Smalltalk Object Notation (STON)

Sven Van Caekenberghe

February 14, 2012

Abstract

Smalltalk Object Notation (STON) is a lightweight, text-based, human-readable data interchange format for class-based object-oriented languages like Smalltalk. It can be used to serialize domain level objects, either for persistency or network transport. As its name suggests, it is based on JSON (Javascript Object Notation). It adds symbols as a primitive value, class tags for object values and references. A working reference implementation in Pharo Smalltalk is available as a proof of concept.

Contents

1 Introduction

1.1 Example

If you have ever seen JSON (Javascript Object Notation) [1], you will be instantly familiar with Smaltalk Object Notation. It uses similar primitive values, with the addition of a symbol type. Some details are slightly different though. For object values an optional class tag precedes a representation that is either a list of values or a map of keys and values. Exactly how each class is represented is up to the implementation, although this document suggests some conventions for well known classes.

Next is an example of what pretty printed STON for a simple object looks like. Even without further explanation, the semantics should be clear.

[1] See http://www.json.org for the specification.

```
TestDomainObject {
  #created : DateAndTime [ '2012-02-14T16:40:15+01:00' ],
  #modified : DateAndTime [ '2012-02-14T16:40:18+01:00' ],
  #integer : 39581,
  #float : 73.84789359463944,
  #description : 'This is a test',
  #color : #green,
  #tags : [
    #two,
    #beta,
    #medium
  ],
  #bytes : ByteArray [ 'afabfdf61d030f43eb67960c0ae9f39f' ],
  #boolean : false
}
```

1.2 Rationale

JSON seems to have hit a sweet spot: it is very simple, yet just powerful enough to represent some of the most common data structures across many different languages. JSON is very readable and relatively easy to type.

However, JSON knows only about lists and maps. There is no concept of object types or classes. This means that it is not easy to encode arbitrary objects, and some of the possible solutions are quite verbose [2].

Adding a symbol (globally unique string) primitive type is a very useful addition: because symbols help to represent constant values in a compact and fast yet readable way, and because symbols allow simpler and more readable map keys.

Allowing shared and circular object structures is also necessary simply because these exist in reality and because they allow for naturally efficient object graphs.

2 Smalltalk Object Notation

2.1 Values

STON encodes a value, which is either a primitive value or an object value. The undefined object *nil* and a reference to an already encountered object are considered values as well.

2.2 Primitive Values

The primitive types are numbers, strings, symbols and boolean.

Numbers are either integers or floats. Integers can be of infinite precision. Floats can be simple fractions or use the full scientific base 10 exponent notation.

Strings are enclosed using single quotes. A backslash is used as an escape mechanism. Some unreadable characters have there own escape code, like in JSON. A general Unicode escape mechanism using four hexadecimal

[2] Encoding the type or class as a property and/or adding an indirection to encode the object's contents.

digits is used to encode any character. STON conventionally encodes all non-ASCII characters.

Symbols are introduced by a sharp sign. Symbols consisting of a limited character set (letters, numbers, a dot, underscore, dash or forward slash) are written literally. Symbols containing characters outside this limited set are encoded like strings, enclosed in single quotes.

Booleans consist of the constants *true* and *false*.

2.3 Object Values

Like JSON, STON uses two primitive compositions: lists and maps. Lists consist an ordered collection of arbitrary objects. Maps consist of an unordered collection of key-value properties. Keys can be strings, symbols or numbers while values are arbitrary objects.

An object in STON has a class tag and a representation. A class tag starts with an alphabetic uppercase letter and contains alphanumeric characters only. A representation is either a list or a map.

There are generic ways to encode arbitrary objects. Non-collection classes are encoded using a map of their instance variables, instance variable name symbol mapped to instance variable value. Collection classes are encoded using a list of their values. Some classes have there own custom representation, often chosen for compactness and readability.

For one selected list like collection subclass, currently *Array*, the class tag is optional, given a list representation. For one selected map like collection subclass, currently *Dictionary*, the class tag is optional, given a map representation. The following pairs are thus equivalent:

```
[1, 2, 3]  =  Array [1, 2, 3]
{#a : 1, #b : 2} = Dictionary {#a : 1, #b : 2}
```

During encoding, some classes can ask for a short list representation. This will instruct the pretty printer to print their encoding on one line.

2.4 References

To support shared objects and cycles in the object graph, STON adds the concept of references. Each object value encountered during depth first traversal is numbered from 1 up. If a object is encountered again, only a reference to its number is recorded. References consist of the at-sign followed by a positive integer. After reconstructing an object graph, references are resolved. Here is an OrderedCollection with three times the same, shared Point object:

```
OrderedCollection [ Point [1, 2], @2, @2 ]
```

A two element Array that refers to itself in its second element will look like this:

```
[ #foo, @1 ]
```

Whether or not strings count as objects and possible shared references, is an open question.

2.5 Conventional Representations

In the current working reference implementation in Pharo Smalltalk, a number of classes received a special, custom representation. These are:

Time a one element array with an ISO style HH:MM:SS string

Date a one element array with an ISO style YYYYMMDD string

DateAndTime, TimeStamp a one element array with an ISO style YYYY-MM-DDTHH:MM:SS.N+TZ.TZ string

Point a two element array with the x and y values

ByteArray a one element array with a hex string

Character a one element array with a one element string

2.6 Implementation

To validate the Smalltalk Object Notation a reference implementation was implemented in Pharo Smalltalk. It is available here:

`http://ss3.gemstone.com/ss/STON`

It works in Pharo Smalltalk versions 1.3 and 1.4. The project contains a full complement of unit tests.

The key methods are instance method *stonOn:* and class or instance method *fromSton:*. Classes can use another external name using the class method *stonName*.

3 BNF

```
value
  primitive-value
  object-value
  reference
  nil
primitive-value
  number
  true
  false
  symbol
  string
object-value
  object
  map
  list
object
  classname map
  classname list
reference
  @ int-index-previous-object-value
map
  {}
  { members }
members
  pair
  pair , members
```

```
pair
  string : value
  symbol : value
  number : value
list
  []
  [ elements ]
elements
  value
  value , elements
string
  ''
  ' chars '
chars
  char
  char chars
char
  any-printable-ASCII-character-
    except-'-or-\
  \'
  \\
  \/
  \b
  \f
  \n
  \r
  \t
  \u four-hex-digits
symbol
  # chars-limited
  # ' chars '
chars-limited
  char-limited
  char-limited chars-limited
char-limited
  a-z A-Z 0-9 - _ . /
classname
  uppercase-alpha-char alphanumeric-char
number
  int
  int frac
  int exp
  int frac exp
int
  digit
  digit1-9 digits
  - digit
  - digit1-9 digits
frac
  . digits
exp
  e digits
digits
  digit
```

```
  digit digits
e
  e
  e+
  e-
  E
  E+
  E-
```