**[Computer science course notes](http://www.microveggies.com/csci)**

**Home**

[**Lecture19 - Introduction to JSON**](http://www.microveggies.com/csci/index.php/csci-2270-lecture-notes/2-lecture19-introduction-to-json)

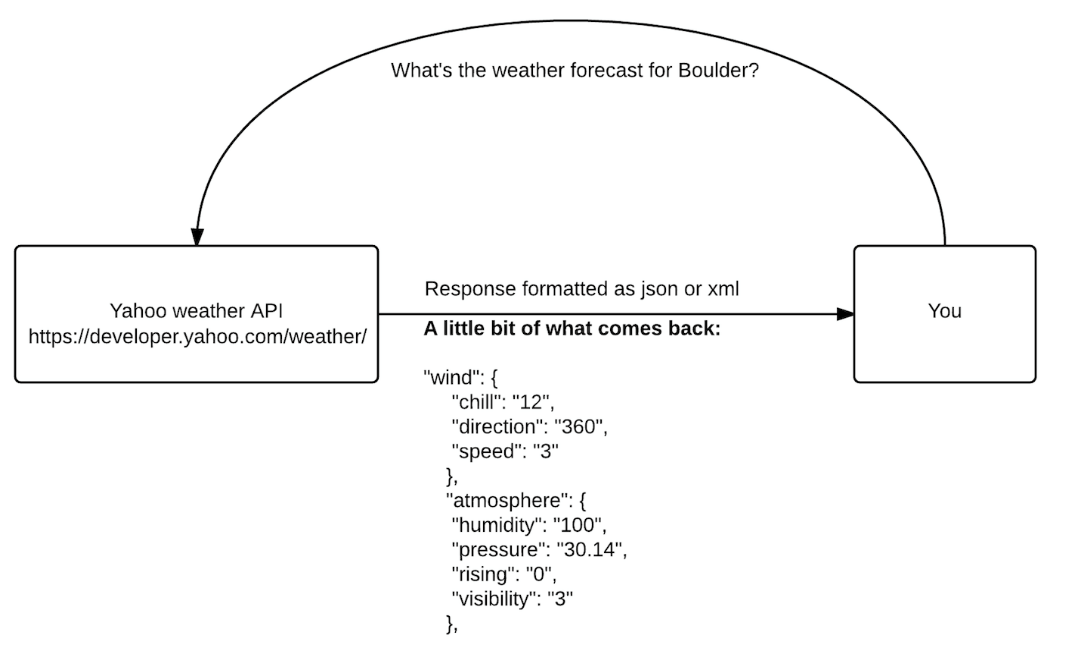
**Details**

Written by Rhonda Hoenigman

 Published: 14 February 2015

 Hits: 2409

JSON (JavaScript Object Notation) is a language-independent data exchange format. If you download data from the Internet, it is often formatted using JSON, and can therefore be read and interpreted using standardized commands for reading JSON files. For example, Yahoo weather has a programming interface that you can use to download information from the Yahoo weather website. When your program queries the Yahoo weather website, data is returned as either json or xml.



Formatting the data in a given way means that the output is standardized and can then be easily processed by the calling program. This approach solves problems with output that we've seen with submitting to COG. The receiver (COG) doesn't know that you didn't mean to add an extra space to your output, and since there is no intrinsic meaning assigned to the output, it's interpreted literally and any output that doesn't exactly match an expected output is just incorrect.

**Data formatted using Key:Value pairs**

The fundamental concept with JSON is that data is packaged using key:value pairs, where the key is a string identifier and the value is the data associated with that key. For example, using the key:value representation, you could represent the linked list of cities you built in Assignment 4 as:

{

"cities": ["Los Angeles", "Phoenix", "Denver", "Dallas", "Atlanta", "Washington, D.C.", "New York", "Boston"]

}

where cities is the key, and the value is the array of cities. Using key:value pairs that can built up from other key:value pairs, we can assign meaning to program outputs in a standardized format that can then be read and interpreted at a distant location.

**JSON Libraries for C++**

Due to the usefulness of JSON, there are several libraries available for parsing and writing JSON files in almost all languages. This greatly simplifies life for any programmer working with JSON because it means you don't have to write a JSON parser yourself, you just have to download an existing library and incorporate it into your code. In this class, we're going to use JSON-C, which is a JSON parser for C/C++. It's also available in the Ubuntu package manager, making it easy to install.

**Installing JSON-C**

To install JSON-C on your VM, open a Terminal window and type:

sudo apt-get install libjson0 libjson0-dev

Using JSON-C in your program

To use the JSON-C library in your program, you need to include it in your source file and also link the library during compilation. To include the JSON-C library, add

#include <json/json.h>

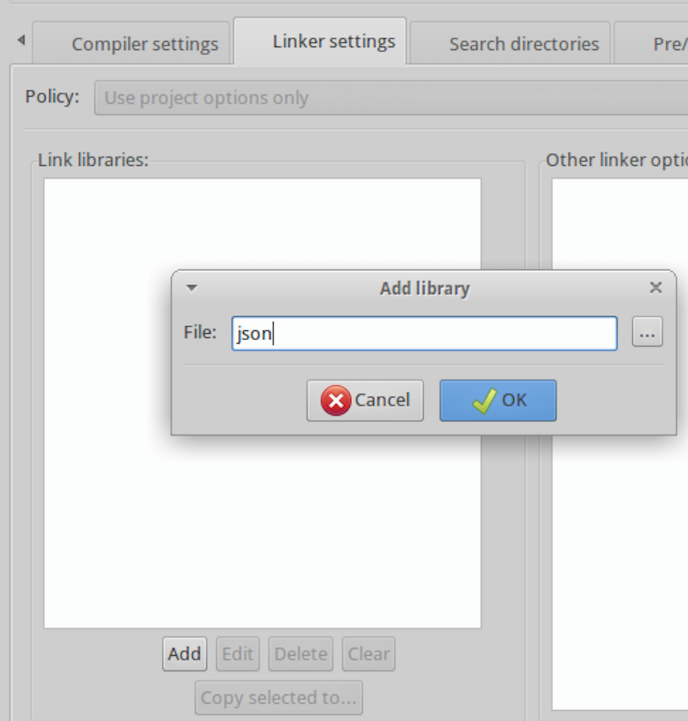
to your source file where you will be using the JSON-C library commands. To link the library into your program, follow the instructions for CodeBlocks or Command-line, depending on how you are compiling your code.

**CodeBlocks Instructions.**

You need to update the build settings to include json.

Select Project -> Build Options -> Linker Settings

In Linker Settings, there is a window called Link libraries. Click the Add button below this window and in the Add library window, type json and click OK.



**Command-line Instructions**

If you are building your code from the command line, you need to manually link in the json library. Add a -l json flag to your build statement. For example, to build Lecture17\_JSON.cpp and link in json, type

g++ Lecture19\_JSON.cpp -l json -o Lecture19\_JSON

**Creating a JSON output**

Formatting output using json starts with creating a json object:

json\_object \*jObj = json\_object\_new\_object();

creates an empty json\_object variable that we can then add key:value pairs to.

**Example 1 Integers:**

Count the number of items in the array A and format the output using json.

int A[10] = {23, 24, 25, 45, 47, 13, 23, 90, 80, 100};

int counter = 0;

for(int i = 0; i < 10; i++){

  if(a[i]> 40){

    counter++;

  }

}

The desired output value is an integer with a key called "counter". We create a new json integer object from the int counter variable, and then add the json integer object to the empty json object.

json\_object \*jCounter = json\_object\_new\_int(counter);

json\_object\_object\_add(jObj, "counter", jCounter);

The json\_object\_new\_int method takes an integer argument and creates a json\_object from the integer. The json\_object\_object\_add takes three arguments: the json\_object that the key:value pair is added to, and the key and value inputs. The key is a string, in this case, "counter", and jCounter is the json integer object.

You can output the jObj by converting it to a string using the json\_object\_to\_json\_string\_ext method:

cout<<json\_object\_to\_json\_string\_ext(jObj, JSON\_C\_TO\_STRING\_PRETTY)<<endl;

The json\_c\_to\_string\_pretty is a built-in flag that outputs the string in a easily readable format.

**Example 2 Strings:**

Return the second item in the array and format the output using json. In this example, the array is an array of strings, so the json object we use will need to handle a string.

string items[4] = {"computer", "jacket", "shoes", "puppy"};

string secondItem = items[1];

Use the json\_object new\_string method to create a json string object from the secondItem variable. The method expects a char\*. The string variable can be cast as a char\* using the c\_str() method on the string.

json\_object \*jItem = json\_object\_new\_string(secondItem.c\_str());

Use the existing json\_object jObj as the container for this key:value pair. Call the key "secondItem" and give it the value of jItem. Display the new json object that contains both the counter and the secondItem.

json\_object\_object\_add(jObj, "secondItem", jItem);

cout<<json\_object\_to\_json\_string\_ext(jObj, JSON\_C\_TO\_STRING\_PRETTY)<<endl;

**Example 3 Arrays:**

Generate an array of city names and output the contents of the array using json. The output should have the key "cities".

string names[6] = {"Los Angeles", "Phoenix", "Tucson", "Denver", "Atlanta", "Boston"};

With a json array object, there are two steps to generating the output. We create an object for the individual array items. For example, if it's an array of strings we need to create a json object for a string. Then, we create the json array type and add the json strings to it.

json\_object \*jArray = json\_object\_new\_array();

for(int i = 0; i < 6; i++{

    /\*Create the json string using the city name string.\*/

    json\_object \*city = json\_object\_new\_string(names[i].c\_str());

    /\*Add the string to the json array to build the array.\*/

    json\_object\_array\_add(jArray, city);

}

/\*Add the array to the object the same method we used for adding the

string and the integer. The key for this one is "cities".\*/

json\_object\_object\_add(jObj,"cities", jArray);

cout<<json\_object\_to\_json\_string(jObj)<<endl;

**Example 4 Objects:**

Objects can also be added to other objects. We can create an object that contains booleans, strings, integers, and arrays, and then add that object to another object that also contains booleans, strings, integers, and arrays. This example creates an object called jObj2 and then adds a string and an integer to it. The object is then assigned the key "Assignment6", and then added to jObj, which we created in the above code.

json\_object \*jObj2 = json\_object\_new\_object();

json\_object \*jStr = json\_object\_new\_string("this is a test");

json\_object\_object\_add(jObj2, "string", jStr);

json\_object \*jInt2 = json\_object\_new\_int(45);

json\_object\_object\_add(jObj2, "int example", jInt2);

json\_object\_object\_add(jObj, "Assignment6", jObj2); //add jobj to jobj2

cout<<json\_object\_to\_json\_string\_ext(jObj, JSON\_C\_TO\_STRING\_PRETTY)<<endl;

 JSON Output:

The output that you generate should look something like this:

{

  "counter":5,

  "item":"jacket",

  "cities":[

    "Los Angeles",

    "Phoenix",

    "Tucson",

    "Denver",

    "Atlanta",

    "Boston"

  ],

  "Assignment6":{

    "string":"this is a test",

    "int example":45

  }

}

The outer {} contains the jObj json\_object. The "cities" array is contained in the [ ] bracks. The "Assignment6" object also has it's own set of { }, and within it are the components of that object.

More information about json-c can be found here:

<https://linuxprograms.wordpress.com/2010/05/20/json-c-libjson-tutorial/>

[Back to Top](http://www.microveggies.com/csci/index.php/8-csci-2270-lecture-notes/2-lecture19-introduction-to-json#top)

© 2015 Computer science course notes