EXI for JSON (EXI4JSON)

W3C Working Group Note 26 July 2018

This version:
https://www.w3.org/TR/2018/NOTE-exi-for-json-20180726/

Latest version:
https://www.w3.org/TR/exi-for-json/

Previous version:
https://www.w3.org/TR/2016/WD-exi-for-json-20160823/

Editors:
Daniel Peintner, Siemens AG
Don Brutzman, Web3D Consortium

Abstract

The EXI format is a compact representation that simultaneously optimizes performance and the utilization of computational resources. The EXI format was designed to support XML representation. With a relatively small set of transformations it may also be used for JSON, a popular format for exchange of structured data on the Web.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

This document has been produced by the Efficient Extensible Interchange (EXI) Working Group. The goals of the EXI Format are discussed in the EXI Format document.

Publication as a Working Group Note does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Please send comments about this document to the public-exi@w3.org mailing list (Archives).

This document was produced by a group operating under the W3C Patent Policy.

This document is governed by the 1 February 2018 W3C Process Document.

Table of Contents

✧ ✧ ✧

1. Introduction

JSON is a popular format for exchange of structured data on the Web and it is specified in [RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format] and [ECMA-404 - The JSON Data Interchange Format]. This document describes how the [Efficient XML Interchange (EXI) Format 1.0 (Second Edition)] specification can be used to represent JSON data efficiently in terms of message size and processing.

2. Concept

The EXI for JSON approach is to equivalently convert any well-formed JSON structures to XML event streams (Appendix D Examples shows some examples) that are directly suitable for datatype-aware EXI representation. Lossless round-trip conversion back to the original JSON structures is supported.

The proposed XML event stream results in a compact format — the so-called EXI for JSON (or EXI4JSON) document — that can be read and written with little additional software. That said, Appendix B XML Schema for EXI4JSON provides an XML Schema describing the EXI for JSON document. EXI processors use the schema-informed grammars that stem from this schema.

The EXI Options describe the EXI options that may be used for any EXI document. Negotiation of what options need to be supported by an EXI for JSON implementation are handled externally to the document. This specification makes use of the default options with the following modifications:

Table 2-1. Predefined EXI4JSON EXI Options

EXI Option	Description	Value
strict	Strict interpretation of schemas is used to achieve better compactness	true
schemaId	Identify the schema information, if any, used to encode the body	"exi4json"

Both EXI Options for strict and schemaId are REQUIRED and cannot be changed. If future versions of EXI for JSON are specified, version identification is reflected in the schemaId value.

3. Representing JSON data using EXI

Any valid JSON data can be converted to equivalent EXI. Similarly, corresponding EXI streams that conform to the rules and schema of this specification can be converted to equivalent JSON. The following subsections specify how JSON data MUST be represented for equivalent round-trip conversion. This approach is not suitable for arbitrary EXI or XML data.

Prefixes are used throughout this section to designate certain namespaces. The bindings shown below are assumed, however, any prefixes can be used in practice if they are properly bound to the namespaces.

Prefix	Namespace Name
j	http://www.w3.org/2015/EXI/json

Also, the specification makes use of EXI event terminology and the associated grammar notation (e.g., `SE` stands for Start Element and `EE` for End Element) that is fully described in the EXI specification dealing with EXI Event Types.

A JSON value is an `object`, `array`, `string`, or `number`, or one of the following three literal names: `true false null`.

3.1 JSON `object`

A JSON `object` is represented as a `j:map` element which may comprise a set of key/value pairs as it content.

```
SE(j:map) <!--key/value pairs--> EE
```

The XML event sequence for a key/value pair is

```
SE(j:key) <!--value--> EE
```

> Note:
>
> If the ***key***-name is not valid with respect to NCName or it conflicts with any existing global element name in the XML schema (e.g., `array` or `string`) the `key` part MUST be escaped as subsequently described in 3.1.1 Key-name Escaping.
>
> That said, any escaped character MUST be unescaped to get back the original JSON ***key***-name (see D.3 Example 3).

3.1.1 Key-name Escaping

We distinguish two types of escaping:

1. Conflict with NCName character(s)

   > Any character CharRef that is not valid in XML names for use within XML names MUST be escaped as follows
   >
   > ```
   > CharRef := '_' [0-9]+ '.'
   > ```
   >
   > The digits after '_' up to the terminating '.' provide the decimal number of the character's code point.
   > (e.g., JSON key `"1 key"` becomes `"_49._32.key"`)
   >
   > > Note:
   > >
   > > In order to represent '_' itself, the underscore character needs to be written as `"_95."` (with period character included) for unambiguous parsing.

2. Conflict with existing EXI4JSON global schema element name

> If the *key*-name is one of the reserved EXI4JSON schema element names (`map`, `array`, `string`, `number`, `boolean`, `null`, or `other`), then the `key` MUST be prefixed with the following character sequence *"_."*
> (e.g., JSON key *"map"* becomes *"_.map"*)

3.2 JSON `array`

A JSON `array` is represented as a `j:array` element which may comprise a collection of values.

```
SE(j:array) <!--values--> EE
```

3.3 JSON `string`

A JSON `string` MAY be represented as a `j:string` element.

```
SE(j:string) CH(string-value) EE
```

The EXI for JSON transformation rules allow to map a string also to one of the following more-optimized XML event sequences

- `SE(j:other) SE(j:base64Binary) CH(string-value) EE EE`
- `SE(j:other) SE(j:dateTime) CH(string-value) EE EE`
- `SE(j:other) SE(j:time) CH(string-value) EE EE`
- `SE(j:other) SE(j:date) CH(string-value) EE EE`

> Note:
>
> The above mentioned choice requires that the string-value is representable by the corresponding EXI datatype.

3.4 JSON `number`

A JSON `number` MAY be represented as a `j:number` element.

```
SE(j:number) CH(number-value) EE
```

The EXI for JSON transformation rules allow to map a number also to one of the following more-optimized XML event sequences

- `SE(j:other) SE(j:integer) CH(number-value) EE EE`
- `SE(j:other) SE(j:decimal) CH(number-value) EE EE`

> Note:
>
> The above mentioned choice requires that the number-value is representable by the corresponding EXI datatype.

3.5 JSON `true`

A JSON `true` is represented as a `j:boolean` element with the Characters (CH) event content equals `"true"`.

```
SE(j:boolean) CH("true") EE
```

3.6 JSON `false`

A JSON `false` is represented as a `j:boolean` element with the Characters (CH) event content equals `"false"`.

```
SE(j:boolean) CH("false") EE
```

3.7 JSON `null`

A JSON `null` is represented as an empty `j:null` element.

```
SE(j:null) EE
```

A References

Efficient XML Interchange (EXI) Format 1.0 (Second Edition)
Efficient XML Interchange (EXI) Format 1.0 (Second Edition), John Schneider, Takuki Kamiya, Daniel Peintner, Rumen Kyusakov, Editors. World Wide Web Consortium. The latest version is available at https://www.w3.org/TR/exi/. (See https://www.w3.org/TR/2014/REC-exi-20140211/.)

RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format
The JavaScript Object Notation (JSON) Data Interchange Format, T. Bray, Editor. Internet Engineering Task Force (IETF), Request for Comments: 7159. Available at https://tools.ietf.org/html/rfc7159 (See https://tools.ietf.org/html/rfc7159.)

ECMA-404 - The JSON Data Interchange Format
The JSON Data Interchange Format, ECMA Standard ECMA-404, first edition, October 2013. Available at http://www.ecma-international.org/publications/standards/Ecma-404.htm (See http://www.ecma-international.org/publications/standards/Ecma-404.htm.)

B XML Schema for EXI4JSON

The following XML schema describes the EXI4JSON document (see also exi4json.xsd).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    targetNamespace="http://www.w3.org/2015/EXI/json" xmlns:j="http://www.w3.org/2015/EXI/json">

    <!--
     * This is a schema for the XML representation of JSON
     *
     * The schema is made available under the terms of the W3C software notice and license
     * at https://www.w3.org/Consortium/Legal/copyright-software-19980720
     *
    -->

    <xs:element name="map" type="j:mapType"/>

    <xs:element name="array" type="j:arrayType"/>

    <xs:element name="string" type="j:stringType"/>

    <xs:element name="number" type="j:numberType"/>

    <xs:element name="boolean" type="j:booleanType"/>

    <xs:element name="null" type="j:nullType"/>

    <xs:element name="other" type="j:otherType"/>


    <xs:complexType name="mapType">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <!-- any element is a map key which contains the actual value  -->
            <!-- "key": 25 -->
            <!--
```

```
                <age>
                    <number>25</number>
                </age>
            -->
            <xs:any processContents="lax" namespace="##targetNamespace"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="arrayType">
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="j:map"/>
            <xs:element ref="j:array"/>
            <xs:element ref="j:string"/>
            <xs:element ref="j:number"/>
            <xs:element ref="j:boolean"/>
            <xs:element ref="j:null"/>
            <xs:element ref="j:other"/>
        </xs:choice>
    </xs:complexType>

    <xs:simpleType name="stringType">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>

    <xs:simpleType name="numberType">
        <xs:restriction base="xs:double">
            <!-- exclude positive and negative infinity, and NaN -->
            <!-- Note: No real effect for EXI Float datatype -->
            <xs:minExclusive value="-INF"/>
            <xs:maxExclusive value="INF"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="booleanType">
        <xs:restriction base="xs:boolean"/>
    </xs:simpleType>

    <xs:complexType name="nullType"/>

    <xs:complexType name="otherType">
        <xs:choice>
            <!-- useful types beyond JSON such as binary, date-times, decimal and integer -->
            <xs:element name="base64Binary">
                <xs:simpleType>
                    <xs:restriction base="xs:base64Binary"/>
                </xs:simpleType>
            </xs:element>
            <xs:element name="dateTime">
                <xs:simpleType>
                    <xs:restriction base="xs:dateTime"/>
                </xs:simpleType>
            </xs:element>
            <xs:element name="time">
                <xs:simpleType>
                    <xs:restriction base="xs:time"/>
                </xs:simpleType>
            </xs:element>
            <xs:element name="date">
                <xs:simpleType>
                    <xs:restriction base="xs:date"/>
                </xs:simpleType>
            </xs:element>
            <xs:element name="integer">
                <xs:simpleType>
                    <xs:restriction base="xs:integer"/>
                </xs:simpleType>
            </xs:element>
            <xs:element name="decimal">
                <xs:simpleType>
                    <xs:restriction base="xs:decimal"/>
                </xs:simpleType>
            </xs:element>
        </xs:choice>
```

```
        </xs:choice>
    </xs:complexType>

</xs:schema>
```

C Design Decisions (Non-Normative)

This section discusses a number of key decision points. A rationale for each decision is given and background information is provided.

C.1 Selection of `other` Datatype Representations

Compared to the basic JSON datatypes and the corresponding EXI datatype mapping (i.e., exi:string, exi:double, and exi:boolean) the element `other` allows for other EXI datatype representations: namely exi:base64Binary, exi:dateTime, exi:dateTime, exi:time, exi:date, exi:integer, and exi:decimal.

The selection of these additional datatypes is based on their foreseen efficiency and potential usage in JSON documents.

C.2 Character Encoding

JSON text may be encoded in UTF-8, UTF-16, or UTF-32 (see JSON Character Encoding). EXI for JSON matches the JSON specification in that it does not provide an explicit label for the included characters.

If possible without loss of correctness, processors are recommended to use the default UTF-8 for maximum interoperability when creating JSON documents.

C.3 Selection of EXI options

EXI for JSON defines a set of predefined EXI Options beyond the default EXI Options.

C.3.1 EXI Option strict

The default EXI value for strict is false to permit event items not declared in the schemas.

The main reason to set strict to true in the EXI for JSON context is to reduce specification and code complexity while at the same time allowing for simple implementations. In section 3. Representing JSON data using EXI it is specified how to map an EXI4JSON stream to JSON. Allowing strict to be false would require to deal with unexpected elements and/or attributes and would make the specification more complex while at the same time increase code complexity. The working group concluded that strict being false does not provide any benefit in this context.

Besides that strict being true increases compactness and allows for realizing more-optimized processors with less code.

C.3.2 EXI Option schemaId

The schemaId is used to identify the schema information used for processing the EXI stream. The value "exi4json" has been chosen to identify the schema in Appendix B XML Schema for EXI4JSON.

C.4 Change of Structure

The EXI for JSON structure has been changed compared to the previous publication. A JSON key is not represented anymore as attribute. Instead it is transformed to an element with the JSON value as nested element.

The reason for this change is to allow for dedicated XML schema definitions (which were not possible before). This change implied escaping (see 3.1.1 Key-name Escaping) with the positive side effect

to generate valid XML streams.

> Note:
>
> EXI streams would not need escaping.

D Examples (Non-Normative)

D.1 Example 1

This example illustrates a simple JSON document with a numbered value and an array of strings.

JSON

```
{
  "keyNumber": 123,
  "keyArrayStrings": [
    "s1",
    "s2"
  ]
}
```

↔

EXI4JSON

```
<j:map xmlns:j="http://www.w3.org/2015/EXI/json">
  <j:keyNumber>
    <j:number>123</j:number>
  </j:keyNumber>
  <j:keyArrayStrings>
    <j:array>
      <j:string>s1</j:string>
      <j:string>s2</j:string>
    </j:array>
  </j:keyArrayStrings>
</j:map>
```

D.2 Example 2

This is example illustrates nested JSON values such as objects and arrays.

JSON

```
{
  "glossary": {
    "title": "example glossary",
    "GlossDiv": {
      "title": "S",
      "GlossList": {
        "GlossEntry": {
          "ID": "SGML",
          "SortAs": "SGML",
          "GlossTerm": "Standard Generalized Markup Language",
          "Acronym": "SGML",
          "Abbrev": "ISO 8879:1986",
          "GlossDef": {
            "para": "A meta-markup language,
              used to create markup languages such as DocBook.",
            "GlossSeeAlso": [
              "GML",
              "XML"
            ]
          },
          "GlossSee": "markup"
        }
      }
    }
  }
}
```

↔

EXI4JSON

```
<j:map xmlns:j="http://www.w3.org/2015/EXI/
  <j:glossary>
    <j:map>
      <j:title>
        <j:string>example glossary</j:strin
      </j:title>
      <j:GlossDiv>
        <j:map>
          <j:title>
            <j:string>S</j:string>
          </j:title>
          <j:GlossList>
            <j:map>
              <j:GlossEntry>
                <j:map>
                  <j:ID>
                    <j:string>SGML</j:strin
                  </j:ID>
                  <j:SortAs>
                    <j:string>SGML</j:strin
                  </j:SortAs>
                  <j:GlossTerm>
                    <j:string>Standard Gene
                  </j:GlossTerm>
                  <j:Acronym>
                    <j:string>SGML</j:strin
                  </j:Acronym>
                  <j:Abbrev>
                    <j:string>ISO 8879:1986
                  </j:Abbrev>
                  <j:GlossDef>
                    <j:map>
                      <j:para>
```

```
                        <j:string>A meta-ma
                          markup languages
                      </j:para>
                      <j:GlossSeeAlso>
                        <j:array>
                          <j:string>GML</j:
                          <j:string>XML</j:
                        </j:array>
                      </j:GlossSeeAlso>
                    </j:map>
                  </j:GlossDef>
                  <j:GlossSee>
                    <j:string>markup</j:str
                  </j:GlossSee>
                </j:map>
              </j:GlossEntry>
            </j:map>
          </j:GlossList>
        </j:map>
      </j:GlossDiv>
    </j:map>
  </j:glossary>
</j:map>
```

D.3 Example 3

This is an example with a keyname `"a number"` which is not valid with respect to NCName and uses 3.1.1 Key-name Escaping.

JSON

```
{
  "a number": 1
}
```

↔

EXI4JSON

```
<j:map xmlns:j="http://www.w3.org/2015/EXI/json">
  <j:a_32.number>
    <j:number>1</j:number>
  </j:a_32.number>
</j:map>
```

E Schema Mapping (Non-Normative)

This section describes best practices in how to map knowledge of JSON data into dedicated XML schema definitions. Dedicated XML schema definitions (compared to the pre-defined "exi4json" schemaId specified in this document) allow for an even more efficient representation by keeping compatibility on a structural basis.

Note: JSON schema is used in the examples but the practices are not limited to JSON schema language.

Editorial note

Add information about JSON schema "additionalProperties": false/true

E.1 JSON object

A JSON object is mapped to an `element` with complexType and the name attribute set to `map`. The child elements appear in `all` so that they can appear in any order. The attribute `minOccurs` is set to "0" if a child element is not required.

JSON schema

```
{
  "type": "object",
  "properties": {
    "firstname": {
      "type": "string"
    },
```

→

XML schema definition

```
<element name="map">
    <complexType>
        <all>
            <element minOccurs="0" name="firstname">
                <complexType>
                    <sequence>
```

```
    "lastname": {
      "type": "string"
    },
    "age": {
      "type": "number"
    }
  },
  "required": [
    "lastname",
    "age"
  ]
}
```

```
                        <!--unnamed-format-->
                        <element name="string" type="string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="age">
                <complexType>
                    <sequence>
                        <element name="number" type="double"/>
                    </sequence>
                </complexType>
            </element>
            <element name="lastname">
                <complexType>
                    <sequence>
                        <!--unnamed-format-->
                        <element name="string" type="string"/>
                    </sequence>
                </complexType>
            </element>
        </all>
    </complexType>
</element>
```

E.2 JSON array

A JSON array is mapped to an `element` with complexType and the name attribute set to `array`. The child elements appear in a `sequence` and the attribute `minOccurs` is set to "0" and `maxOccurs` to "unbounded". The child elements itself depend on the items type.

JSON schema

```
{
  "type": "array",
  "items": {
    "type": "number"
  }
}
```

→ XML schema definition

```
<element name="array">
    <complexType>
        <sequence maxOccurs="unbounded" minOccurs="0">
            <element name="number" type="double"/>
        </sequence>
    </complexType>
</element>
```

E.3 JSON string

A JSON string is mapped to an element with the name attribute set to `string` and typed as string.

JSON schema

```
{
  "type": "string"
}
```

→ XML schema definition

```
<element name="string" type="string"/>
```

E.4 JSON number

A JSON number is mapped to an element with the name attribute set to `number` and typed as double.

Editorial note

Differentiate between "number" and "integer".

JSON schema

```
{
  "type": "number"
}
```

→ XML schema definition

```
<element name="number" type="double"/>
```

E.5 JSON boolean

A JSON boolean is mapped to an element with the name attribute set to `boolean` and typed as boolean.

JSON schema

```
{
  "type": "boolean"
}
```

→

XML schema definition

```
<element name="boolean" type="boolean"/>
```

E.6 JSON null

A JSON null is mapped to an element with the name attribute set to `null` and typed as empty complexType.

JSON schema

```
{
  "type": "null"
}
```

→

XML schema definition

```
<element name="null">
    <complexType/>
</element>
```

E.7 JSON enumeration

A JSON enumeration is mapped to an element based on the desired type.

JSON schema

```
{
  "type": "string",
  "enum": ["red", "amber", "green"]
}
```

→

XML schema definition

```
<element name="string">
    <simpleType>
        <restriction base="string">
            <enumeration value="red"/>
            <enumeration value="green"/>
            <enumeration value="amber"/>
        </restriction>
    </simpleType>
</element>
```

F Acknowledgements (Non-Normative)

This document is the work of the Efficient XML Interchange (EXI) WG.

Members of the Working Group are (at the time of writing, sorted alphabetically by last name):

- Carine Bournez, W3C/ERCIM (staff contact)
- Don Brutzman, Web3D Consortium
- Michael Cokus, MITRE Corporation
- Fernández, Javier D., Vienna University of Economics and Business
- Joerg Heuer, Siemens AG
- Sebastian Käbisch, Siemens AG
- Takuki Kamiya, Fujitsu Laboratories of America (chair)
- Richard Kuntschke, Siemens AG

- Matsukura, Ryuichi, Fujitsu Laboratories
- Don McGregor, Web3D Consortium
- Daniel Peintner, Siemens AG
- Liam Quin, W3C/MIT
- Mohamed Zergaoui, INNOVIMAX

The EXI Working Group would like to acknowledge the following former members or external experts for their leadership, guidance and expertise they provided throughout the process of creating this document (sorted alphabetically by last name):

- Yusuke Doi
- Youenn Fablet
- Bruce Hill
- Rumen Kyusakov
- Stephen Williams
- Brett Zamir

The EXI Working Group especially acknowledges and thanks Liam Quin (W3C/MIT) for his guidance and insight throughout the development of the EXI family of recommendations.