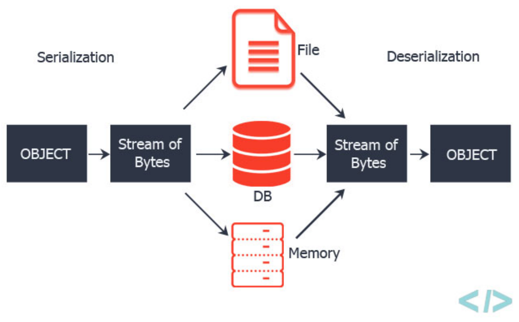
Lab 1: Introduction to Serialization

Neil Johari

May 14, 2019

1 Introduction



pc: codenuclear.com

Serialization (also known as marshalling) is a powerful data storage technique employed to deal with data that is intended to transmit or store the state of objects in a computer program.

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.

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) and YAML were pushed as light-weight alternatives to XML. JSON is often used to send data back and forth between web clients and servers, while YAML has become popular for configuration files. Since YAML 1.2, YAML is fully compatible with JSON as an official subset. [1].

In 2001, Google began work on 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]

```xml
<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]

```yaml
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
{
    "json": [
        "rigid",
        "better for data interchange"
    ],
    "yaml": [
        "slim and flexible",
        "better for configuration"
    ],
    "object": {
        "key": "value",
        "array": [
            {
                "null_value": null
            },
            {
                "boolean": true
            },
            {
                "integer": 1
            }
        ]
    },
    "paragraph": "Blank lines denote\nparagraph breaks\n",
    "content": "Or we\ncan auto\nconvert line breaks\nto save space"
}
```

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 abstracts a sequence 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. Typically, 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.

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 we will focus on importing and using two serialization frameworks on an Arduino Nano. We will pipe our stream output into a file using the OpenLog, and will later access this file on our computers and deserialize the written data. Here, we will investigate what factors must be considered when working on a microcontroller.

4.1 Constraints

An Arduino Nano has 32 KB of flash memory, of which 2 KB are used by the bootloader. There are a maximum of 30720 bytes left for your program. It's important to realize 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 optimization, and you can always reach out to your IA to get more information.

4.2 Libraries

4.2.1 ArduinoJson

4.2.2 Nanopb

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