Fluxion 1.0 Standard

by haltroy

Introduction

Throught our daily lives, we use markup languages to receive and give information to each other. The languages we use have to be converted to a specific format in order to be consistent with each other.

Most markup languages embed the information in a text-based form called “string” in modern programming languages. Strings are an array of numbers (defined by the encoding standard on how big the number is, example: UTF-16 uses 16-bit integer numbers that range of $\pm 2^{16} - 1$ which equals to 2 bytes in a file per character). Sometimes the information we have to store takes less than what a single character string can store (example: booleans usually depicted as “true” [4 characters] and “false” [5 characters] strings which takes 8 and 10 bytes for a 1/8th of a single byte information. For short; string-based markup languages generally use 64 to 80 times more space to store the same information).

Fluxion on the other hand, stores the data as what they are in memory. So a single byte that is used in the programming will still be a single byte instead of twice or thrice of the same size. This way, Fluxion can also read the data without the developer needing to convert from string back to the required value, saving time by only casting which is just telling the machine to consider that part as the required type.

Fluxion can store single informations such as Null, booleans, bytes and signed bytes as well as integers, characters, strings and byte arrays.

Nodes and Attributes

A Fluxion node is the heart of this operation. It can support different types of data and sub-nodes of itself. A Fluxion node can also have attributes which also can have different types but not sub-attributes or sub-nodes.

Data Types

Data types that Fluxion 1.0 supports and their identification bytes are given in the following table:

Data Type	ID
Null	0
Boolean (True state)	1
Boolean (False state)	2
Byte	3
Signed Byte	4
16-bit Character	5
16-bit Integer	6
16-bit Unsigned Integer	7
32-bit Integer	8
32-bit Unsigned Integer	9
64-bit Integer	10
64-bit Unsigned Integer	11
Float	12
Double	13
String	14
Byte Array	15

Encoding

Fluxion supports these encoding standards:

Encoding	ID
UTF-8	0
UTF-16	1
UTF-32	2

Variable-Length Encoding

Sometimes Fluxion has to store a 32-bit integer to determine the size of the name, the count of children/attributes or the size of the value. 32-bit integers costs 4 bytes and most of the time the other 3 bytes are just bunch of zeros. Instead of encoding the 4 bytes as it is, we use Variable-Length encoding to encode that 32-bit integer number. We check the 7^{th} (starts from zero) bit to determine if we should stop reading the next bits. If the 7^{th} bit is set, we stop reading any bits for the integer and we continue adding the next byte into our integer until we find that byte with 7^{th} bit set.

Header

A Fluxion file's header features these (in order):

1. The FLX mark: The first 3 bytes to determine if the file is a Fluxion file. These bytes are (in order) 0x46, 0x4C and 0x58.
2. The Version Byte: A single byte representing the version of Fluxion that used to create this file.
3. Encoding Byte: A single byte representing the Encoding (see table above) used for strings and names of each node/attribute.

Data

Fluxion encodes the data in this order for each node after encoding the header:

1. Value type and flags: This single byte determines the flags and the value types (see table above) about a single node. The first 4 bytes (ranging from 0 to 15) are used for the value types and the other 4 bytes are used for flags. The first flag determines if the node/attribute has a name, the second flag determines if this node has children and the third flag determines if the node has attributes. The last flag is not used, we don't recommend setting it for compatibility. Here's a schema about this byte:

0	1	1	1	1	1	1	1
Unused Flag	No Attribute Flag	No Children Flag	Has Name Flag	Value Type			

2. If the node has children (determined by the "No Children Flag" before), we encode the children count (a 32-bit integer number) with Vairable-length encoding here.
3. If the node has a name (determined by the "Has Name Flag" before), we first get the 32-bit integer using the Variable-Length encoding method to get the length of the name and then we read the name with the exact same size and convert it to the proper string using the encoding method mentioned in the header.
4. We read the value of the node. If the node is a byte array or string, we first read the length (32-bit integer) of the said byte array or string using the Variable-Length encoding and then read the value. Then if the value is string, we convert the read bytes into string using the encoding mentioned in the header.
5. If the node has attributes, first we read the length (32-bit integer) of the attributes list using Variable-length encoding and then we repeat the steps 1, 3 and 4 for each attribute.
6. If our node has children, we continue reading the stream.

Difference

With these differences, Fluxion can hold the exact same information better than its competitors. Here's a visualization of them (viewed from a text editor) in this table:

XML	JSON	YML	Fluxion
`<?xml version="1.0" encoding="utf-8" ?><root><User Value="mike" Age="35"><User Value="jeremy" Age="10" /></User></root>`	`{"user": {"name":"mike","age":35,"children":[{"user": {"name":"jeremy","age": 10}}]}}`	`user:  name: mike  age: 35  children:  - user:  name: jeremy  age: 10`	`FLXP MyRootNodeUsermike▒Age#>Userjeremy▒Age`
119 bytes	82 bytes	91 bytes	63 bytes

General Disadvantages of Fluxion:

- Can’t read the information directly from text editors.
- No comment support.

Advantages of Fluxion:

- Way smaller file size.
- Non-string values are no longer required to be parsed.
- Value and subnodes are separated from each other.
- Might be faster to process since the footprint of information is rather too small compared to others.
- Encoding of the file is embedded inside the file itself (similar to the XML) and supports multiple choices.