|  |  |
| --- | --- |
| **GAT150 – Introduction to Game Programming** |  |

***Json***

***OVERVIEW***

In this assignment you will integrate a JSON file loader. The JSON is a human-readable text file format. The JSON file will be used to store scene, game object and component definitions. The level designer can modify the JSON file to add, remove and alter the objects in the game.

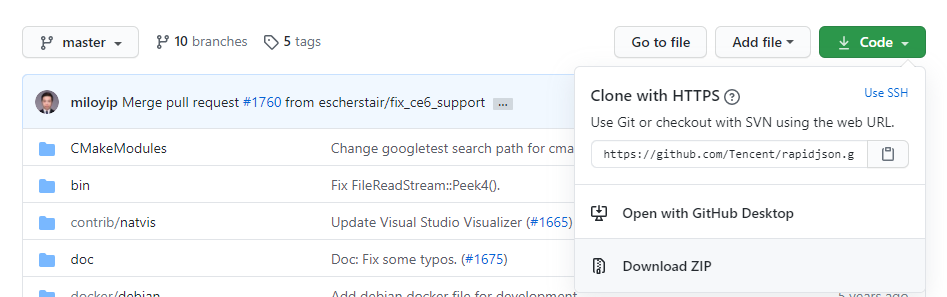
***GRADING***

This assignment is worth 50 points. To receive full credit, your engine must include the JSON file loader functions and read data from a JSON file.

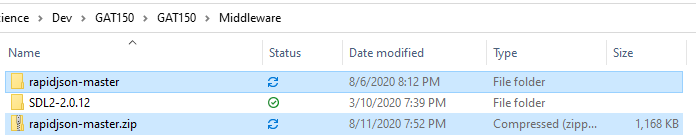
***INSTRUCTIONS***

# Install the RapidJSON code

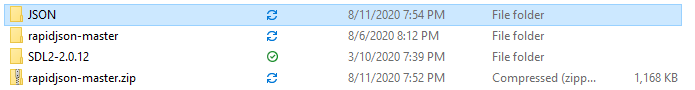
* Download the code for RapidJSON (<https://rapidjson.org/>)
  + Download the .zip from the RapidJSON GitHub
  + <https://github.com/Tencent/rapidjson/>



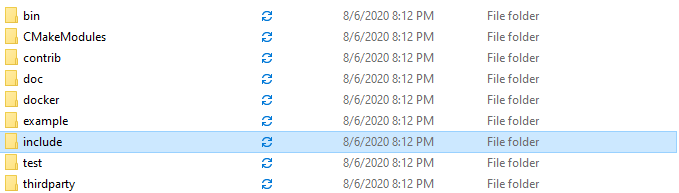
* Extract the .zip in the Middleware folder



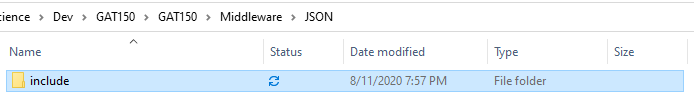
* Create a folder called JSON



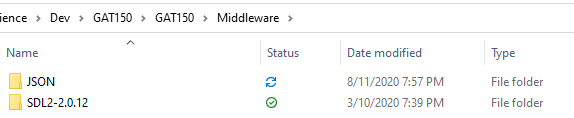
* Copy the *include* folder inside rapidjson-master



* Paste the include folder inside the newly created JSON folder
  + The *include* folder contains the only code we need

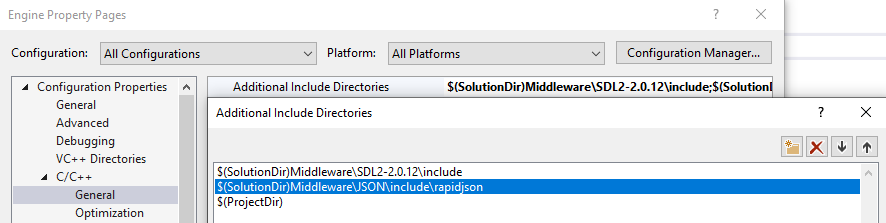


* Delete the .zip and rapidjson-master folder

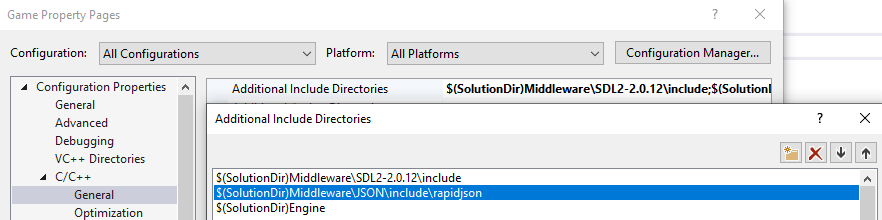


# Include the RapidJSON code in the project

* Add the JSON include directory to the *Engine* project *Additional Include Directories*
  + Make sure the *Configuration* is set to *All Configurations*
  + Make sure the *Platform* is set to *All Platforms*
  + $(SolutionDir)Middleware\JSON\include\rapidjson

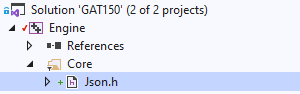


* Add the JSON include directory to the *Game* project *Additional Include Directories*
  + Make sure the *Configuration* is set to *All Configurations*
  + Make sure the *Platform* is set to *All Platforms*
  + $(SolutionDir)Middleware\JSON\include\rapidjson



# Create the JSON loader and reader functions

* Create a Json.h file in Core
  + Make sure it exists in the Core folder in the file explorer



* The JSON functions will not exist in a class and will exist as functions
* You can place the functions in a nested namespace
  + In this example it is in nc->json, the function call would be nc::json::*function*()

#include "document.h"

#include <string>

namespace nc

{

namespace json

{

bool Load(const std::string& filename, rapidjson::Document& document);

bool Get(const rapidjson::Value& value, const std::string& name, int& data);

}

}

* Create the function definitions (ctrl + .)
* Move the newly created Json.cpp to the Core folder
* Here are empty functions to show the namespaces
  + If preferred, the functions can be left as created nc::json::Load(…)

#include "pch.h"

#include "Json.h"

namespace nc

{

namespace json

{

bool Load(const std::string& filename, rapidjson::Document& document)

{

return false;

}

bool Get(const rapidjson::Value& value, const std::string& name, int& data)

{

return false;

}

}

}

**bool Load(const std::string& filename, rapidjson::Document& document)**

bool success = false;

std::ifstream stream(filename);

if (stream.is\_open())

{

rapidjson::IStreamWrapper istream(stream);

document.ParseStream(istream);

success = document.IsObject();

}

return success;

**bool Get(const rapidjson::Value& value, const std::string& name, int& data)**

// check if 'name' member exists

auto iter = value.FindMember(name.c\_str());

if (iter == value.MemberEnd())

{

return false;

}

// check if type is desired type

auto& property = iter->value;

if (property.IsInt() == false)

{

return false;

}

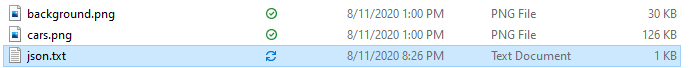
// set data

data = property.GetInt();

return true;

# Create a JSON file

* Create a text file called json.txt



* Add the text to read in the json.txt

{

"integer1": 123,

"integer2": 456

}

It is common to have the JSON file not in the correct format. Use this page to verify your JSON file:

[https://jsonformatter.curiousconcept.com/#](https://jsonformatter.curiousconcept.com/)

* Copy the contents of the JSON file and paste it into the page
* Click *Process*
* It will notify you of any errors if they exist

# Load and Read the JSON file

Main.cpp

#include "Core/Json.h"

…

int main(int, char\*\*)

{

rapidjson::Document document;

nc::json::Load("json.txt", document);

int i1;

nc::json::Get(document, "integer1", i1);

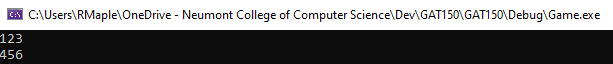
std::cout << i1 << std::endl;

int i2;

nc::json::Get(document, "integer2", i2);

std::cout << i2 << std::endl;

Run the program and the command prompt should show the values for integer1 and interger2



# Create functions for data types

* Add the following data types that can be read from a JSON file

bool Get(const rapidjson::Value& value, const std::string& name, float& data);

bool Get(const rapidjson::Value& value, const std::string& name, bool& data);

bool Get(const rapidjson::Value& value, const std::string& name, std::string& data);

bool Get(const rapidjson::Value& value, const std::string& name, Vector2& data);

bool Get(const rapidjson::Value& value, const std::string& name, Color& data);

* The float, bool and string types should be very similar to the int
* The Vector2 and Color need some extra code

Here is the Vector2 version, implement the Color version using this as a guide. Remember the Color has 4 components (r,g,b,a).

bool Get(const rapidjson::Value& value, const std::string& name, Vector2& data)

{

auto iter = value.FindMember(name.c\_str());

if (iter == value.MemberEnd())

{

return false;

}

**auto& property = iter->value;**

**if (property.IsArray() == false || property.Size() != 2)**

**{**

**return false;**

**}**

**for (rapidjson::SizeType i = 0; i < 2; i++)**

**{**

**if (property[i].IsNumber() == false)**

**{**

**return false;**

**}**

**}**

**data.x = property[0].GetFloat();**

**data.y = property[1].GetFloat();**

return true;

}

# Load all the supported data types

* Paste the following into the json.txt, this will replace what is in there

{

"string":"hello world",

"boolean":true,

"integer1":123,

"integer2":456,

"float":3.1416,

"vector2":[

100.0,

200.0

],

"color":[

1,

0,

1,

1

]

}

* Have all the data members read in main

std::string str;

nc::json::Get(document, "string", str);

std::cout << str << std::endl;

bool b;

nc::json::Get(document, "bool", b);

std::cout << b << std::endl;

int i1;

nc::json::Get(document, "integer1", i1);

std::cout << i1 << std::endl;

int i2;

nc::json::Get(document, "integer2", i2);

std::cout << i2 << std::endl;

float f;

nc::json::Get(document, "float", f);

std::cout << f << std::endl;

nc::Vector2 v2;

nc::json::Get(document, "vector2", v2);

std::cout << v2 << std::endl;

nc::Color color;

nc::json::Get(document, "color", color);

std::cout << color << std::endl;

The output stream operator overloads will need to be created for Vector2 and Color:

Example of the Vector2 definition:

std::ostream& operator<<(std::ostream& stream, Vector2& v)

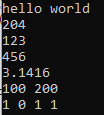
{

stream << v.x << " " << v.y;

return stream;

}

Submit a screenshot of the console output for submission:



***RESOURCES***

<https://rapidjson.org/>

<https://jsonformatter.curiousconcept.com/>