**Conversion Rules for FROST**

Pete Howard 3rd April 2016

**Order of Precedence**

The sequence in which everything below is stated is heavily significant. The sequence defines an order of precedence for all the logic.

**File Type Inference**

Many <> = XML

Many {} = JSON

Many commas = CSV

Colons at end of lines, or sequences of lines starting with minus = YAML

Many semi colon or equals as second field in line = .Ini

Default = white space delimited

**Fields and Types**

We define fields as the smallest indivisible elements present in files that carry data rather than being part of the file syntax.

We will infer a type for each field {float, boolean, int, string}.

The type is inferred lexically in the order of precedence listed.

Inference of floats includes from scientific notation, but not hex or octal conventions.

Inference of Booleans includes the ‘y’ and ‘n’ strings (case insensitive)

**Tags and Naming Conventions**

We intend to exploit naming and lexical conventions to subdivide fields still further. For example to isolate the implied fragments from “ANN\_OUT\_LE\_03”, or “ANNOUTLE03”.

FROST’s output will always include the complete fields – so nothing is lost, but it will often also make the sub-tags available for fields as well to aid machine-reading of the output.

The rules for recognizing tags (“Tagification”) are as follows:

When a field can be judged to have more than one syllable when split on the underscore or minus character. Then the syllables thus defined become the tags.

When a field can be judged to have more than one syllable when split on letter and number segments. Then the syllables thus define the tags. A letter or number sequence can be as little as a single letter or number. Contiguous letters or contiguous numbers however constitute a single syllable.

We infer the type of tags once they have been isolated exactly the same as we do for fields.

**Whitespace File Type**

Files are made from discrete lines – as delimited by linefeed or newline characters or the pair. Plus EOF.

In what follows, trailing and leading whitespace in lines is ignored.

A line that starts with a hash is a comment line.

A comment line is not necessarily passive.

Unless otherwise stated, empty, or whitespace-only lines do not influence the following interpretation rules.

Entities are output in FROST in the same order as they are encountered.

**Fields**

Comment lines are not split into fields.

Non comment lines are treated with a conceptual pre-pass, that modifies single or double-quoted strings that are present. These have any constituent spaces replaced (conceptually at any rate) with a special sentinel character and the quotes are removed.

Then fields are isolated by splitting the lines on whitespace sequences.

**Tables**

We define and capture 2D tables.

A table is defined as a series of 2 or more contiguous lines which:

Are not comment lines.

Have 2 or more fields.

Have the same number of fields.

Have sufficiently matching field types. (treat int & float as matching)

Tables will show up in the output in the same order as they appear in the input.

We support the notion of tables having meta data present in comment lines. Like a name or key for the table, and names or keys for the constituent columns.

We look for table meta data in comment lines that immediately precede a table as follows.

**Column Headings**

If a comment line exists preceding a table, with no intervening blank line, we conceptually remove the hash character and attempt to split the remainder of the line into fields. If this yields the same number of fields as the table has columns. Then the fields are taken to be column names for the table.

The columns in FROST tables will be addressable by these column names when present, in addition to their column number.

**Table Names or Keys**

If a comment line exists preceding a table, regardless of intervening blank lines, and we have not previously decided that that line contains column headings. Then we regard that part of the line following the hash as being a name or key for the table that follows.

FROST tables will be addressable using these table names or keys in addition to using the fall back method of using the sequence of tables to discriminate between them.

The table name will also be tagified.

**Key – Value Pairs**

Remember we are writing these detection rules in order of precedence. So it follows that for what follows to kick in, a line must have been judged not to be part of a table.

If the line comprises three fields, the middle of which is the equals character or a colon character. Then the first and third will be taken to be the elements of a key-value pair.

If the line comprises two fields. Then these are taken to be the elements of a key-value pair.

**Single Rows of Values**

We might have relaxed the definition of a table to include tables with just a single row to capture a row of values. But we suspect this would be too eager to conclude thus and would thus mask other interpretations that we would wish to take higher precedence. Like the key-value pair above, or something else we haven’t realised we need yet. So here it is as a relatively low-precedence choice.

A row with three or more fields is taken to be a single row of values and will be included in FROST as a sequence of (inferred typed) values.

**Stand Alone Values**

The only type of non-comment line that can get through to here is a line with just one field. These will be treated by FROST as a stand alone, implicitly-typed value. The only structural interpretation available to the consumer will be where in the input sequence it occurs.

**The Structure of Last Resort**

It is not unusual for people to invent somewhat arbitrary structure meta data in these file. Often using comment lines. We saw a very common example earlier with tables. But we need an open and flexible general purpose method to provide an open architecture for structure in FROST.

To provide this we are going to assign a serial number to every single element in FROST. Not only every field that is output, but every element and container, and every comment line. Everything has a serial number. The serial number comes from the sequence things appear 1,2,3...n and can be viewed as that item’s FROST address. This is part of FROSTs fundamental output and therefore provides machinery of last resort for client apps to navigate and find things.

Because comments are most likely the key to ad-hoc structural arrangements we do something special for them.

We attach to every comment element in the FROST output a sub element dictionary that gives its position relationships with other things in terms of their FROST serial numbers:

When this comment line is part of a contiguous block of comment lines – the range of serial numbers between which the others can be found.

Is the next, non-comment thing a table? If so which one.

Is the next thing a key-value pair. If so which one.

Etc.

**CSV File Type**

We choose to define the processing of this file type, purely in terms of deltas from the processing of white space files, as they have most of the concepts in common.

**Fields**

Non comment lines that contain quoted strings are not altered.

The fields are isolated by splitting the lines on commas, and then trimming each field thus found of leading and trailing space.

**Key – Value Pairs**

This rule for white space files **IS NOT** applied. If the line comprises three fields, the middle of which is the equals character or a colon character. Then the first and third will be taken to be the elements of a key-value pair.

**.Ini / YAML / JSON / XML File Types**

These file types are all strongly structured natively, so their native structure is carried through to FROST with only minor interference.

Nothing is removed.

The only addition is the inclusion of tagification to good looking candidate fields.