*A Meta Format for Anything*

# Structured Raw Meta Format

Johannes Jendersie
September 23, 2013

**Abstract**

Structured Raw a minimal meta format which stores binary information and can be used in many different ways. Like Json files this format stores its own specification so everybody should be able to understand the contents of a file by just looking at it. Sraw is capable of storing additional meta information to specify complex formats for models,... as well as packed binary data. Furthermore it is easy to change formats later without endanger the files compatibility.

**Contents**



## 1 Introduction

I like the feeling of Json files. You have data? Name it and write it down. You want to load it? Open it in a text editor and look how the data is named and what is given. What I don't like is that it takes more space and must be parsed on read. Storing large arrays is inefficient and might be easier.

So I came up with an own binary format specification which can cover both things. Of course you cannot look at it in a text editor since it is binary but a fast dump should be easy enough and will discover a sufficient amount of information.

In my opinion the main policies and ideas to achieve the targets are:

- Everything needs an identifier
- The format is hierarchical
- Everything has the same structure (no special cases)
- As few overhead as possible
- Large file support (64Bit)
- Iteration without reading the data should be possible

## 2 Specification

In the following grammatic things in capital letters are non-terminals and everything else terminals. Square brackets [] surround optional or conditional included data. {#∗} I will use as symbol for type-casted array data. Numbers for sizes are given in bytes. Strings has to be saved in UTF8. Every elementary data type must be saved as little endian.

| | |
|---|---|
| FILE | DATA |
| DATA | TYPE IDENTIFIER NELEMS [SIZE*] {#∗} |
| TYPE | CODE ELEM_TYPE |
| IDENTIFIER | STRING8 |
| CODE | 0x0 sizeof next NELEMS is 1<br>0x1 sizeof next NELEMS is 2<br>0x2 sizeof next NELEMS is 4<br>0x3 sizeof next NELEMS is 8 |
| ELEM_TYPE | 0x0 DATA<br>0x1 STRING8<br>0x2 STRING16<br>0x3 STRING32<br>0x4 STRING64<br>0x5 bit<br>0x6 int8<br>0x7 uint8<br>0x8 int16<br>0x9 uint16<br>0xa int32<br>0xb uint32<br>0xc int64<br>0xd uint64<br>0xe float<br>0xf double |
| NELEMS | uint8, uint16, uint32 or uint64 depending on CODE flag |
| SIZE | uint64 |
| STRING8 | uint8** {#uint8} |
| STRING16 | uint16** {#uint8} |
| STRING32 | uint32** {#uint8} |
| STRING64 | uint64** {#uint8} |

* The size is given for DATA and STRING element types. It is the data block size until the next data block. It can be used to skip everything after NELEMS/SIZE to the beginning of the next

DATA on the same level. This might be useful for faster iteration (e.g. dump of identifiers). For all types except DATA, STRING and bit the size can be computed as sizeof(Type) * NELEMS. For bit type the formula is ceil(NELEMS/8).

** The length of the string is given in advance

## 3 Example Interface

```
// Access elementary data
int i = (int)SomeNode["ScreenWidth"];

// Stable access with a default value if data
// is not found.
float f = SomeNode["Speed"].GetFloat(1.618f);

// Array access in both variants again
a[3] = (float)SomeArray[3];
a[4] = SomeArray[4].GetFloat(1.618f);

// Hierarchical
Root["Config"]["Device"]["Width"].GetInt(1024);

// Write access (build from nodes)
Root.Add("Config", SomeNode);
// Write access (add elementary [array] data)
SomeNode.Add("Indices", UINT16, 3141);

// Direct read/write access (if existing)
void* pData = SomeNode.GetData();
```