JAMN - JAMN (is) Asset Meta Notation
Spec - Prerelease

Basic Types

There are a small set of basic types:

Array
Object/Dict
Encoded
Number
String
Boolean

These are further specified by optional type designators.

Array

An array is an ordered list of values. An array is introduced by the single square bracket character [
Following this are 0 or more values, each value is terminated with a semi-colon character ;
Finally a closing square bracket] finishes the array.

<array> ::= '[' [<value>';'] ... ']'

Object (Dict)

An object is introduced by a single brace character { followed by 0 or more fields, each field is terminated
with a semi-colon character ;. Finally a closing } terminates the object.

A field is a string, followed by a colon character : followed by a value.

<field> ::= <string> ':' <value>
<object> ::= '{' [<field>';'] ... '}'

Number

A number is an (optional) - character, followed by a set of digits with optional decimal point and exponential.
They MUST start with a digit (or minus) and no whitespace may occur within a number.

Numbers have a minimum range of 2^64 (int64/uint64), or float64. However implementations MAY support larger numbers.
Larger numbers MUST be prefixed with a type designator (and may be rejected by implementations), it is an error
for numbers larger than the minimum supported range to NOT be prefixed.

Special values %nan and %inf, %negnan, %neginf introduce the (floating point) Not-a-number and infinity.

Hex (0x...), Octal (0o...), Binary (0b...) numbers are available. The prefix MUST be lower case and
appear immediately after the opening 0. Prefixed numbers may NOT have a '-' character prefix.

Underscores may occur in a number anywhere after the first digit and are ignored.

Exponential specifiers are either 'e' or 'E' which may occur once within the number.

Numbers MUST be terminated by whitespace or a semicolon, no other character may occur after a number.

String

A basic string is surrounded by "" quotes. Within a basic string there are various escape sequences:

\" (embed quotes)
\n (embed newline)
\\ (embed slash escape character)

A multiline string is surrounded by `` back quotes. If the opening quote is immediately followed by a newline
then that newline is omitted, all other newlines are included in the string.
The back quote can be embedded with a double ``. No other escape sequences are available.

Strings have a maximum length of 128 MB. Strings MAY be larger if they are prefixed by a type designator ($str_<len>),
however implementations must reject documents with larger strings if they are unsupported.

Ident (or Naked) strings are an ASCII subset of Unicode, they must start with an alpha character, underscore or . and may continue with any combination of alpha-numerics, underscores or . / \ .

Ident strings have a maximum length of 256 characters (necessarily 256 BYTES as ASCII subset).

<string> ::= <basic-string> | <ident-string> | <multiline-string>

Boolean

Boolean values are specified by the special values %true and %false.

<boolean> ::= '%true' | '%false'

Null

The null value is specified by the special value %null.

<null> ::= '%null'

Semi-colons
===========

Semi-colons terminate every value in an array or field in an object.

There are automatic insertion rules.

Within an object a semi-colon is inserted into the stream at any newline after any ordinary token (number, string, special value) or] or }. Semi-colons are also inserted before an outer closing] or } token.

Within an array a semi-colon is inserted into the stream after any whitespace after any ordinary token.

Semi-colons can still be used to present fields on a single line. Note that the semicolon is a *terminator*
and comes after all values (arrays and objects included). A semi-colon is also introduced at EOF/end of stream.

Extraneous semi-colons are an error (they don't introduce a null value, only the %null can do that). However the insertion rules allow any amount of newlines or whitespace (as appropriate) and only one semi-colon will be added.

PseudoTypes (ptypes)

All values may be preceeded with a type designator. A type designator is introduced with a $ immediately followed by a string (NO whitespace is allowed following the $). The interpretation of type designators is implementation specific, however a set of types are recommended:

f32, f64, i8, i16, i32, i64, str, ref, any.

These are called PseudoTypes (abbreviated to ptype). The 'any' type is the default type of objects within Jamn.

<ptype> ::= '$' <string>

Array Types

An array type is specified using an element type, a trailing underscore and an optional element count.
The element type specifier may include underscores, it is only the final underscore that counts.
This rule allows specifying nested arrays.

<elem-type> ::= <string>
<num-elems> ::= <number>

<arr-type-string> := <elem-type>_[<num-elems>]

Specifies an array of <elem-type>, optionally of length <num-elems>.

An implementation is allowed to ignore the specifiers and allow the following value to *not* be an array, but
implementations *are* allowed to call such a document invalid. It is also allowable for the actual elements and length
of an array to not match the ptype, however again this may be an invalid document for a given implementation.

Encoded

An encoded object consists of character '=' followed immediately by a string specifying the encoding, followed by a single '=' character, followed by the encoded data.

Whitespace *should not* appear after the initial space following encoding string, however if the encoding allows ignoring whitespace then it is allowed.

$bigstring ="base64"= TWFueSBoYW5kcyBtYWtlIGxpZ2h0IHdvcmsu

The exact encoding depends on the encoding method and the encoding string may include length or delimiters etc. as colon seperated fields. Due to the nature of encodings an implementation may reject a document as invalid if it does not support the encoding, however implementations *may* make a best-effort attempt

to look for an end-of-value semicolon (or brace etc.) in order to skip the encoded value.

The type designator is optional.

Since a JAMN document must be UTF-8 it isn't possible to include binary data directly.

Comments

Comments are values and may only occur in specific places.
A comment begins with a # and continues to the end of a line. # characters within strings are NOT considered
as introducing comments.

Top Level Values (Documents)

Any value is allowed at top-level and DOES NOT NEED surrounding braces or brackets. However this introduces an
ambiguity which is resolved simply - the first value seen determines whether the top-level is an
object or an array. If there is a ':' character then the parser is within an object, otherwise it must be an array.

References/Pointers

Jamn documents support the notion of reference or pointer. These are introduced by type designator $ref
on a string value. The string defines the path (using forward slash '/') from the root of the document to a particular value. Arrays are 0-indexed integers and object fields are referenced by strings.

Grammar

```
<basic-value> ::= <null>
                | <boolean>
                | <number>
                | <string>
                | <array>
                | <object>

<value> ::= [<ptype>] <basic-value>
```