TCC Low Level Parser

Purpose

The TCC low level parser is designed to convert the low level ASCII based configuration files into a binary format which can then be downloaded to the Alpha processor boards. This parser only performs a very minimal check on the data because it assumes the high level parser has already done this.

Input File Syntax

The parser expects the input file to be divided into objects. Each object must start with the declaration:

```
new Object {
```

where "Object" is the type of object being declared. The end of an object definition is a line containing a single "}" symbol.

Between the pairs of brackets there must be a series of fields keyword / value pairs which are defined using the syntax:

```
KEYWORD = value
```

The parser program requires at least one space between the keyword, the equal sign and the value. Three data types are permitted as values. These are:

- **String** – any series of non-space characters enclosed by double quotes
- **Float** – any non-string value containing a decimal point
- **Integer** – any non-float and non-string value

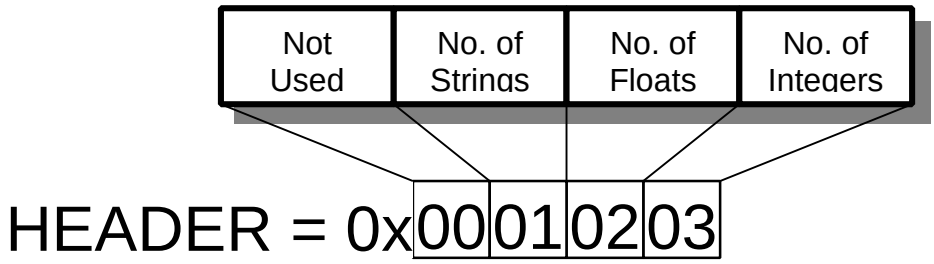
These rules show how the program determines the field type. Integers and floats are evaluated using the "atoi()" and "atof()" library routines. This means that if they are passed a non-numeric value the result will be read as zero.

Multiple values may be declared for a single keyword by separating them with commas. For example:

```
KEYWORD = value1,value2,value3,value4
```

Is an allowable syntax, with each value potentially being of a different data type. The parser then associates each value with the given keyword.

Each object must have as its first keyword “HEADER” and this must contain the number of each data type which is in the object encoded into a single 32-bit word as illustrated below:



This header is compared against the actual number of each data type which are found and if there is a difference and error is generated.

Further checking of parameters is performed by the relevant object parser. This is passed a list of keyword / value pairs and then performs minimal checking to ensure the object is valid. If the object is not valid and error is generated and the parser will abort.

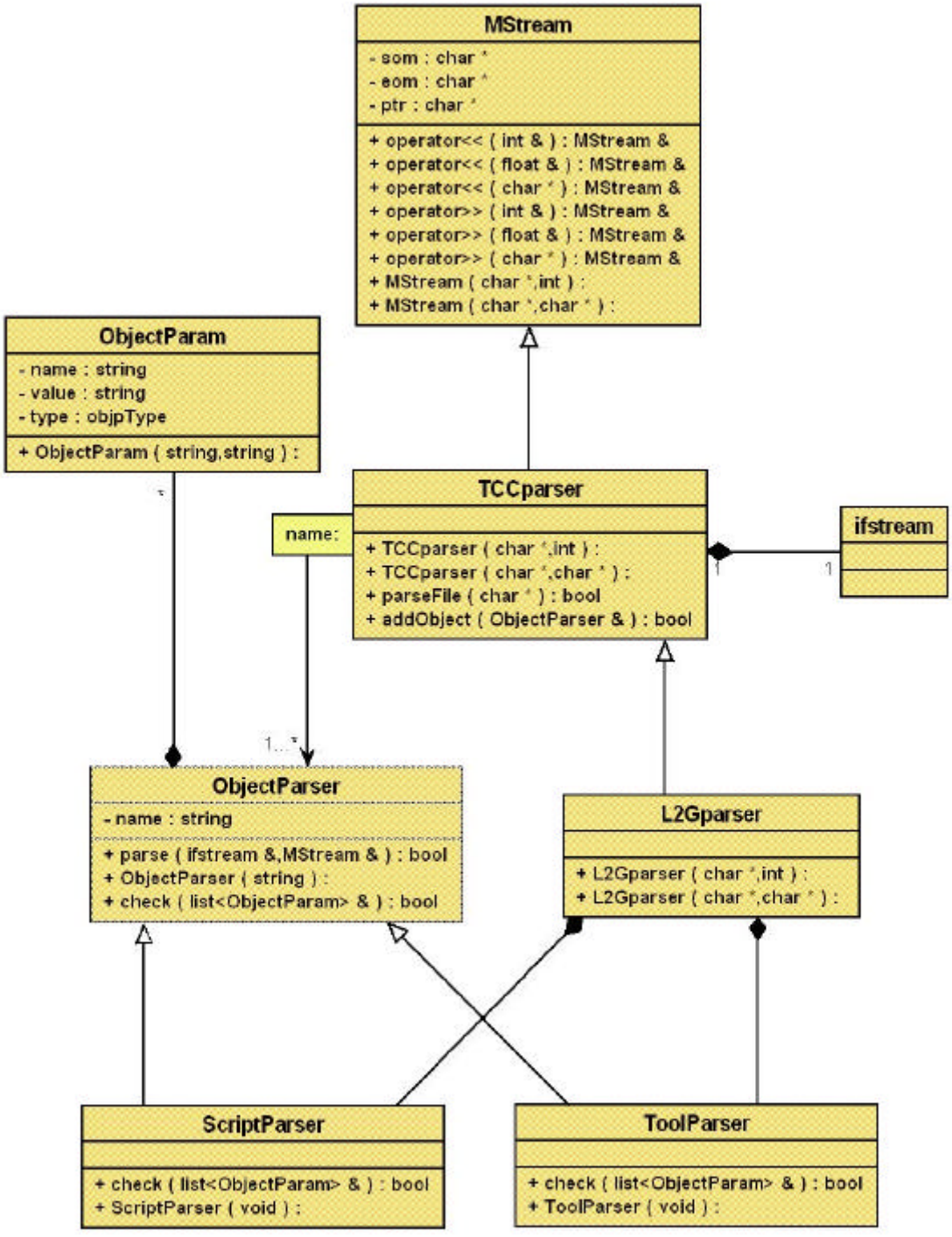
Example

An example of a possible level 2 global configuration file is given below. This file contains two types of objects, scripts and tools.

```
new Tool {
     HEADER    = 0x010203
     TYPE      = 1
     INDEX     = 2
     NAME      = "Loose_Electron"
     DPHI      = 0.3
     EOPMIN    = 0.8
}

new Tool {
     HEADER    = 0x010203
     TYPE      = 2
     INDEX     = 2
     NAME      = "Loose_Jet"
     DPHI      = 0.3
     DETA      = 0.3
}

new Tool {
     HEADER = 0x010008
     TYPE   = 3
     INDEX  = 1
     NAME   = "e/Jet_Mass"
     NTOOL  = 2
     TOOL   = 1,2
     TOOL   = 2,2
}

new Script {
     HEADER = 0x01030c
     BIT    = 1
     NAME   = "e/Jet/Mass_Script"
     NTOOL  = 3
     TOOL   = 1,2,10.0,2
     TOOL   = 2,2,10.0,1
     TOOL   = 3,1,50.0,1
}
```

Technical Implementation

The class diagram for the TCC parser program is shown below with the specific classes need to interpret the Level 2 global configuration file added.



The floats and integers are all stored as 32-bit quantities in the Alpha processor byte order. Strings are stored as null terminated character arrays which are padded up to the next multiple of 4 bytes. This is to ensure that all integer and float quantities are stored on 4-byte aligned memory locations as required by the Alpha processor.

The *MStream* class performs the conversion into binary as well as swapping the data into Alpha byte order (if needed) and padding strings to 4-byte multiples. It is written without using the STL library so that it can also be compiled on the Alpha board and used for reading in the binary data.

The *TCCparser* class controls the input of ASCII file and checks each line for the start of a new object. When a new object is found the relevant *ObjectParser* class is called to read in and then check the object. Specific parsers for different input file formats inherit from the generic *TCCparser* class and declare specific *ObjectParsers* to the inherited *TCCparser* class. It is these *ObjectParser*s which are passed the object found in the input file. If no parser is found for a given object then a error will be generated.

The specific object parsers all implement the virtual method *check()* which is passed a list of *ObjectParam* classes. There is one instance of this class for each object parameter found and the class stores the keyword, the value and the data type. The purpose of the check function is to ensure that the basic fields required for all objects of that type are present. If they are the routine returns “true” otherwise it must return “false” which will cause the parser to abort.