Blink JSON Format Specification

beta4 - 2013-06-05

This document specifies a method for encoding messages structured by a Blink schema into the JSON format.

Contents

Appendices

1 Overview

The core Blink specification [BLINK] defines a schema language for specifying the structure of data messages. It also defines a binary format for messages defined by such schema.

This specification provides a complementary format where messages are encoded in JavaScript Object Notation [JSON].

In the JSON format:

- Messages and dynamic groups are mapped to JSON objects. The object has a string property named **`$type`** and the value is the name of the group.
- Within a group, each field is mapped to a property with the same name as the field.
- Primitive data values are represented as numbers, strings and Booleans.
- Subgroups are represented as JSON objects.
- Sequences are represented as JSON arrays.

The following shows what the first example in the core Blink specification would look like in the JSON format:

Assuming the schema:

```
Hello -> string Greeting
```

a **`Hello`** message carrying the greeting "Hello World" would be encoded as

```
{ "$type": "Hello", "Greeting": "Hello World" }
```

2 Basic Mapping

The following sections specifies how to represent Blink messages in the JSON format.

2.1 Wrapper Array

A top-level stream of Blink messages is represented as a JSON array:

```
[
  { "$type": "Hello", "Greeting": "Hello" },
  { "$type": "Hello", "Greeting": "Hello Again" }
]
```

2.2 Message

Top-level messages and internal dynamic groups are represented as JSON objects. Each such object has a special string property named **`$type`** that contains the name of the group. If the group is defined in a schema with a namespace declaration, then the name should be qualified by prefixing it with the namespace name and a colon.

Assuming a schema:

```
namespace Draw
Rect -> ...
```

then a **Draw:Rect** message would be represented as

```
{ "$type": "Draw:Rect", ... }
```

2.3 Field

Each field of a group is represented by a property with the same name. The order of the field properties is not significant.

An optional field that has no value is simply left out.

Assuming a schema:

```
namespace Draw
Rect -> u32 Width, u32 Height, string Text?
```

then an instance could look like

```
{ "$type": "Draw:Rect", "Width": 10, "Height": 20 }
```

or, including the optional **Text** field:

```
{ "$type": "Draw:Rect", "Text": "Square", "Width": 17,
 "Height": 17 }
```

2.4 Integer

Integers with the bit widths 8, 16, and 32 are encoded unmodified as JSON numbers. 64-bit integers are encoded as unmodified JSON numbers if the absolute value is less than 10^{15}. Otherwise, the value is encoded as a string containing the number in decimal notation. This is to prevent information loss due to the limited precision of JSON numbers.

2.5 Decimal

A **decimal** value where the absolute value of the mantissa is less than 10^{15} is encoded as a JSON number in decimal or scientific notation. Otherwise, the value is encoded as a string containing the decimal or scientific representation of the number. This is to prevent information loss due to the limited precision of JSON numbers.

2.6 Floating point

An **f64** value is represented as a JSON number in decimal or scientific notation. The values positive and negative infinity and not-a-number are mapped to the strings **"Inf"**, **"-Inf"** and **"NaN"** respectively.

2.7 String

String values are encoded as JSON strings. If a string type has a max size property, then the string must not be longer that the specified value when represented as a [UTF-8] byte sequence.

2.8 Binary and Fixed

If a binary or fixed value is a valid UTF-8 sequence, then it can be encoded as a JSON string, otherwise it must be encoded as a *hexadecimal list*. A hex list is a JSON array of strings. Each string comprises hex digits and possibly spaces. The corresponding byte sequence value is obtained by concatenating all the strings, removing any spaces and finally translating each pair of hex digits into a byte. It is an error if the number of hex digits is not a multiple of two.

If a binary type has a max size property, then the byte sequence that results from a hex list must not be longer that the specified value. For a fixed type, the byte sequence must be exactly as long as specified in the size property.

Assuming a schema:

```
inetAddr = fixed (4)
Packet -> inetAddr Host, ...
```

then a host address can be represented like this:

```
{ ... "Host":["3e 6d 3c ea"] ... }
```

2.9 Boolean

A Boolean value is mapped to a JSON Boolean.

Assuming a schema:

```
Flags ->
  bool Trace, bool Warn
```

then a message could look like this:

```
{ "$type": "Flags", "Trace": false, "Warn": true }
```

2.10 Time and Date

Values of the time and timestamp types **millitime**, **nanotime**, **date**, **timeOfDayMilli** and **timeOfDayNano** are represented as strings. The format of these strings is the same as specified for these types in the Blink Tag Specification [TAG].

2.11 Enumeration

An enumeration value is represented by the corresponding symbol name from the schema:

```
Color = Red | Green | Blue
Car   -> Color Color
```

Assuming the schema above, the color of a car would be represented like this:

```
{ "$type": "Car", "Color": "Blue" }
```

2.12 Sequence

A sequence is represented as a JSON array.

Assuming a schema:

```
Sample -> u32 Values []
```

then a message could look like this:

```
{ "$type": "Sample", "Values": [1, 2, 3] }
```

2.13 Static Group

A static group value is represented by a JSON object.

Assuming a schema:

```
Point -> u32 X, u32 Y
Line  -> Point From, Point To
Path  -> Point Points []
```

then a message could look like this:

```
{
  "$type": "Line",
  From: { "X": 0, "Y": 0 },
  To: { "X": 10, "Y": 10 }
}
```

A sequence of static groups would look like this:

```
{
  "$type": "Path",
  "Points": [
    { "X": 0, "Y": 0 },
    { "X": 10, "Y": 10 },
    { "X": 17, "Y": 10 }
  ]
}
```

2.14 Dynamic Group

A dynamic group is represented in the same way as a top level message: by a JSON object with the special **`$type`** string property indicating the dynamic type of the object.

Assuming a schema:

```
Shape
Rect   : Shape -> u32 Width, u32 Height
Circle : Shape -> u32 Radius
Frame  : Shape -> Shape* Content
Canvas -> Shape* [] Shapes
```

then a **`Frame`** is represented like this:

```
{
  "$type": "Frame",
  "Content": {
    "$type": "Rect",
    "Width": 10,
    "Height": 20
  }
}
```

A sequence of dynamic groups would be represented like this:

```
{
  "$type": "Canvas",
  "Shapes": [
    { "$type": "Rect", "Width": 10, "Height": 20 }
    { "$type": "Circle", "Radius": 10 }
  ]
}
```

3 Message Extensions

Extension content of messages or internal dynamic groups is represented as the special property **`$extension`**. The value of the property is an array of dynamic group objects corresponding to the extension content.

Assuming a schema:

```
Mail  -> string Subject, string Body
      Trace -> string Hop
```

and an extended message:

```
@Mail|Subject=Hello|Body=How are you?|
      [@Trace|Hop=local.eg.org;@Trace|Hop=mail.eg.org]
```

then it would be represented like this in JSON:

```
{
  "$type": "Mail",
  "Subject": "Hello",
  "Body": "How are you?",
  "$extension": [
    { "$type": "Trace", "Hop": "local.eg.org" },
    { "$type": "Trace", "Hop": "mail.eg.org" }
  ]
}
```

A References

BLINK	http://blinkprotocol.org/spec/BlinkSpec-beta4.pdf
JSON	http://json.org/
TAG	http://blinkprotocol.org/spec/BlinkTagSpec-beta4.pdf
UTF-8	http://tools.ietf.org/html/rfc3629