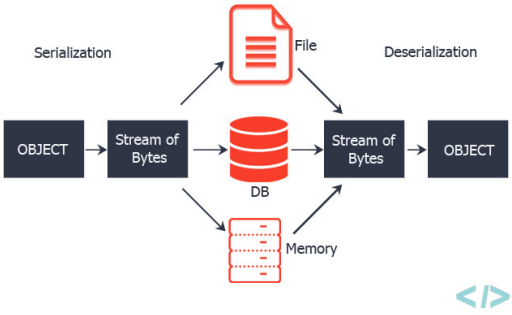
Lab 1: Introduction to Serialization

Neil Johari <njohari@umich.edu>

May 27, 2019

1 Introduction



pc: codenuclear.com

Serialization (also known as marshalling) is a technique used to share the state of data in memory.

During serialization, data structures and objects are converted into a stream of bytes, which then can be stored on a disk or shared over a network. Serialization becomes useful when data must be reconstructed via deserialization (or unmarshalling): the serialized stream is used to recreate the original object in memory, and the new object is semantically equivalent to when it was serialized.

In this lab we will explore a brief history of various serialization techniques and understand how computer files work. Then we will investigate how we can implement these techniques into microcontroller driven systems. In the following labs, we will actually implement two examples of serialization in ArduinoC and corresponding deserialization.

1.1 What's the point?

We mentioned that serialization is used to share the state of data in memory—but what does this really mean? When would you use it?

In a computer program, an object's lifespan is relatively short. The object may be removed by a garbage collector when it is no longer needed, and will most certaintly be destroyed when the program terminates.

Sometimes, we would like to "save" this object—or at least everything about it—and be able to "revive" it at a later time (perhaps even on another program and computer). An example of when we'd like to do this is communicating efficiently between different microcontrollers, or saving program data to an SD card to analyze it more effectively later on.

Serialization will help us accomplish these goals!

2 History

In the 1990s, Extensible Markup Language (XML) was developed by the W3C as a file format intended to structure data documents such that they were both easily readable by humans and also parsable in computer programs. Please note that XML is not the same thing as HTML; HTML is used to describe how a page should be presented in browsers, while XML describes content.

Later, Javascript Object Notation (JSON) was developed as a light-weight alternative to XML. JSON is often used to send data back and forth between web clients and servers.

YAML is another alternative to XML which has become popular for configuration files. Since YAML 1.2, YAML is fully compatible with JSON as an official subset. The two appear very distinct, but happen to be able to encode similar data [1].

In 2001, Google began developing Protocol Buffers (Protobuf). Google describes Protobuf as a "language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler" [2]. Protobuf is the best of both the human readable and machine parsable worlds. Protobuf "messages" are defined in the proto language and are easy to read; data serialized using these messages are written extremely efficiently in binary. We will investigate Protobuf in more depth in a later lab, so don't worry if the concepts don't seem to click right now.

It is worth noting that many popular languages such as C, C++, Java and Ruby provide methods to serialize and deserialize objects into binary streams.

Listing 1: Sample XML document [3]

```
<food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>
           Two of our famous Belgian Waffles with plenty of real
               maple syrup
    </description>
    <calories>650</calories>
</food>
```

Listing 2: Sample YAML document [4]

```
json:
  - rigid
  - better for data interchange
yaml:
  - slim and flexible
  - better for configuration
object:
    key: value
  array:
    - null_value:
    - boolean: true
    - integer: 1
```

Listing 3: Sample JSON document [4]

```
{
    "json": [
        "rigid",
        "better for data interchange"
    ],
    "yaml": [
        "slim and flexible",
        "better for configuration"
    ],
    "object": {
        "key": "value",
```

```
        "array": [
            {
                "null_value": null
            },
            {
                "boolean": true
            },
            {
                "integer": 1
            }
        ]
    }
}
```

Listing 4: Sample Proto document [2]

```
message Person {
    required string name = 1;
    required int32 id = 2;
    optional string email = 3;
}
```

3 Terminology and Background

This section covers some basic IO terminology which you may find useful as you proceed through labs where you will implement serialization libraries on your own.

3.1 Streams

Streams are a useful abstraction for handling incoming or outgoing data (or both). A stream is an abstract series of objects which may not all be available at any given moment. A common analogy is that streams are like conveyor belts which can either deliver objects towards or away from you; you can either place objects onto the belt or take objects off it [5].

A use of streams you may be familiar with is the `sed` (stream editor) utility in Unix. The utility can process files line-by-line without ever loading

the entire file at once. Streams are also the abstraction used to support network sockets.

Understanding what streams are is important as most serialization techniques yield a binary stream representing the serialized object.

3.2 Computer Files

Computer files are simply data containers, nothing more. The data is often contained in a 1D byte array and the file extension carries meaning as to how a program should read the contents of the file. Many file types also reserve a few bytes in the beginning of the file to store metadata about the file [6].

In terms of how the data within the file is organized, the file type has full control how data should be split into smaller units. For instance, plain text files often use newlines as a delimeter between "lines". It is important to realize that a computer file is very versatile; it is not difficult to invent a new file type, which we will do in a later lab.

4 Serialization in Microcontrollers

In the next labs will we implement two different serialization frameworks on our Arduino Nanos. We will use these frameworks to serailize data into a file using OpenLog. Here, we will investigate what factors must be considered when working on a microcontroller.

4.1 Constraints

An Arduino Nano has 32KB of storage, meaning the programs you write cannot be larger than 30KB, as 2KB of this are taken up by a program called a bootloader, which is what allows you to program your device [7]. Please note that going past $\approx 70\%$ of the allocated memory will lead to a warning stating `Low memory available, stability problems may occur`.

We will try to be careful when choosing a library so that the memory footprint is low. If you run into a warning like this, try to optimize your program! There are many resources online which give helpful tips on optimizing C programs.

4.2 OpenLog

In the following labs we will be writing our serialized streams to OpenLog. We will use files on the OpenLog as our medium for persistent data storage. If you are unfamiliar with the OpenLog, please review Lab 3: Writing and Plotting Data and the corresponding YouTube video.

4.3 Libraries

Many projects using microcontrollers—for instance a high altitude weather balloon payload or an automated gardening system—often have lots of sensors constantly collecting data. In order to use what we've learned in this lab to organize our data, we create an object to store sensor readings, and then serialize that object to our file.

4.3.1 ArduinoJson

The ArduinoJson library prides itself on being an efficient way to handle JSON in ArduinoC. It is primarily used to decode the output from web APIs like that of Twitter, popular weather services, etc. We can still use it for our purposes by constructing a JSON document, populating it with data measurements, and then serializing it to an SD card.

4.3.2 Nanopb

The Nanopb is a Protobuf library that also prides itself on being efficient, and was written specifically for embedded systems with restricted memory. It comes with a Protobuf Message encoder and decoder, allowing us to easily create data measurement messages and serialize those to our SD card.

4.4 Comparison of libraries

	Pros	Cons
ArduinoJson / JSON	Lightweight Easy to install	Data types aren't enforced by a schema
Nanopb / Protobuf	Lightweight Well defined data types	Tedious to install

References

[1] O. Ben-Kiki, C. Evans, and I. d. Net, *YAML Ain't Markup Language (YAML™) Version 1.2*, Jan 2009. [Online]. Available: https://yaml.org/spec/1.2/spec.html

[2] "Protocol buffers — google developers." [Online]. Available: https://developers.google.com/protocol-buffers/

[3] "Xml tutorial." [Online]. Available: https://www.w3schools.com/xml/

[4] "Yaml to json." [Online]. Available: https://www.json2yaml.com/convert-yaml-to-json

[5] S. Jessop, "What is a stream?" Aug 2009. [Online]. Available: https://stackoverflow.com/a/1216400

[6] "Computer file," May 2019. [Online]. Available: https://en.wikipedia.org/wiki/Computer_file

[7] "Arduino nano tech specs." [Online]. Available: https://store.arduino.cc/usa/arduino-nano