GetPot and JSON

Paolo Joseph Baioni
paolojoseph.baioni@polimi.it

March 7, 2025

Intro to Getpot

GetPot

Input File and Command Line Parsing

Web page

- `http://getpot.sourceforge.net/`

GetPot

GetPot is a *header file only* library, to facilitate command line and config file parsing. Useful for changing algorithm parameters without recompiling etc. . .

GetPot provides a class to parse `argc` and `argv` in alternative to POSIX standard `getopt` (in `C`).

Passing parameters directly on the command line

```
#include "GetPot"

#include <iostream>

int main(int argc, char **argv) {
  GetPot command_line(argc, argv);

  const double a          = command_line("a", 0.);
  const double b          = command_line("b", 1.);
  const int    n_intervals = command_line("n_intervals", 10);

  std::cout << a << " " << b << " " << n_intervals << std::endl;

  return 0;
}
```

Parameters are read via the call `operator()` for the `GetPot` class. It requires the name of the parameter to be read and the default value. The type is deduced from the class of the default value.

Run

```
./main
./main a=10 b=70
./main b=70.5 n_intervals=100
./main n_intervals=100 a=10
```

Sorting of command line arguments is not relevant.

Configuration files

```
#--------------------------------------------------
# Data file for numerical integration.
#--------------------------------------------------
[integration]
    [./domain]
        a = 2.0
        b = 4.0
    [../]

    [./mesh]
        n_intervals = 100
    [../]
[../]
```

How to parse from C++

```
#include "GetPot"

#include <iostream>

int main(int argc, char **argv) {
  GetPot datafile("data");

  const double a          = datafile("integration/domain/a", 0.);
  const double b          = datafile("integration/domain/b", 1.);
  const int    n_intervals = datafile("integration/mesh/n_intervals", 10);

  std::cout << a << " " << b << " " << n_intervals << std::endl;

  return 0;
}
```

How to parse from C++

```
#include "GetPot"

#include <iostream>
#include <string>

int main(int argc, char **argv) {
  GetPot datafile("data");

  const std::string global_section = "integration/";
  const std::string section1       = global_section + "domain/";
  const std::string section2       = global_section + "mesh/";

  const double a = datafile((section1 + "a").data(), 0.0);
  const double b = datafile((section1 + "b").data(), 1.0);
  const int n_intervals = datafile((section2 + "n_intervals").data(), 10);

  std::cout << a << " " << b << " " << n_intervals << std::endl;

  return 0;
}
```

Why not both?

The name of the config file may be passed on the command line

```
#include "GetPot"

#include <iostream>
#include <string>

int main(int argc, char **argv) {
  GetPot            command_line(argc, argv);
  const std::string filename = command_line.follow("data", 2, "-f", "--file");

  GetPot            datafile(filename.c_str());
  const std::string section = "integration/domain/";

  const double a = datafile((section + "a").data(), 0.0);
  const double b = datafile((section + "b").data(), 0.0);

  std::cout << "Integration range: \n" <<
      "a = " << a << "\n" <<
      "b = " << b << std::endl;
```

Run

```
./main -f data1
./main --file data1
./main
```

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript programming language. Gain traction to exchange data between client and server, nowdays is very widely used and supported by many languages.

JSON object is an unordered set of name/value pairs. An object begins with (left brace) and ends with (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).

A value can be one of the following

- object itself
- list
- string
- boolean
- number
- `null`

JSON

`https://github.com/nlohmann/json` is a C++ library with

- Intuitive syntax: it behaves just like an STL container. In fact, it satisfies the ReversibleContainer requirement.
- Trivial integration. Our whole code consists of a single header file json.hpp
- Memory efficiency. Each JSON object has an overhead of one pointer (the maximal size of a union) and one enumeration element (1 byte). The default generalization uses the following C++ data types: `std::string` for strings, `int64_t`, `uint64_t` or double for numbers, `std::map` for objects, std::vector for arrays, and `bool` for Booleans.

A simple example

```cpp
// need to install the json submodule, after can compile with
// g++ ex1.cpp -I../../../../../Examples/include
#include <fstream>
#include <iostream>

#include "json.hpp"

using json = nlohmann::json;

int main() {
  std::ifstream f("data.json");
  json data = json::parse(f);
  const double a = data["domain"].value("a", 0.0);
  const double b = data["domain"].value("b", 1.0);
  const unsigned int n_intervals = data["mesh"].value("n_intervals", 10);
  std::cout << a << " " << b << " " << n_intervals << std::endl;
  return 0;
}
```