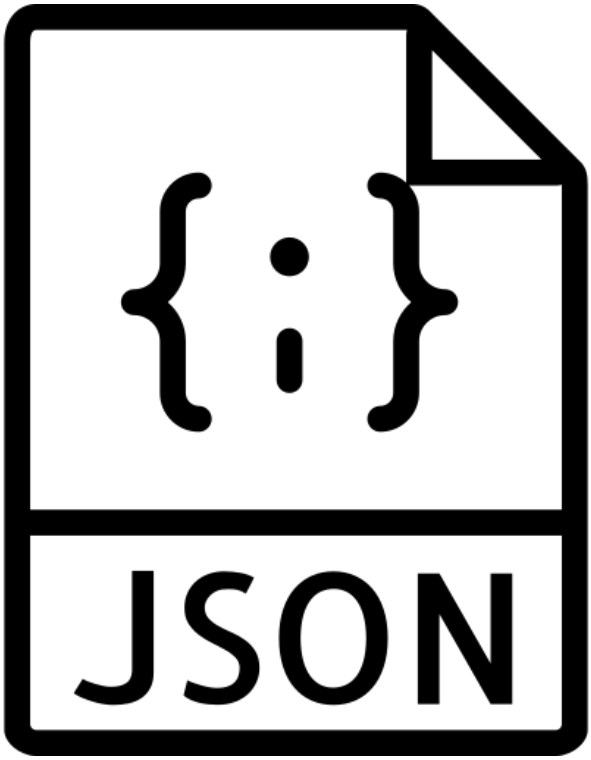



# JSON encoding/decoding

Project Proposal for Google Summer of Code 2020

—

Abdallah Khaled Ahmed Mahmoud Elshamy
Third year student at Computer and Systems department
Faculty of engineering - Ain Shams University
Cairo - Egypt

# Table of Contents



# Introduction

- My name is **Abdallah Khaled Elshamy**, a student from Egypt. I am a hard-working person who is passionate about his work. I am a third year undergraduate student studying computer and systems engineering at faculty of engineering - Ain Shams University.
- I speak Arabic (native) and English (good written communication skills).
- I will dedicate the whole summer to the project. I will treat it as a full-time job (40 hours/week or more if necessary).
- I will be available on the IRC channel as long as I am online. I check my email periodically.
- I may take a one week summer vacation during July and may not be able to work for the estimated period through this week (I will inform my mentor as fast as I can.)

## Contact Information


- E-mail: **abdallah.k.elshamy@gmail.com**
- User name on Savannah: **Abdallah_Elshamy**
- Nickname on IRC: **Abdallah_Elshamy**
- Location**: Cairo,Egypt**
- Time zone: **(UTC+2)**
- Public profile: **User:Abdallah Elshamy - Octave**
- Github: **Abdallah-Elshamy**
- Phone number: **+201272295337**


## Self-assessment

- I certainly love to receive advice from more experienced developers. Listening to them and asking questions is definitely helpful for me to become a better developer. I think the mentorship in GSoC is one of the coolest things about it. I also like to give advice and to help other people but only when I think that I have the sufficient amount of knowledge to benefit them.
- Criticism is greatly important and highly appreciated by me as it allows me to learn from other's experiences. To be useful for me I prefer it to:
    - Be as specific as possible.
    - Be supported with examples.
    - Use clear language and avoid ambiguity.
    - Be given in points so I can track my improvement in a better way.

## Motivation

JavaScript Object Notation, in short JSON, is a very common human readable and structured data format. Unfortunately, GNU Octave still lacks builtin support for that data format. Having JSON support, Octave can improve for example it's web service functions, which often exchange JSON data these days.

**Related bugs:** bug #53100

## Task

evaluate (and cherry pick from) different implementations (octave-jsonstuff , jsonlab , JSONio and octave-rapidjson) to create MATLAB-compatible jsonencode and jsondecode functions. This involves proper documentation of the work and unit tests to ensure the correctness of the implementation.

## Deliverables

The project will deliver MATLAB-compatible jsonencode and jsondecode functions. It will also produce a test suite to verify the two functions and a proper documentation to them.

## Libraries to be used

We will use RapidJSON for the following reasons:

- It has a better performance(according to this benchmark), code quality and documentation.
- It's a header only library which can be integrated easily in Octave.
- It does not depend on external libraries (even on STL.)
- It is Unicode-friendly.

## Plan

1. During the community bonding period, I will get more familiar with the organization and solve more bugs.
2. The first output of my work will be a complete test suite extracted from the four approaches and my additions to it. It will be MATLAB-compatible for benchmarking.
3. Assess comprehensively the libraries with tests and benchmarks. Create reliable figures and graphics to give good insights performance and MATLAB-compatible data processing.
4. Produce a C++ implementation for jsondecode/jsonencode
5. Convert the test suite to Octave BIST
6. Provide proper documentation for the functions.
7. Integrate the  functions into Octave core.
8. Use the community and mentors feedback after submissions to perfect the patch.

## Milestones

1. **26/6: Deliver test suite (first evaluation period starts on 29/6)**
2. **20/7: Deliver jsondecode (second evaluation period starts on 27/7)**
3. **05/8: Deliver jsonencode (final week starts on 24/8)**

## Timeline

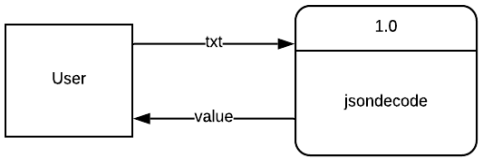
| From-To | Duration | Task | Hours/ Week |
|---|---|---|---|
| 04/5 - 01/6 (Community Bonding Period) | 27 days | getting more familiar with the organization and solving more bugs | 20-25 |
| 01/6 - 21/6* | 20 days | Preparing the test suite | 7-10 |
| 21/6 - 03/7 | 12 days | Finalizing the test suite, running tests on the libraries and Creating reliable figures. | 40-45 |
| 03/7 - 06/7 | 3 days | Analyzing results and taking design decisions with the mentors. | 40-45 |
| 06/7 - 18/7 | 12 days | Implementing jsondecode | 40-45 |
| 18/7 - 20/7 | 2 days | Buffering | 40-45 |
| 20/7 - 03/8 | 14 days | Implementing jsonencode | 40-45 |
| 03/8 - 07/8 | 4 days | Buffering & Documenting | 40-45 |
| 07/8 - 12/8 | 5 days | Converting the test suite to Octave BIST | 40-45 |
| 12/8 - 17/8 | 5 days | Cleaning the code and preparing the patch | 40-45 |
| 17/8 - 31/8 | 14 days | Perfecting the patch with the community feedback. | 40-45 |

* My final exams are expected to start from 1/6 to 21/6 (It may change. I will inform my mentor as soon as I know.)

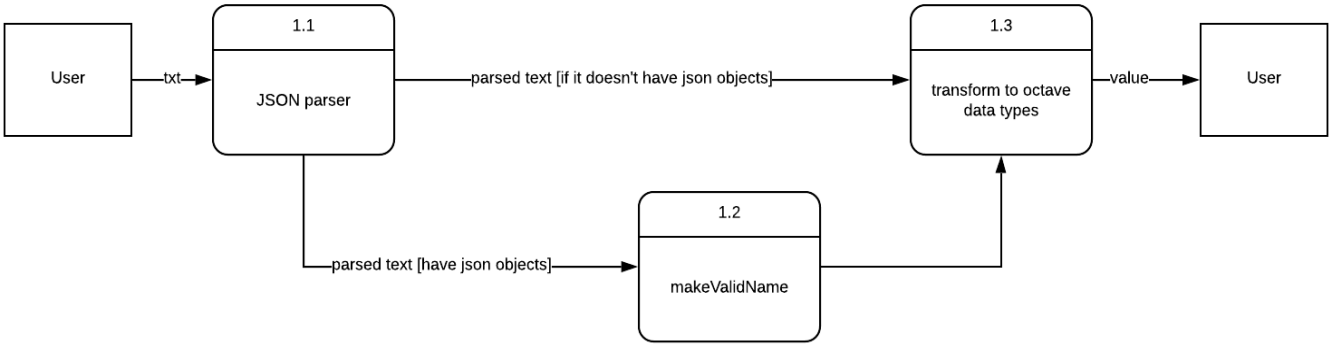
## Implementation Details

1. The test suite: The strategy we will follow in building the test suite will be:
    a. Extract the tests from the previous implementations.
        i. Extract tests from “test_jsonread.m” and “test_jsonwrite.m” from JSONio (needs to be MATLAB-compatible.)
        ii. Extract tests from the “examples” folder in jsonlab (also needs to be MATLAB-compatible.)
        iii. Extract tests from the “tests” folder and the BIST in “load_json.cc” and “save_json.cc” in octave-rapidjson (needs to be MATLAB-compatible.)
        iv. Extract the BIST from “inst/jsondecode.m” and “inst/jsondecode.m” in octave-jsonstuff (already MATLAB-compatible.)
        v. Add tests which cover the features from the official MATLAB documentation.
    b. Organize the tests and add missing cases (if any) to make sure that the suite covers all the data types, corner cases and compatibility considerations (for example: jsonencode have different behaviours in encoding NaN and inf. jsondecode may change the original string due to makeValidName function.)
    c. Ensure the suite has some big test cases to ensure the accuracy of measuring the performance.
    d. Make the test suite MATLAB-compatible.
    e. To be able to benchmark the previous implementations with the same test suite we will write a wrapper around non-MATLAB-compatible implementations to follow MATLAB interfaces (jsonencode, jsondecode).
    f. In the end, convert the test suite to Octave BIST.
2. Jsondecode: What this function does is illustrated in the following data flow diagram

Context diagram




Data flow diagram



The choice for the parser is already decided. What remains is to determine the efficient algorithm for the transform part and we will do this based on the results we get from running our test suite on the implementations: octave-rapidjson (after wrapping it to be MATLAB-compatible) and octave-jsonstuff (as both implementations use RapidJSON). We will pick the best performing and the most MATLAB-compatible and make corrections if it fails some tests. If octave-jsonstuff performs better we will:

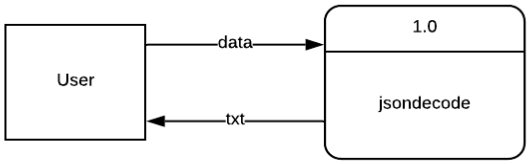
- Move some parts ("condensation" algorithm) to C++.
- Tidy the code, add documentation and add BIST.

If octave-rapidjson performs better we will:

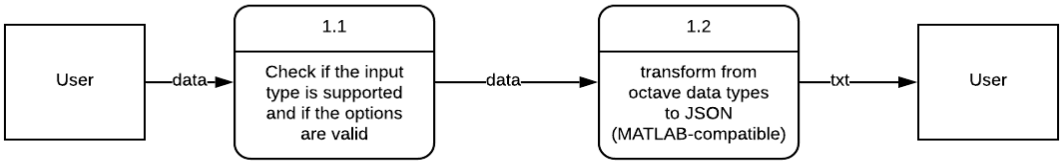
- Instead of using wrappers we will refactor the interface of the function itself to be MATLAB-compatible.
- Tidy the code, add documentation and add BIST.

3. Jsonencode: What this function does is illustrated in the following data flow diagram

Context diagram




Data flow diagram




There are two approaches to implement the transform part:

1. The approach followed by octave-rapidjson was to use PrettyWriter class supported by RapidJSON (This is the only implementation that is using C++.)
2. The approach followed by octave-jsonstuff , jsonlab and JSONio was to use a set of functions that convert to the correct JSON format (a function that converts string to JSON , a function that converts struct to JSON ...etc) but all of them are written in m-scripts.

What we will do will be the following:

- To be able to assess both approaches, we will write the second approach (which is written in m-script) in C++. The set of functions will support all the data types that jsonencode supports. An example of a function in the set of functions that encodes numeric values is :

```
template <typename T>
static std::string
jsonencode_numeric (T x)
{
  if (! isscalar (x))
    error ("jsonencode: Internal error: input must be scalar");
  if (isinteger (x))
    return sprintf ("%d", x);
  else
    {
      if (iscomplex (x))
        error ("jsonencode: Complex numbers are not supported");
      else if (isnan (x) || isinf (x))
        {
          if (ConvertInfAndNaN)
            return "null";
          else
            {
              if (isinf (x) && x < 0)
                return "-Infinity";
              else if (isinf (x) && x > 0)
                return "Infinity";
              else
                return "NaN";
            }
        }
      else if (isintval (x))
        return sprintf ("%d", x);
      else
        // This is the most concise way I could figure out for
        // printing floating point numbers without roundoff.
        return sprintf ("%.17e", x);
    }
}
```

We will also check if the input type has a jsonencode method to provide compatibility for future data types.

- Write a wrapper around octave-rapidjson to be MATLAB-compatible.

- Use the test bench to determine the best performing and the most MATLAB-compatible and make corrections if it fails some tests.
- If the best is octave-rapidjson we will refactor the interface of the function itself to be MATLAB-compatible and get rid of the wrapper.
- Tidy the code, add documentation and add BIST.

## Contribution to Octave

Bug: #57041

I have submitted a patch to add the new functions "startsWith" and "endsWith" to Octave.

## Qualification

- I really want to contribute to Octave and to engage in the community.
- I have a good knowledge of algorithmic analysis (I have successfully completed Algorithms specialization on Coursera and participated in local and national competitive programming contests.)
- I have a good knowledge of C++ and used it in many projects.
- I have been using Octave/MATLAB for about two years. I wrote many m-scripts.
- I have a good knowledge of JSON format.
- I have a good knowledge of Python and a fair knowledge of Java.
- I am comfortable with using version control systems (both Git and Mercurial.)
- I am comfortable with the make tool and I successfully built Octave.
- I am familiar with IRC and the mailing list.
- I always want to expand my knowledge and I am always ready to learn more.