University College London - Department of Computer Science

JSON Parser Using JFlex and CUP

Student
German Malsagov

Word Count: 576

November 11, 2017

1 Overview

The system I designed allows to perform the syntax analysis of JSON output according to the official JSON specifications available at `www.json.org`. This process is split into two phases: lexical analysis, often referred to as lexing or scanning, and syntax analysis (parsing). Thus, I created two components in order to accomplish these processes: a JFlex lexer (*Scanner.jflex*) and a CUP parser (*Parser.cup*). The model accepts input from the text file (input.text) and checks the validity of JSON components. If the input contains invalid objects, a default error message will be displayed. Successful parsing will be indicated by printing corresponding message in the panel window.

Requirements:

User must install the most recent Java SDK and Apache Ant in order to compile and run the Parser.

How to Execute:

Make sure that you changed your directory to the Parser's folder. To execute the compiler, type the following commands into your terminal window:

- ant jar
- java -jar jar/Compiler.jar input.test

To change your input file, replace input.test by the name of desired test file located in the same folder.

2 Scanner File

The Scanner performs lexical analysis of the software that the user is trying to compile. In essence, lexer recognizes words and symbols in the source code and converts them into a stream of tokens (a sequence of characters that represents a grammatically correct unit in programming language). A common way to describe these tokens is by using *regular expressions*, or *regex*, as given by ECMA-404 JSON standards available in the link mentioned above. Once regex is matched, corresponding action is performed. At this stage, comments and blank spaces are eliminated as well.

Initially, basic regex macros were defined ('digit' and 'char') which resulted in forming more complex macros such as 'number' and 'string'. This allowed for the construction of the tokens which then can be inserted into the parser file. The scanner also scans for permitted JSON punctuation tokens (such as brackets, commas and colons) and outputs appropriate symbols to the parser file.

3 Parser File

Within the parser file, grammar rules and tokens (terminal and nonterminal symbols) are defined. The terminal symbols are fundamental symbols characterised by a formal grammar (with no associated values such as punctuation grammar and with associated values such as String String and Integer number), whilst non-terminal symbols are created from terminal symbols using grammar rules. The same notation was used as in the JSON.org website given in the coursework brief for symbols and grammar rules. To determine if a parsing was successful, the console prints: "Parsing completed successfully. No errors were detected." and if there is an error, it prints the default error messages for syntax and illegal char errors.

4 Tests

The tests that I used in order to verify the accuracy of our system include the official examples of messages from JSON website. The use of them allowed us to check how our code responds to nested arrays and objects. In addition, we've conducted tests for Booleans, positive and negative exponents, two-character escape sequences and others. I also tested my code for some common user errors such as missed comma or colon, extra comma, incomplete closure of an object or array etc. Our model successfully passed all valid JSON tests and failed invalid ones. Further description and results of all the tests that were conducted could be found in test-table.pdf. The test files were attached for the readers to try themselves.