

Hardware and System Specification Command-Line-Interface Tool

Evan Elias Young
College of Engineering and Computing
Missouri University of Science and Technology
Rolla, Missouri 65401–3066
Email: eeymrr@umsystem.edu

Abstract—There are many tools available to retrieve system specifications, however many of these lack cross-platform support and machine-readable output. Many different operating systems include a command-line-interface to retrieve some part of the information, however this is often not human-readable. A single tool which works on every operating system, includes human-readable output, and implements native libraries to quickly retrieve accurate information is necessary. Working together many different system libraries, formats, data types, and parsing revealed which operating systems include the most legacy code, which operating systems are the most streamlined, and the way in which libraries come together to form a cohesive application.

I. BACKGROUND

This demo file is intended to serve as a “starter file” for IEEE conference papers produced under L^AT_EX using IEEE-tran.cls version 1.8b and later. I wish you the best of success.

mds
August 26, 2015

A. Subsection Heading Here

Subsection text here.

1) Subsubsection Heading Here: Subsubsection text here.

II. LIBRARIES

A. JSON for Modern C++

JSON parsing was desired as it is easily machine readable and very portable, therefore some implementation of JSON was required. *JSON for Modern C++* was chosen as a single-header option, written for C++17 it includes very modern features and encourages the use of these quite well. In addition to modern features, the structuring and destructing is simple to implement and very efficient.

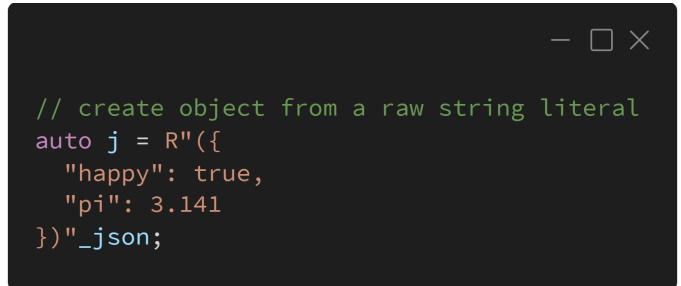
B. *argparse*

For a command-line interface, argument parsing was necessary to differentiate queries, listings, and selection. *argparse* was chosen as a single-header option, written for C++17 it includes very modern features and encourages the use of these quite well. In addition to modern features, the implementation and usage is very similar to python and is quite efficient too.

III. RESULTS

IV. CONCLUSION

The conclusion goes here.



```
// create object from a raw string literal
auto j = R"({
    "happy": true,
    "pi": 3.141
})"_json;
```

Fig. 1. Declaration of simple JSON object through the use of raw string literal suffixes.



```
// create base argument parser
argparse::ArgumentParser args("cspec", "0.1.0");

// add human-friendly argument
args.add_argument("--human", "-H")
    .default_value(false)
    .implicit_value(true)
    .nargs(0)
    .help("displays values in a human-friendly format");

// add main action argument
args.add_argument("action")
    .help("...")
    .required()
    .action(
        [](const string &val) -> string
        {
            if (!val == "list" || val == "get")
                throw std::invalid_argument("...");
            return val;
        });
```

Fig. 2. Declaration of argument parser object with one optional flag and one positional action.

ACKNOWLEDGMENT

I would like to thank Niels Lohmann et. al. for their contributions to *JSON for Modern C++*, and Pranav et. al. for their contributions to *argparse*.

REFERENCES

- [1] S. Whims, V. Kents, et. al., *Example: Getting WMI Data from the Local Computer*. Seattle, Washington: Microsoft, 2021.
- [2] G. Dicanio, *Use Modern C++ to Access the Windows Registry*, vol. 32 no. 5. Seattle, Washington: Microsoft, 2017.