A JSON Encoding for X3D

Roy Walmsley
Web3D Consortium
roy.walmsley@ntlworld.com

Donald Brutzman
Naval Postgraduate School
brutzman@nps.edu

John Carlson
Web3D Consortium
john@carlsonsolutiondesign.com

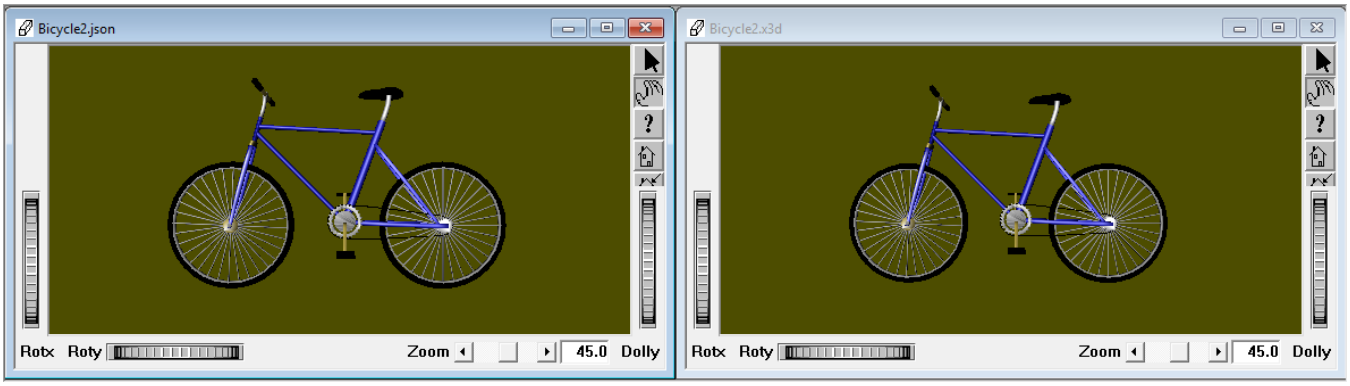

Figure 1: *Comparison of JSON (left) and XML (right) encoded bicycle model*

Abstract

X3D is a royalty-free openly published standard for 3D graphics, that has been ratified in a suite of ISO/IEC international standards. This paper reports on the development of a new standard for a JSON encoding.

The basic structures of the JSON language are summarized, and the mapping of the X3D abstract definitions to these structures detailed. The work on a JSON schema for validation of the X3D content is described, including some comparisons of the expressive power of the JSON and XML schemas which show that the JSON schema validation of the JSON encoding offers enhanced validation possibilities. Finally the early work on different implementations of the new encoding is presented, which confirm the overall success of the encoding.

Keywords: X3D, JSON, ISO/IEC standard, encoding

Concepts: • **General and reference~Computing standards, RFCs and guidelines** • *Computing methodologies~Virtual reality*

1 Introduction

Extensible 3D (X3D) is a royalty-free open standard for 3D graphics. A suite of international standards has been ratified by the International Standards Association (ISO), the first ISO/IEC standard being published in 2005. Active development is still continuing, with further standards being prepared. This paper reports on one of those, namely the introduction of a new fourth encoding, based on the popular JSON language.

The remainder of this paper is structure as follows. Sections 2 and 3 give an overview of X3D and JSON respectively, including a summary of the existing international standards for each. Section 4 details the JSON encoding, with comparisons to the existing XML encoding. Section 5 describes the development and testing processes used. Section 6 reports on validation and the generation of a JSON schema. Section 7 introduces implementations that are under development for the new encoding. Finally, section 8 concludes with a summary of further work.

2 X3D Overview

X3D is a royalty-free, open standard that defines both a file format specification and run-time architecture to represent and communicate 3D scenes, objects, events, behaviours and environments. A suite of International Standards Organization ratified standards has been developed and published. These standards provide a system for the storage, retrieval and playback of real time graphics content embedded in applications or web pages, all within an open architecture to support a wide range of domains and user scenarios.

X3D incorporates a modular design, with a rich set of componentized features that can be tailored for use in engineering and scientific visualization, CAD and architecture, geographical information systems, medical visualization, training and simulation, multimedia, entertainment, education, and more.

2.1 Architecture

Conceptually, an X3D application is a time-based 3D space that contains both graphic and aural objects. These objects can be loaded from predefined files in various formats and dynamically modified, or even dynamically created, through a variety of mechanisms.

Web3D '16, July 22 - 24, 2016, Anaheim, CA, USA

ACM 978-1-4503-4428-9/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2945292.2945304

The X3D system architecture is shown in Figure 2. An application, whether stand-alone or part of a web browser, for example, requires a number of components. These include parsers and loaders to read incoming files and/or streams, which may be in different formats, a scene graph manager to handle the resulting scene graph, including user defined prototypes, scripting engines to handle scripts defined within the scene, and an event manager to organize events that may arise both internally within the scene, or externally by user interaction or external applications.

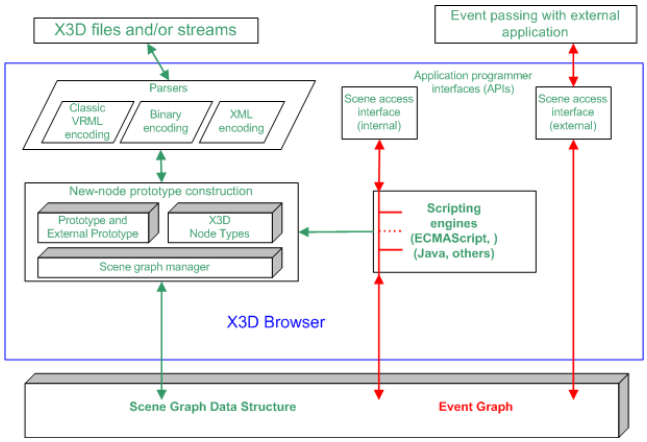

Figure 2: *X3D architecture*

2.2 ISO/IEC standards structure

The X3D suite of international standards is divided into three sets. These are the abstract structures and API, encodings, and language bindings. Daly and Brutzman [2000] present a detailed review which is summarized here.

The abstract structures and API are specified in ISO/IEC 19775, which has two parts, as follows:

- Extensible 3D (X3D) – Part 1: Architecture and base components [2013]
- Extensible 3D (X3D) – Part 2: Scene access interface (SAI) [2015]

Part 1 includes the definitions of 233 nodes as well as other statements and operational principles, all in a file format independent way. Part 2 covers the API to handle both internally and externally generated interactions, independently of any programming language.

The encodings set, ISO/IEC 19776, consists of three parts:

- Extensible 3D (X3D) Encodings – Part 1: Extensible Markup Language (XML) encoding [2015]
- Extensible 3D (X3D) Encodings – Part 2: Classic VRML encoding [2015]
- Extensible 3D (X3D) Encodings – Part 3: Compressed binary encoding [2015]

Each of these standards maps the abstract objects defined in Part 1 of ISO/IEC 19775 to a specific X3D encoding using a particular format.

The final set, language bindings, ISO/IEC 19777, consists of two parts:

- Extensible 3D (X3D) language bindings – Part 1: ECMAScript [2006]
- Extensible 3D (X3D) language bindings – Part 2: Java [2006]

Each of these standards maps the language independent API defined in Part 2 of ISO/IEC 19775 to a specific programming language.

This paper describes a new JSON encoding that is being prepared for submission as an additional part to ISO/IEC 19776.

3 JSON overview

JSON (JavaScript Object Notation) is an open standard format, first specified and published by Douglas Crockford. JSON is a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript programming language, but is programming language independent. There are JSON implementations in many different programming languages.

3.1 JSON standards

The specification of JSON is covered in a number of standards. The earliest was as a subset of JavaScript within the ECMA-262 [2015] standard. This has now been published as ISO/IEC 16262:2011 international standard. It describes both the JSON lexical and JSON syntactic grammars. There is also a separate ECMA-404 [2013] standard.

JSON was also originally published as RFC4627, but this is now obsolete and has been replaced by RFC7159 [BRAY 2014].

All these standards have identical grammars. The RFC references the ECMA documents, but contains additional material, such as security vulnerability considerations, for internet usage.

3.2 JSON structure

JSON defines a small set of rules for the portable representation of structured data. It uses human readable text to transmit data objects consisting of attribute-value pairs.

JSON has just two primary data structures: objects and arrays.

- Objects: A collection of name/value pairs, where the name of each pair should be unique within the collection.
- Arrays: An ordered list of values, each of which may be of any of the seven types.

JSON defines a value to be one of seven types. These are a string, a number, an object, an array, true, false, or null. The original web site [ANONYMOUS] provides a full description of the grammar in both graphical and text formats.

4 Encoding description

Aligning X3D scene graphs with the JSON data structures described above was the guiding principle of the design process. A further goal was that the JSON encoding should be "round trippable", i.e. starting from one encoding, say XML, translating to JSON, and then translating back to the original encoding should result in the original and resulting XML encodings having identical functionality.

4.1 Nodes

The primary construct of a scene graph are the nodes. In terms of the abstract specification 19775-1 these are any of the 233 X3D nodes defined therein, all of which are derived from the abstract type X3DNode. In the XML encoding nodes are defined as XML elements. In the JSON encoding nodes are defined as objects.

The name (i.e. node type) of a node in the XML encoding is in the XML tag. In JSON the node name is the name portion of the name/value pair. Taking an empty Group node as an example, the XML encoding would be

```
<Group/>
```

In JSON this becomes

```
"Group": {}
```

The name is expressed as a JSON string, which is always double quoted. The colon acts as the name value pair separator. Finally the left and right curly braces denote an object. Since the object is empty, all the attributes of the Group object assume their default values.

4.2 Fields

The attributes of X3D nodes are known as fields. They can be categorized in various ways, and the one of particular interest here is whether the field contains simple values, e.g. numbers or strings, or whether the fields hold references to other nodes. In the discussion below the terms for the two categories will be 'value field' and 'node field' respectively.

During development it became clear that it was necessary to make it easy to distinguish the names of fields from the names of nodes, since both appear as JSON strings. It was decided to prepend a non-alphabetic character to field names, using different characters for the two field categories. This makes it easy for parsers, loaders and validators to recognize the string as a field name, and identify any errors in usage.

4.2.1 Value fields

Value fields in JSON are prepended with the '@' symbol. In XML the value field is encoded as an attribute of the element node. On extending the previous empty Group example to include the *bboxSize* field the XML encoding would be:

```
<Group bboxSize='-1 -1 -1'/>
```

In JSON a field is encoded as a property of the object. The corresponding JSON encoding for this extended example becomes:

```
"Group": {
    "@bboxSize": [-1, -1, -1]
}
```

Since this particular field has an array of three values, the JSON encoding specifies the values in an array structure which is delineated by the square brackets and uses the comma to separate values.

4.2.2 Node fields

The distinguishing character for a node field is the '-' symbol. In XML node fields are not directly specified as attributes of the node, because their values are represented as children elements. So, the containing node field has to be separately specified using the 'containerField' attribute of the child node, to indicate which field of the parent node the child node belongs to. Extending the previous example to include the ViewPoint node as the only value in the *children* field of the Group node, the XML encoding becomes:

```
<Group bboxSize='-1 -1 -1'>
  <Viewpoint containerField='children'/>
</Group>
```

In JSON this slightly unnatural approach is not necessary. The node field is encoded, like a value field, as a property of the containing object. The node representing the value is then encoded as an object. So the equivalent JSON encoding becomes:

```
"Group": {
   "@bboxSize": [-1, -1, -1],
   "-children": [
      { "Viewpoint": {}
     }
   ]
}
```

As the *children* field can hold multiple child nodes the value(s) are encoded into an array. The JSON syntax permits array elements to be any of the seven types. Child nodes are therefore encoded as objects with the only object property being the type of the node.

Had the node field been one which only accepted a single node the node field would have been encoded as an object, without requiring an array. This object would have had a single property, whose name is the X3D node type. This can be illustrated using the Appearance node, which has a *material* field that accepts a single node. The JSON encoding would be:

```
"Appearance": {
   "-material": {
      "Material": {
        "diffuseColor": [0.7,0.4,0.1]
      }
   }
}
```

The equivalent XML encoding would be:

```
<Appearance>
  <Material containerField='material'
    diffuseColor='0.7 0.4 0.1'/>
</Appearance>
```

4.3 Comments

In JSON there is no specific provision for comments, unlike the other encoding formats. For example, in XML, comments can be included anywhere, and take the form

```
<!-- This is a comment -->
```

JSON must encode comments using the standard structures. This has been done using a name/value pair where the name is "#comment" and the value is a string. The above XML comment would be encoded in JSON as

```
"#comment": "This is a comment"
```

The positioning of comments within a JSON encoding, however, proved difficult. Consider the previous examples, with two comments included. In XML this might be:

```
<Group bboxSize='-1 -1 -1'>
  <!-- Before Viewpoint -->
  <Viewpoint containerField='children'/>
  <!-- After Viewpoint -->
</Group>
```

A first attempt at an equivalent JSON encoding produces:

```
"Group": {
   "@bboxSize": [-1, -1, -1],
   "#comment": "Before Viewpoint",
   "-children": [
      "Viewpoint": {}
   ],
   "#comment": "After Viewpoint"
}
```

The difficulty here is that, according to the JSON specifications, property names of JSON objects should be unique. The encoding was being designed to adhere to this principle, so the listing above could not be used. This was resolved by always placing comments in a "-children" field. The "-children" field, being an array is permitted to have multiple items with the same name. The encoding therefore becomes:

```
"Group": {
   "@bboxSize": [-1, -1, -1],
   "-children": [
      "#comment": "Before Viewpoint",
      "Viewpoint": {},
      "#comment": "After Viewpoint"
   ]
}
```

4.4 ROUTEs

X3D ROUTEs presented the same type of issues as comments, and were resolved in a similar way. An X3D ROUTE is encoded in XML as an element, similar to a node. Multiple ROUTEs can appear together, anywhere within a scene.

In JSON a ROUTE is encoded as an object, with the name "ROUTE", which has four properties "fromField", "fromNode", "toField", and "toNode". Like comments, ROUTEs are placed into the array value of a "-children" field.

4.5 Embedded source code

X3D Script and shader nodes can contain embedded source code. For XML these are encoded into a CDATA section. There is no similar provision in JSON to the XML CDATA. Therefore the encoding has to incorporate this into the standard JSON structures.

This was accomplished in the JSON encoding by using a name/value pair with the name "#sourceText" and the value as an array of strings, one string for each line of the CDATA text. The following short JSON encoding example illustrates this.

```
"Script": {
  "@DEF":"myScript",
  "#sourceText":[
    "ecmascript:",
    "// Include source code here ",
    "function anySFBool (val, timestamp)",
    "{",
    "\tsomeMFInt32 = 0; ",
    "}"
  ]
}
```

5 Development and testing

5.1 Example archives

One of the principle assets available when designing the encoding was the large examples archive held by the Web3D Consortium. The total number of examples was in excess of 3800. Their primary encoding is XML. They cover virtually all the nodes in the X3D standards. The first step was to automate the conversion of all of these examples from XML to JSON.

5.2 Stylesheet conversion

The automatic conversion of an example from XML to JSON was accomplished by using an XML to JSON stylesheet converter, using XSL version 2.0. The stylesheet was run in batch mode on every example in the archive. After each conversion, the resultant JSON output file was tested for conformance to JSON using JSLint, a well-known JavaScript quality assessment tool [CROCKFORD 2008].

This process highlighted two JSON specific issues. The first was character escaping in strings. XML and JSON have different character escaping requirements. Furthermore, in XML, CDATA sections require less escaping than in non-CDATA sections. Care was needed to ensure that XML character escaping was correctly identified and that JSON character escaping was used when required.

Source code or shader text were the most difficult to correctly identify character escaping. In XML such text is plain text within a CDATA section. However, on encoding into JSON the text is encoded into a string, which is delineated with double quotes. Any double quotes occurring in the XML text, which don't need to be escaped, do need to be escaped in JSON. For example, consider the following line extracted from a longer CDATA section in XML:

```
        sceneString='<X3D version="3.1"
profile="interchange">\n' +
```

The only escaped character is the line feed character towards the end. When this is converted to JSON, however, the double quotes also need to be escaped. So the correct JSON encoding is:

```
        "sceneString='<X3D version=\"3.1\"
profile=\"interchange\">\n' +",
```

The second issue was number representation. JSON only permits decimal numbers. In contrast, XML permits other formats, such as hexadecimal. The stylesheet converter must, therefore, ensure all numbers are decimal and convert them to decimal if not.

6 Validation

6.1 Schema development

The next step in the development process was to validate the resulting JSON encodings for consistency with the X3D standards. This is accomplished in XML using multiple techniques, with varying levels of expressive power. The simplest available for X3D is the document type definition (DTD). Then there is an XML schema, and finally, a Schematron, which has the most comprehensive validation capability.

For JSON, a schema was developed covering all X3D nodes. Automated schema generation tools were investigated but all were found to be unsuitable. The schema was manually designed using a tool with a graphical user interface. The final schema, which is over 17500 lines in length, can be viewed online or downloaded from http://www.web3d.org/specifications/x3d-3.3-JSONSchema.json.

Once complete, the schema was incorporated into the batch conversion process as an additional test on each JSON file produced.

The JSON schema was found to have more expressive power than the corresponding XML schema. For example, the JSON encoding separates child node content into the individual containing fields, whereas the XML encoding merges all the children into one group as child elements of the node, irrespective of the field. The XML schema, therefore, can only validate the combined child content, and often cannot be as strict as theoretically desired. JSON, on the other hand, can validate each individual node field's children. This enables stricter validation in JSON.

Another improvement in JSON validation concerns arrays. In JSON the value of each individual item can be validated independently. This is not the case with XML, where little validation is usually possible.

6.2 Schema specifications

At the time of writing there are no specifications covering JSON schema. Galiegue and Zyp [2013] submitted an IETF internet draft. Although this expired on August 4th 2013, it is still in regular use as the most recent draft. Independent work is currently proceeding to update this.

7 Implementations

The final stage of the practical development was

implementations covering loading and display of JSON encoded X3D scenes.

This stage also included document validation. This is important to ensure that security vulnerabilities, such as cross site scripting, are minimized, particularly when the JSON document, once loaded, may be handed off to other tools, e.g. rendering within an HTML document. Validation becomes part of the pipeline of handling X3D JSON.

For web browser testing two applications are available, which support incorporating XML encoded X3D into an HTML web page, are X3DOM and Cobweb. A JSON loader has been successfully developed for these, albeit with the restrictions of the two applications.

X3DOM, for example, does not support prototypes. However, an additional JSON prototype expander is in development to overcome this.

A second implementation in C++ is also under development. This is a standalone application that can already display XML encoded scenes. A loader for JSON has been successfully added. Figure 1 shows the same bicycle model encoded both as JSON and as XML, with JSON on the left and XML on the right. The original XML encoding was converted to JSON using the tools described earlier.

8 Further work

Over the years four versions of X3D have been standardized, with improvements and additions in the later versions. There are XML schemas for each version. Work is ongoing to automate the generation of JSON schemas for each individual version, starting from the XML schemas, using an XML encoded object model as an intermediate step. Work is also continuing on all the implementations mentioned above.

Drafting of a new encoding specification is in progress. On completion it will be submitted to ISO/IEC for ratification as part 5 of the IEC/ISO standard 19776.

Future work may also consider utilization of the Efficient XML Interchange (EXI) for JSON, currently being drafted by the W3C [2016], as a means for simultaneously optimizing performance and the utilization of computational resources.

References

ANONYMOUS. Introducing JSON. *http://www.json.org/*

BRAY, T. 2014. The JavaScript Object Notation (JSON) Data Interchange Format. *http://www.rfc-editor.org/rfc/rfc7159.txt*

CROCKFORD, D. 2008. *Javascript: The Good Parts*. O'Reilly Media.

DALY, L., AND BRUTZMAN, D. 2000. X3D: Extensible 3D Graphics Standard. *IEEE Signal Processing Magazine* (Nov), 130-135.

ECMA-262 2015. ECMAScript 2015 Language Specification. *http://www.ecma-international.org/publications/standards/Ecma-262.htm*

ECMA-404 2013. The JSON Data Interchange Format. *http://www.ecma-international.org/publications/standards/Ecma-404.htm*

GALIEGUE, F. AND ZYP, K. 2013. JSON Schema: core definitions and terminology draft-zyp-json-schema-04. *Internet Engineering Task Force Internet Draft.*

ISO/IEC 19775-1:2013. Information Technology – Computer graphics, image processing and environmental data representation – Extensible 3D (X3D) – Part 1: Architecture and base components

ISO/IEC 19775-2:2015. Information Technology – Computer graphics, image processing and environmental data representation – Extensible 3D (X3D) – Part 2: Scene access interface (SAI)

ISO/IEC 19775-2:2015. Information Technology – Computer graphics, image processing and environmental data representation – Extensible 3D (X3D) – Part 2: Scene access interface (SAI)

ISO/IEC 19776-1:2015. Information Technology – Computer graphics, image processing and environmental data representation – Extensible 3D (X3D) encodings – Part 1: Extensible Markup Language (XML) encoding

ISO/IEC 19776-2:2015. Information Technology – Computer graphics, image processing and environmental data representation – Extensible 3D (X3D) encodings – Part 2: Classic VRML encoding

ISO/IEC 19776-3:2015. Information Technology – Computer graphics, image processing and environmental data representation – Extensible 3D (X3D) encodings – Part 3: Compressed binary encoding

ISO/IEC 19777-1:2006. Information Technology – Computer graphics and image processing – Extensible 3D (X3D) language bindings – Part 1: ECMAScript

ISO/IEC 19777-2:2006. Information Technology – Computer graphics and image processing – Extensible 3D (X3D) language bindings – Part 2: Java

W3C. Efficient XML Interchange (EXI) for JSON. W3C First Public Working Draft 28 January 2016. http://www.w3.org/TR/exi-for-json