XEP-0295: JSON Encodings for XMPP

Kevin Smith
mailto:kevin@kismith.co.uk
xmpp:kevin@doomsong.co.uk

Matthew Wild
mailto:mwild1@gmail.com
xmpp:me@matthewwild.co.uk

2011-04-01
Version 1.0

Status	Type	Short Name
Active	Humorous	N/A

This specification defines an alternative JSON encoding for XMPP stanzas and other elements.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2024 by the XMPP Standards Foundation (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy> or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1 Introduction

It has long been known that XML is an outdated, failed format for interchangeable data serialization. While it does, admittedly, provide all the features that XMPP needs, XML is not without its share of detractors. Indeed, some years ago this led to the (sadly short-lived) attempt to provide a binary encoding for XMPP given in Binary XMPP (XEP-0239) [1]. Unfortunately, the binary encoding lacked the main advantages of XML in its human readability, so the search for better encodings has led us to JSON. JSON is a very popular format and it is sensible to utilize this popularity by reframing XMPP stanzas in JSON. JSON is not as expressive as XML in terms of namespacing, so this document presents a method of encoding stanzas, including namespaces in JSON.

2 Protocol

The recently updated XMPP Core [2] documents the legacy XML encoding of XMPP, and readers are urged to refer to that spec not just for other details of the protocol but also to appreciate the relative elegance of the encoding contained within this extension.
Let us consider a fairly standard XMPP message stanza:

Listing 1: XML-encoded XMPP stanza

```xml
<message to="alice@wonderland.lit" id="if00lu" >
  <body>Hi you</body>
  <body xml:lang="cy">Prynhawn da</body>
  <nick xmlns="http://jabber.org/protocol/nick">Alice</nick>
  <active xmlns="http://jabber.org/protocol/chatstates"/>
</message>
```

Given the need to include the namespaces within the JSON, an immediately obvious structure may be something like::

Listing 2: Naïve JSON representation

```json
{"stanza-name": "message",
 "stanza-namespace": "jabber:client",
 "stanza-attributes": {
  "to":"alice@wonderland.lit",
  "id":"if00lu"
 }
 "stanza-children": [
  {
```

---

[1]XEP-0239: Binary XMPP <https://xmpp.org/extensions/xep-0239.html>.

[2]RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

```
        "element-name": "body",
        "element-namespace": "jabber:client",
        "element-value": "Hi you"
  },
  {
        "element-name": "body",
        "element-namespace": "jabber:client",
        "element-language": "cy",
        "element-value": "Prynhawn da"
  },
  {
        "element-name": "nick",
        "element-namespace": "http://jabber.org/protocol/nick",
        "element-value": "Alice"
  },
  {

        "element-name": "active",
        "element-namespace": "http://jabber.org/protocol/chatstates",
  }
 ]
}
```

While many of the advantages of JSON over XML can be observed in this encoding (particularly the inherent brevity), an even more compact representation has been developed. Instead of reserializing the traditional XML stanzas in this manner, it is possible to wrap the stanzas within JSON, thereby enjoying the best of both worlds:

Listing 3: Advanced JSON-encoding

```
{"s":"<message to='alice@wonderland.lit' id='if00lu'><body>Hi you</
    body><body xml:lang='cy'>Prynhawn da</body><nick xmlns='http://
    jabber.org/protocol/nick'>Alice</nick><active xmlns='http://jabber
    .org/protocol/chatstates'/></message>"}
```

To use this improved encoding (eminently suitable both for c2s and s2s connections), entities should follow the connection rules defined in XMPP Core [3] and immediately start sending JSON-encoded data. Receiving entities should detect the presence of an open-brace ('{') character as the first octet received on a stream to be a signal to continue with JSON encoding. Servers supporting only the legacy XML encoding will necessarily respond with an error when receiving the improved JSON format, and entities will know to reconnect and continue with the legacy format.

---

[3]RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

3 Examples

Hopefully the beauty of this approach will be apparent at this stage, but in case some lingering doubts remain (and with the hope of aiding interoperability), more examples are provided here:

Listing 4: XML-encoded Message with Security Label

```
<message to='romeo@example.net' from='juliet@example.com/balcony'>
    <body>This content is classified.</body>
    <securitylabel xmlns='urn:xmpp:sec-label:0'>
        <displaymarking fgcolor='black' bgcolor='red'>SECRET</
            displaymarking>
        <label><esssecuritylabel xmlns='urn:xmpp:sec-label:ess:0'>
            MQYCAQQGASk=
        </esssecuritylabel></label>
    </securitylabel>
</message>
```

Listing 5: JSON-encoded Message with Security Label

```
{"s":"<message to='romeo@example.net' from='juliet@example.com/balcony
    '><body>This content is classified.</body><securitylabel xmlns='
    urn:xmpp:sec-label:0'><displaymarking fgcolor='black' bgcolor='red
    '>SECRET</displaymarking><label><esssecuritylabel xmlns='
    urn:xmpp:sec-label:ess:0'>MQYCAQQGASk=</esssecuritylabel></label
    ></securitylabel></message>"}
```

Listing 6: XML-encoded XHTML-IM message

```
<message from='ladymacbeth@shakespeare.lit/castle'
         to='macbeth@chat.shakespeare.lit'
         type='groupchat'>
  <body>Yet here's a spot.</body>
  <html xmlns='http://jabber.org/protocol/xhtml-im'>
    <body xmlns='http://www.w3.org/1999/xhtml'>
      <p>
        Yet here's a spot.
        <img alt='A spot'
             src='cid:sha1+8
                 f35fef110ffc5df08d579a50083ff9308fb6242@bob.xmpp.org'
                 />
      </p>
    </body>
  </html>
</message>
```

Listing 7: JSON-encoded XHTML-IM message

```
{"s":"<message␣from='ladymacbeth@shakespeare.lit/castle'␣to='
    macbeth@chat.shakespeare.lit'␣type='groupchat'><body>Yet␣here's␣a␣
    spot.</body><html␣xmlns='http://jabber.org/protocol/xhtml-im'><
    body␣xmlns='http://www.w3.org/1999/xhtml'><p>Yet␣here's␣a␣spot.<
    img␣alt='A␣spot'␣src='cid:sha1+8
    f35fef110ffc5df08d579a50083ff9308fb6242@bob.xmpp.org'/></p></body
    ></html></message>"}
```

Listing 8: XML-encoded Dialback Key Transmission

```
<db:result
    from='sender.tld'
    to='target.tld'>
  1e701f120f66824b57303384e83b51feba858024fd2221d39f7acc52dcf767a9
</db:result>
```

Listing 9: JSON-encoded Dialback Key Transmission

```
{"s":"<db:result␣from='sender.tld'␣to='target.tld'>1
    e701f120f66824b57303384e83b51feba858024fd2221d39f7acc52dcf767a9</
    db:result>"}
```

Listing 10: XML-encoded Event Publication

```
<iq from='juliet@capulet.lit/balcony' type='set' id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='http://jabber.org/protocol/tune'>
      <item>
        <tune xmlns='http://jabber.org/protocol/tune'>
          <artist>Gerald Finzi</artist>
          <length>255</length>
          <source>Music for 'Love's Labors Lost'␣(Suite␣for␣small␣
              orchestra)</source>
          <title>Introduction␣(Allegro␣vigoroso)</title>
          <track>1</track>
        </tune>
      </item>
    </publish>
  </pubsub>
</iq>
```

Listing 11: JSON-encoded Event Publication

```
{"s":"<iq␣from='juliet@capulet.lit/balcony'␣type='set'␣id='pub1'><
    pubsub␣xmlns='http://jabber.org/protocol/pubsub'><publish␣node='
    http://jabber.org/protocol/tune'><item><tune␣xmlns='http://jabber.
    org/protocol/tune'><artist>Gerald␣Finzi</artist><length>255</
    length><source>Music␣for␣'Love's␣Labors␣Lost'␣(Suite␣for␣small␣
    orchestra)</source><title>Introduction␣(Allegro␣vigoroso)</title><
    track>1</track></tune></item></publish></pubsub></iq>"}
```

Listing 12: XML-encoded Stream Header

```
<stream:stream
      from='juliet@example.com'
      to='example.com'
      version='1.0'
      xml:lang='en'
      xmlns='jabber:client'
      xmlns:stream='http://etherx.jabber.org/streams'>
```

Listing 13: JSON-encoded Stream Header

```
{"s":"<stream:stream␣from='juliet@example.com'␣to='example.com'␣
    version='1.0'␣xml:lang='en'␣xmlns='jabber:client'␣xmlns:stream='
    http://etherx.jabber.org/streams'>"}
```

Beautiful, elegant *and* efficient at the same time.

4 Internationalization Considerations

It is hoped that this representation introduces no new internationalization considerations, although it is acknowledged that if there are cultures where the symbols {}: are considered to be more offensive than <>= the legacy XML encoding may be preferred.

5 Security Considerations

Implementors should be aware that the JSON encoding involves 8 additional bytes for each stanza, and this introduces a considerable risk of buffer over-flow attacks. While new code-bases will hopefully be designed with this in mind, existing codebases will need to be entirely upgraded, with every buffer increased in size by at least 8 bytes to address this potentially serious potential vulnerability.

6 IANA Considerations

None.

7 XMPP Registrar Considerations

The XMPP Registrar may wish to consider maintenance of dual registries - for both XML and JSON encodings, but this is OPTIONAL.

8 JSON Schema

```
{
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "s": {
    "type": "string",
    "required": true,
    "format": "xml"
    }
  }
}
```

9 Acknowledgements