A JSON to CMake tool

Anthony J. Zukaitis, David A. Dixon, and Austin P. McCartney
XCP-3 Monte Carlo Codes and Applications
Los Alamos National Laboratory

January 22, 2019

1 Introduction

As part of the conversion of the MCNP®testing system to CTest[1], a tool was needed to generate CMake variables from JSON formatted data structure. The system was first implemented using the JSON to CMake parser [3] which with some slight modifications provided data useful for the MCNP CTest testing system. Upon further evaluation, we found it to be slow when setting up the 1600 tests during the CMake configuration step. Not only was it slow, it did not validate the JSON files during configuration allowing for unexpected behavior.

Upon some web searching we came across a simple recursive python example that would provide the same capability needed by our testing system[4]. This python example was effectively wrapped into a single python script that was used to convert the JSON data into CMake variables.

This tool is attached as json_to_cmake.py and is listed in the appendix. In essence, the tool provides single variables for each value defined within the JSON file. The name for the variable is constructed by using the associated object names and array indicies needed to deference the variable itself. In the following sections, examples are provided to better describe how this is done.

2 Simple Examples

For example, take the following JSON object containing a single name/value pair:

```
{ mykey : myvalue }
```

Using the name json as the name of the JSON object, would define the following CMake variables

```
set( json mykey )
set( json.myvalue )
```

the first variable json being a CMake list of values defined at the current object/array level of the variable in question. Next we will define an array example:

```
{ myarray : [ val1, val2, val3 ] }
```

would define the following CMake variables

```
set( json myarray )
set( json.myarray 0 1 2 )
set( json.myarray.0 val1 )
set( json.myarray.1 val2 )
set( json.myarray.2 val3 )
```

3 Complex Example

Using an example file:

```
{
  "1darray" : [ 1,2,3,4,5 ],
  "2darray" : [ [11,12,13],
                [21,22,23],
                [31,32,33] ],
  "arrayofobjects" : [ { "key1" : "val1" },{ "key2" : "val2" } ] ,
  "objectsofarray" : { "keyarr1" : [1,2,3] , "keyarr2" : [4,5,6] },
  "objectof2darray" : { "keyarr2d" : [ [11,12],[21,22] ]}
}
```

Produces the following CMake variables:

```
set( json.arrayofobjects.1 key2 )
set( json.2darray.1 0 1 2 )
set( json.2darray.0 0 1 2 )
set( json.arrayofobjects 0 1 )
set( json.2darray.2 0 1 2 )
set( json.objectof2darray.keyarr2d.1.1 22 )
set( json.2darray.0.1 12 )
set( json.2darray.0.0 11 )
set( json.objectof2darray.keyarr2d.1.0 21 )
set( json.2darray.1.0 21 )
set( json.2darray.1.1 22 )
set( json.2darray.1.2 23 )
set( json.arrayofobjects.0 key1 )
set( json.objectsofarray.keyarr1 0 1 2 )
set( json.objectsofarray.keyarr2 0 1 2 )
set( json.1darray 0 1 2 3 4 )
set( json.arrayofobjects.1.key2 val2 )
set( json.objectsofarray.keyarr2.2 6 )
set( json.objectsofarray.keyarr2.1 5 )
set( json.objectsofarray.keyarr2.0 4 )
set( json arrayofobjects 2darray objectof2darray objectsofarray 1darray )
set( json.2darray 0 1 2 )
set( json.2darray.2.2 33 )
set( json.2darray.2.1 32 )
set( json.2darray.2.0 31 )
set( json.objectof2darray keyarr2d )
set( json.1darray.2 3 )
set( json.1darray.3 4 )
set( json.objectof2darray.keyarr2d 0 1 )
set( json.1darray.1 2 )
set( json.1darray.4 5 )
set( json.objectsofarray.keyarr1.0 1 )
set( json.objectsofarray.keyarr1.1 2 )
set( json.objectsofarray.keyarr1.2 3 )
set( json.objectsofarray keyarr1 keyarr2 )
set( json.objectof2darray.keyarr2d.0.0 11 )
set( json.objectof2darray.keyarr2d.0.1 12 )
set( json.objectof2darray.keyarr2d.0 0 1 )
set( json.objectof2darray.keyarr2d.1 0 1 )
```

```
set( json.arrayofobjects.0.key1 val1 )
set( json.1darray.0 1 )
set( json.2darray.0.2 13 )
```

4 Usage

Usage of the tool is:

```
usage: json_to_cmake.py [-h] [--output OUTPUT] [--name NAME] [--delim DELIM]
                        jsonfile

Flatten a json file into single name value pairs for use with CMake.

positional arguments:
  jsonfile Name of json file to flatten

optional arguments:
  -h, --help show this help message and exit
  --output OUTPUT Name of output file to write to
  --name NAME Namespace to use for json variables (default: json)
  --delim DELIM Delimiter to use for namespace separation (default: .)
```

A Appendix

```
#!/usr/bin/env python
from __future__ import print_function
import json
import pprint
import argparse

def flatten_json(y,namespace="json",delim="."):
        out={}
        def flatten(x, name='',delim=delim):
                if isinstance(x,dict):
                        for a in x:
                                flatten(x[a], name + a + delim)
                        out[name[:-len(delim)]]=list(x.keys())

                elif isinstance(x,list):
                        i=0
                        for a in x:
                                flatten(a, name + str(i) + delim)
                                i+=1
                        out[name[:-len(delim)]]=list(range(0,i))
                else:
                        out[name[:-len(delim)]] = x

        flatten(y,name=namespace+delim)
        return out

parser = argparse.ArgumentParser(\
        description="Flatten a json file into single name value pairs for use with CMake.")
parser.add_argument("jsonfile",\
        help="Name of json file to flatten",type=str)
parser.add_argument("--output",\
        help="Name of output file to write to",type=str)
parser.add_argument("--name",\
        help="Namespace to use for json variables (default: %(default)s)",type=str,default="json")
parser.add_argument("--delim",\
        help="Delimiter to use for namespace separation (default: %(default)s)",type=str,default=".")
args=parser.parse_args()

with open(args.jsonfile,'r') as json_file:
        json_obj = json.load(json_file)
        flat=flatten_json(json_obj,namespace=args.name,delim=args.delim)
        if( args.output ) :
                f1=open(args.output,"w")
        for key in flat:
                if isinstance(flat[key],list):
                        output=" ".join(str(x) for x in flat[key])
                else:
                        output=str(flat[key])
                mystring="set( " + str(key) + " " + output +" )\n"
                if( args.output ) :
                        f1.write(mystring)
                print(mystring,end='')
        if( args.output ) :
                f1.close()
```

References

[1] Completion of CTest Support in the MCNP®Code Base. Los Alamos National Laboratory Memo XCP-3:16-030

[2] `http://json.org/`

[3] https://github.com/sbellus/json-cmake

[4] https://towardsdatascience.com/flattening-json-objects-in-python-f5343c794b10