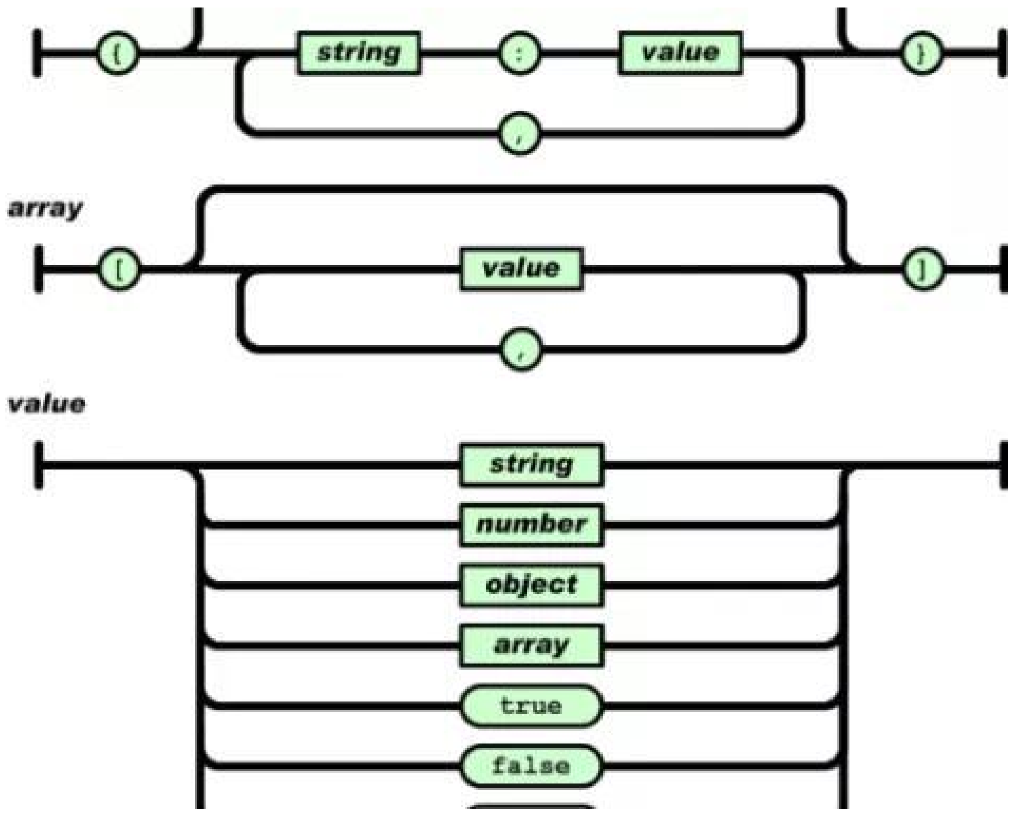
Common ESP8266 tools

ESP8266: The JSON Streaming Parser

by squix78 on 27. January 2017 0 comments



You might not know it but the most important puzzle piece for all my recent ESP8266 projects is a thing called a streaming parser. Keep reading if you want to know what that is, how it works and why it is so important for my ESP8266 projects.

Maybe we have to start even before: what is a parser? You are most certainly using parsers every day. A parser is a piece of code that analyses a document by reading in its content. For that it usually has some knowledge about the structure of the document, sometimes called a schema or syntax. The web browser you are using to read this post uses an HTML parser to understand the tags that are downloaded from my webserver and then put into a visualization with formatted text, pictures and links.

So now that we roughly understand what a parser is the next question would be what is a streaming parser? With a modern smart phone or desktop computer we often don't need streaming parsers anymore, we use document object model parsers (DOM) parsers instead. A DOM parser creates a tree like structure of the document it parses, keeps this structure in memory and makes it available for the code that does something meaningful with it. DOM parsers are very easy to use, fast and convenient. But this convenience comes at the price of memory requirements. The DOM parser needs a lot of memory, since it keeps the whole document in memory until it is advised to get rid of it. If you have a lot of RAM and not so big documents this is perfectly fine. But if the documents are very big compared to the available (heap) memory you might run into a serious problem.

Imagine the parser to be something like a water meter. A water meter which works like a DOM parser needs a bucket and measures the amount of water by filling the bucket and then measuring the weight of the water in the bucket. If there comes a lot of water then the bucket must be big. A water meter which works like a streaming parser measures the water while it flows through and doesn't care what happens to the water afterwards. The bucket in this analogy is the heap or working memory of your micro controller, the water is the stream of bits and bytes that you receive, either from the file system or from a remote server. And the parser does not just measure the amount of bits and bytes but also tries to understand the content. The streaming parser doesn't care how big the document (or the amount of water) is, it just uses from the stream the information it is interested in. Streaming parsers are also referred to as event based parsers since the react to certain events in the data stream. DOM parser are referred to as a tree based parser since they build a full representation of the document in the tree-like structure. In an HTML tree the <html> element would be the root of the tree…

The JSON Streaming Parser Library

"Ok, I got it, but why is this important for the ESP8266?" you might ask. Embedded devices usually have very limited resources available. One scarce resource is the heap memory. Many of the REST APIs I am consuming in my projects have very big documents of information, but we are usually just interested in a small fraction of it. As mentioned earlier, a tree based parser would load the whole document into memory and make it available once the document stream has ended. And this would just crash the ESP8266 pretty quickly.

This made me port a PHP JSON parser over to C++ and make it available as a library, mostly to be used in my own projects. Let's have a look at the header file of the JsonListener:

```
class JsonListener {
  public:
    virtual void whitespace(char c) = 0;
    virtual void startDocument() = 0;
    virtual void key(String key) = 0;
    virtual void value(String value) = 0;
    virtual void endArray() = 0;
    virtual void endObject() = 0;
    virtual void endDocument() = 0;
    virtual void startArray() = 0;
    virtual void startObject() = 0;
};
```

JsonListener.h hosted with ❤ by **GitHub** **view raw**

The methods here are callback methods which will get invoked if the respective event happens while parsing the document. Let's start with an example. For the JSON object {"name": "Eichhorn"} we get the following invocations:

1. startDocument()
2. startObject()
3. key("name")
4. value("Eichhorn")
5. endObject()
6. endDocument()

I often just implement (AKA "write code") for *key* and the *value* method. In the key method I take the store the value of the key parameter. Then later in the value method I check what the last key was I had seen and then I store the value in the appropriate variable. For the example from before I would do

```
[..]
void ExampleListener::key(String key) {
  currentKey_ = key;
}

void ExampleListener::value(String key) {
  if (currentKey_ == "name") {
    name_ = value;
  } else if (currentKey_ == "city") {
    city_ = value;
  }
}
[..]
```

ExampleListener.cpp hosted with ❤ by **GitHub** **view raw**

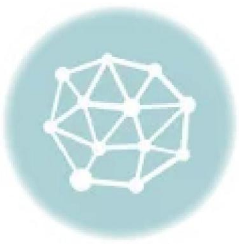
In the stream of the object {“name”: “Eichhorn”} we will first get a call to the method key with the value “name” which we store in currentKey_. Next the parser will detect a value and call our value method with the value “Eichhorn”. The parser can now make the connection (or create a context) that after the key name the value Eichhorn should be stored in the member variable name_.

Conclusion

For a document or object of the size we had in the example a streaming parser is usually an extreme overkill. It is complicated to use, requires you to write a lot of code and is memory wise probably even worse than a tree parser. It is only recommended to implement a streaming parser if you have big objects or if you just don’t know how big your object might be. In those cases a streaming parser will be a good friend, since it only requires memory for the objects you actually want to use from the whole big object. You can find my library here: https://github.com/squix78/json-streaming-parser

Do you like this post? A typical project like the ESP8266 Weather Station takes many hours of my free time to implement and even more to maintain. Maybe you feel like offering me a 🍻beer for it? Modern technology allows teleportation of beers around the world;-) It is easy and painless and much appreciated.

Related Posts:


ESP8266: Offline Debugging with

ESP32 unboxing!

ESP8266 Power Supply: Running

WeatherStation Kit ready for

Edit

Post navigation

ESP8266 Weather Station Projects

Leave a Reply

Enter your comment here...

SEARCH

Search

Search ... Search

CART

× **WEATHERSTATION KIT W/ WHITE OLED**
1 × $19.90

Subtotal: $19.90

View Cart Checkout

ESP8266 WEATHERSTATION

ESP8266 IoT Starter Kit with...

$21.90 Prime

Shop now

ESP8266 Weather...

$3.99

Shop now