TokenNameValueTypes

Intro

Token Name values

The ofp engine, and specifically, a raPified (binarised) file identifies only a few different 'types' of Token Name

```
aString = "A string";
anInteger = 1234567;
aFloat = 0.0123;
anArray[] = {.....};
```

Everything that can be stated in a config.cpp, a description.ext, a mission.sqm for TokenName value pairs devolves to one of the above 'types'.

Boolean

Separately, a **fifth** type exists called boolean.

```
aBoolean = 0;
```

In fact the engine only stores and 'sees' this value as an integer. But by convention in humanly readable text files (config.cpp vs config.bin), integers that can only have zero or non zero values are declared in statements as

```
#define false 0
#define true 1
LightOn = true;
```

A **boolean** as such, *does not exist* in binarised (raP) files. It is an integer. It is *only* a poetic licence for an author when creating text (config.cpp) files.

Integers

```
anInteger = 99;
```

integer values are signed values held in four bytes of memory when raPified.

Degrees

a signed integer between -360 and +360. There is no degrees type. It is simply a poetic licence in well written config.cpp's, similar to boolean.

Floats

```
aFloat=1.0;
```

float values are held in four bytes of memory when raPified. Note that this automatically means it is not, in C terms a double.

Strings

Anything enclosed in quotes ("...") is automatically a string. In addition, anything that cannot be represented as a float or integer, is also a string.

MathFormula

A catch to the unwary is

```
aString= 1 * 6 + 99/3;
```

this is **not** binarised into an integer (or a float), but remains 'in string'

See At the End of the Day for what actually happens to this 'string'.

Arrays[]={...};

Arrays are of unrestricted content in terms of what 'types' they can have in them, and, their number.

Arrays, can be (and often are) embedded within arrays.

Arrays can have mixtures of type in the same array

```
sound[] = {"fuel_explosion",10.000000,1};
```

Quite frequently a specific array will have a fixed number of dimensions. For instance

```
color[]={1.0,0.3,0.6,0.0};
```

The fact is, the color array as used by the ofp engine, has, 4 values. (all of them floats)

However consider this

```
cargoAction[]={"ManActM113Medic","ManActM113Medic","ManActM113Injured"};
```

the number of elements in this array, happens to be 3, and happens to describe a Medic bmp that can carry three soldiers!

Other models carry more (or less).

At the End of the Day

The engine will always look at its raPified (binarised) data in the context it wants. A config.cpp - generally a text file - will always be processed into an internal config.bin (raPified) before use by the engine.

Thus, while a value is stated to be a float

```
count=1.0;
```

it is only the **TokenName** that is of interest to the engine initially.

Thus

```
count="1.0";  // a string
count=1;      // an integer
count="1"     // a string that looks like an integer
```

all represent the floating value **1.0**.

The engine will massage the token name to relevance.

In config.cpp, and thus its raPified equivalent config.bin, it is a matter of convenience and an ease in processing load to write the TokenName in the 'type' most often used.

For examples of this duality see `count` in Magazine Parameters and Weapon Parameters in the VBS Developer Reference.