR3	Support C preprocessor directives and macros
FR3-A	Support #include
FR3-B	Support #define and #if
FR3-C	Support `WIN32`, `_WIN64`, `__sparc` etc
FR4	Support user configuration
FR4-A	Recognize invalid values for struct members
FR4-B	Support enumerated named values or a bit strings
FR4-C	Custom handling of specific data types
FR5	Structs with headers and/or trailers
FR6	Handle input which size and endian depends on platform
FR6-A	Flags specified for each platform
FR6-B	Flags which signal the platform
FR7	Support parameters from command line
FR7-A	Support parameters for c-header file
FR7-B	Support for configuration file
FR7-C	Support batch mode of c-header and configuration
FR7-D	Don't regenerate dissectors
NR1	Run on latest Windows & Solaris OS
NR2	Dissector run on Windows & Solaris, Intel & Sparc
NR3	User interface shall be command line
NR4	Sufficient documentation for generating Lua-scripts
NR5	Sufficient documentation for extending functionality
NR6	Code should follow PEP8 and PEP20
NR7	Code should be documented by docstrings

1.2 Use Cases

1.2.1 Actors

An actor specifies a role played by an external person or thing that interact with our utility. We have three types of actors to consider. First is the

primary actor which in our case is the user of our utility. He who feeds it a C file to generate dissectors. A secondary actor is someone who configures our utility to change the output of it. Finally we have an offstage actor which does not use our utility himself, but uses the output dissectors in Wireshark.

We have defined two use case actors for our utility. The customer has specified that the user is the most important actor.

User User of the generated Wireshark dissectors, offstage actor

Developer User and configurer of utility, primary and secondary actor

1.2.2 Use Case Diagrams

TODO!! Desperately need help for this....

1.3 Prioritization

The team has, in cooperation with the customer, prioritized the requirements in three categories: *a*) High, *b*) Medium or *c*) Low.

High Core functionality of the utility which must be implemented.

Medium Requirements that will improve the value of the utility.

Low Requirements that will not add much value to the utility.

1.4 Complexity

The team has estimated the complexity for each requirement. We use the same categories as for requirements priority: *a*) High, *b*) Medium or *c*) Low.

High Functionality which seems difficult and non-trivial to create.

Medium Functionality that seems time consuming but straight forward.

Low Requirements that are trivial to implement.

1.5 List of requirements

Table 1.2 lists the functional requirements, while Table 1.3 lists the non-functional requirements. Each requirement have a priority (Pri) and a complexity (Cmp): High (H), Medium (M) or Low (L).

1.6 Production backlog

Table 1.4 contains a complete product backlog.

Table 1.2: Functional Requirements

ID	Description	Pri.	Cmp.
FR1	The utility must be able to read basic C language struct definitions from C header files	H	
FR1-A	The utility must support the following basic data types: int, float, char and boolean	H	L
FR1-B	The utility must support members of type enums	H	L
FR1-C	The utility must support members of type structs	H	M
FR1-D	The utility must support members of type unions	M	M
FR1-E	The utility must support member of type array	H	M
FR2	The utility must be able to generate lua-script for Wireshark dissectors for the binary representation of C struct	H	
FR2-A	The dissector shall be able to display simple structs	H	L
FR2-B	The dissector shall be able to support structs within structs	M	M
FR2-C	The dissector must support Wiresharks built-in filter and search on attributes	H	L
FR3	The utility must support C preprocessor directives and macros	H	
FR3-A	The utility shall support #include	H	L
FR3-B	The utility shall support #define and #if	H	L
FR3-C	The utility shall support `WIN32`, `_WIN32`, `_WIN64`, `__sparc__`, `__sparc` and `sun`	M	H
FR4	The utility must support user configuration	M	
FR4-A	The dissector shall be able to recognize invalid values for a struct member. Allowed ranges should be specified by configuration	L	L
FR4-B	Configuration must support integer members which represent enumerated named value or a bit string	M	L
FR4-C	Configuration must support custom handling of specific data types. E.g. a 'time_t' may be interpreted to contain a unixtime value, and be displayed as a date	L	M
FR5	A struct may have a header and/or trailer (other registered protocol). The configuration must support the use of integer members to indicate the number of other structs that will follow in the trailer	L	H
FR6	The dissectors must be able to handle binary input which size and endian depends on originating platform	M	
FR6-A	Flags must be specified for each platform	M	M
FR6-B	Flags within message headers should signal the platform	M	H
FR7	The utility shall support parameters from command line	H	
FR7-A	Command line shall support parameters for c-header file	H	L
FR7-B	Command line shall support for configuration file	H	L
FR7-C	Command line shall support batch mode of c-header and configuration file	L	M
FR7-D	When running batch mode, dissectors that already are generated, shall not be regenerated, if the source are not modified since last run	L	M

Table 1.3: Non-Functional Requirements

ID	Description	Pri.	Cmp.
NR1	The utility shall be able to run on latest Windows and Solaris operating system	M	L
NR2	The dissector shall be able to run on Windows x86, Windows x86-64, Solaris x86, Solaris x86-64 and Solaris SPARC	M	M
NR3	The utilities user interface shall be command line. No clicking!	H	L
NR4	The configuration shall have sufficient documentation to allow a person with no previous knowledge of the system to be able to use it to generate LUA-scripts after X hours of reading	M	M
NR5	The configuration should have sufficient documentation to allow a person, already proficient with the system, to understand the code well enough to be able to extend it's functionality after Y hours of reading	M	M
NR6	The utility code should follow standard python coding convention as specified by PEP8, and try to follow python style guidelines defined by PEP20	H	L
NR7	The utilities code should be documented by python docstrings which should explain the use of the code. Python modules, classes, functions and methods should have docstrings	M	L

Table 1.4: Product Backlog

		Hours	
Req.	Description	Est.	Act.
FR1-A	Support basic data types: int, float, char, boolean	5	-
FR2-A	The dissector shall be able to display simple structs	10	-
FR3-A	The utility shall support #include	1	-
FR3-B	The utility shall support #define and #if	1	-
FR4-A	Recognition of invalid values for a struct member	20	-
FR7-A	Command line shall support parameters for c-header file	2	-
FR7-B	Command line shall support for configuration file	5	-