mbeddr Header Importer Architecture

How Header Importer Works?

Mohammadreza Basirati

January 20, 2015

1 Introduction

This document gives a brief but complete description about how the tool Header Importer works. It assumes that you are familiar with lexical scanner, lexical parser and Jetbrains MPS[1].

Header Importer is a tool to import declarations from C header files into an mbeddr[2] project, so developers can use them in their code.

The process of importing a header file into an mbeddr project goes in three steps: 1. Scanning header files 2. Parsing scanner tokens 3. Importing declarations into mbeddr external module structure. We will discuss each step in detail in the following sections.

Step one and two work along in a Java project to prepare the required input for step three. The header importer tool Java project is located in folder bparser. Our scanner generator produces lexer.java file and our parser generator produces two files: sym.java and parser.java. Lexer.java will tokenize the input file. Parser.java file can recognize the tokens by their identifier which has been defined in sym.java file. The last part of the Java project consists of classes which will be used to keep header file declarations. At the last stage, MPS importer get the declarations and import them into an mbeddr external module(Figure 1).

2 Scanner

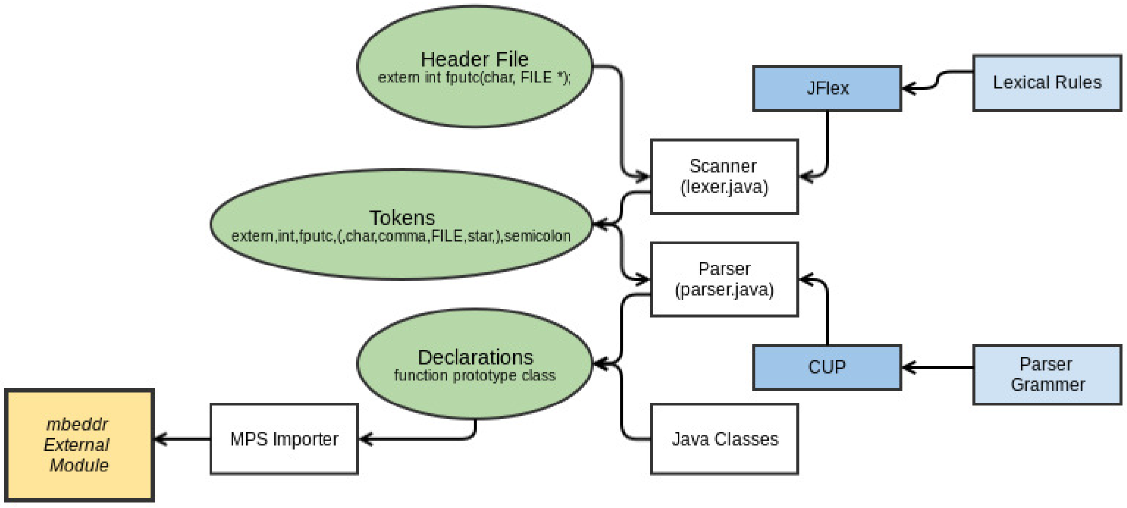
The first stage is to tokenize the header file. For this task we used JFlex[3] scanner. You can find the scanner files for the header importer inside the `"scanner_parser"` folder. For compatibility issue of our tool over different versions of header files, we enhanced the scanner to be able to tokenize the gcc stdio.

[1]More information about MPS on https://www.jetbrains.com/mps/

[2]More information about mbeddr on http://mbeddr.com/

[3]JFlex Web Site http://jflex.de/

Figure 1: Header Importer Architecture



The scanner has three states which can identify one line comments, multiple lines comments, and strings. All the charachter sequences tokens that the scanner returns consists of: all possible operators in c, define, undef, if, ifdef, ifndef, else, endif, include, extern, pragma, typedef, struct, all c types (int, char, etc.), numbers , and identifiers. Right now, the parser does not use all operators' tokens, but they can be used in future for possible improvement.

The scanner can recognize the compiler preprocessing words which begin with double underscore and returns the token COMPWORD for them. Other compiler preprocessing phrases will be ignored by the scanner.

The lines that begin with define keyword for declaring macros and constants, scanner returns the whole line to the parser and doesn't tokenize their expression. In the parser section we will discuss this issue in more detail.

3 Parser

The parser has been generated by CUP[4]. JFlex which we used for generating our scanner, is designed to work togetegher with CUP. Basically the parser gets tokens from the scanner and matches them by the grammers which we defined in our .cup file. The parser files of header importer located in `"scanner_parser"` folder.

The grammer of the parser consists of two main parts: preprocessing steps and general declarations. Preprocessing steps comprise of the define, if, ifdef,

[4]CUP Web Site http://www2.cs.tum.edu/projects/cup/

ifndef, else, endif, include, and COMPWORD token that is the token returned by scanner for preprocessing words used by compilers.

General declaration part, consists of 4 type of declaration: typedef declaration, struct declaration, variable declaration, and function prototype declaration.

3.1 Declaration Classes

In the Java project we have several classes for managing and keeping the declarations. These classes are used by parser to define declarations and used by MPS importer part to get the declarations and import them into mbeddr module. In this section we will have a description for each class and how they colaborate in the parsing phase.

CodeGenerator	CodeGenerator class is responsible for being a proxy between the parser and MPS part. The parser export parsed declarations by calling CodeGenerator methods.On the other hand, the MPS part gets the declarations from CodeGenerator list of all declarations. Besides, CodeGenerator class has the responsibility of taking care of if blocks and structs.
Declaration	Declaration is an abstract class which all other classes execpt for CodeGenerator, extend it.
ConditionalBlock	This class is used for defining if blocks. It has a two list of declarations: true block and false block.
Define	It is used for define declarations.
Function	It keeps the information for function declarations and function pointers. It has a list of parameters, a return type, and a name like all other declarations.
Variable	It defines a variable declaration.
Struct	It has a list of struct declarations and a name.
Typedef	It defines a typedef declaration.
Include	It holds a name for what should be included.

4 MPS Importer