JSON Data Injection in C++

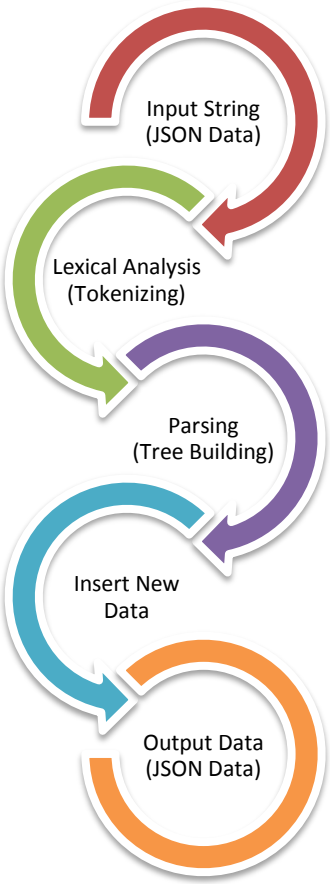
Clayton Crawford & Samuel Carey

Summary

This document will serve as a general reference on how to receive JSON data into a C++ program, manipulate the data by adding new information, and then outputting the updated data to the console/terminal or separate file. This process involves four subtasks:

- Receiving JSON input data (specifically through cin >>)
- Converting the data into usable objects (lexical analysis and parsing)
- Editing the data (inserting into tree)
- Outputting data in JSON format (specifically through cout <<)

This document will also mention the use of a JSON C++ parser library called JsonCpp.



JSON Input Data (istream / cin >>)

C++ has various classes built in that can handle input data. The 'istream' class has an object called 'cin' that is used as the standard input stream. The common input methods using 'cin' are from the keyboard or a file. Assuming that the JSON input data will be located in some file, one can simply execute their program with the input file being piped into the program. An example of this would be:

```
// > cat inputFile.txt | main.out

#include <iostream>
using namespace std;

int main() {
      string lineInput;
      while (cin >> lineInput) {
        cout << lineInput;
      }
      return 0;
}
```

JSON Data Injection in C++

Clayton Crawford & Samuel Carey

Lexical Analysis and Parsing

Once the program is setup to receive JSON input data, the program will then need a way to convert the series of characters and strings into a usable and meaningful structure. This process begins by introducing a lexical analyzer, also called a tokenizer, which converts a sequence of characters into a sequence of tokens. Once the data has been properly tokenized, the tokens then need to be put into a hierarchical data structure (tree) by means of a parser. The parser will give each token some sort of identifying characteristic so that it will know how to populate the tree and where to insert each node (token).

People have taken it upon themselves to build their own tokenizer and parser libraries and are nice enough to distribute them as open source. One such library is called JsonCpp. It can take JSON input and put it into a tree so that the programmer can use the data as they see fit.

Inserting New Data

Insertion of new data will be performed by an insertion function of the data structure that is used to store the JSON data. An algorithm may have to be worked out in order to insert the new data at the correct spot in the existing data. Most open source parsing libraries have functions to facilitate the insertion of new data.

Outputting JSON Data (ostream / cout <<)

Similar to ‘istream’ and ‘cin’, C++ has ‘ostream’ and its member ‘cout’ to output to a console or file. A traversal algorithm would need to be used in order to output the JSON data correctly. This algorithm may also be responsible for adding various formatting options such as braces, colons, quotations, spacing, tabbing, and etc. to meet the JSON standards and schema.

JSON Data Injection in C++

Clayton Crawford & Samuel Carey

Installation of JsonCpp

Step 1. Download the amalgamated version of JsonCpp:

i. http://sourceforge.net/projects/jsoncpp/files/jsoncpp/0.6.0-rc2/jsoncpp-src-0.6.0-rc2-amalgamation.tar.gz/download

ii. This parser has some very easy-to-use functions. The newest version 0.6 can combine (amalgamate) all of its headers and source files into just 1 header and 1 source file. No library installation required. However, a typo must first be corrected in Step 4.

Step 2. Untar into your project folder.

Step 3. Open '**jsoncpp-src-amalgamation0.6.0-rc2/json/json.h**'

Step 4. Change line: '**#define JSON_IS_AMALGATED**'
To: '**#define JSON_IS_AMALGAMATION**' and save. The bug has been fixed.

Step 5. Add '**#include <json/json.h>**' to the top of your .cpp file.

Step 6. Add the folder '**jsoncpp-src-amalgamation0.6.0-rc2/**' to the include paths for your project.

Step 7. Add '**jsoncpp-src-amalgamation0.6.0-rc2/jsoncpp.cpp**' to your list of source files to be compiled.

Step 8. Implement JsonCpp functionality into your other source files.

Quick test from the Terminal application (Mac/UNIX/Linux):

Download the **jsoncpp-src-amalgamation0.6.0-rc2** folder under **shared/DataInjection** on lemur. From within that folder, execute these commands:

```
$  g++ -I. test.cpp jsoncpp.cpp
$  ./a.out < input.txt
```

This will build the included test program and read in a test JSON file.
These files, roughly adapted from the examples on the JsonCpp project home page, show the fundamental functionality of the library, but feel free to experiment with the more complicated implementations.