XMPP Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 14, 2015

L. Stout, Ed.
&yet
J. Moffitt
Mozilla
E. Cestari
cstar industries

September 10, 2014

An XMPP Sub-protocol for WebSocket
draft-ietf-xmpp-websocket-10

Abstract

This document defines a binding for the XMPP protocol over a WebSocket transport layer. A WebSocket binding for XMPP provides higher performance than the current HTTP binding for XMPP.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

To date, applications using the Extensible Messaging and Presence Protocol (XMPP) (see [RFC6120] and [RFC6121]) on the Web have made use of BOSH (see [XEP-0124] and [XEP-0206]), an XMPP binding to HTTP. BOSH is based on the HTTP long polling technique, and it suffers from high transport overhead compared to XMPP's native binding to TCP. In addition, there are a number of other known issues with long polling [RFC6202], which have an impact on BOSH-based systems.

It would be much better in most circumstances to avoid tunneling XMPP over HTTP long polled connections and instead use the XMPP protocol directly. However, the APIs and sandbox that browsers have provided do not allow this. The WebSocket protocol [RFC6455] exists to solve

these kinds of problems and is a bidirectional protocol that provides a simple message-based framing layer, allowing for more robust and efficient communication in web applications.

The WebSocket protocol enables two-way communication between a client and a server, effectively emulating TCP at the application layer and therefore overcoming many of the problems with existing long-polling techniques for bidirectional HTTP. This document defines a WebSocket sub-protocol for XMPP.

The WebSocket binding for XMPP is designed for use by browser-based applications (e.g., XMPP clients written in JavaScript). These applications typically are used to access the same information and communication opportunities (e.g., the same XMPP "roster" of contacts) as clients that access connect to an XMPP server over the TCP binding defined in [RFC6120]. Although the only essential difference is the underlying transport binding, relevant implications (e.g., framing methods and discovery processes) are highlighted in this specification.

2. Terminology

The basic unit of framing in the WebSocket protocol is called a message. In XMPP, the basic unit is the stanza, which is a subset of the first-level children of each document in an XMPP stream (see Section 9 of [RFC6120]). XMPP also has a concept of messages, which are stanzas with a top-level element of <message/>. In this document, the word "message" will mean a WebSocket message, not an XMPP message stanza, unless otherwise noted.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. XMPP Sub-Protocol

3.1. Handshake

The XMPP sub-protocol is used to transport XMPP over a WebSocket connection. The client and server agree to this protocol during the WebSocket handshake (see Section 1.3 of [RFC6455]).

During the WebSocket handshake, the client MUST include the value 'xmpp' in the list of protocols for the 'Sec-WebSocket-Protocol' header. The reply from the server MUST also contain 'xmpp' in its own 'Sec-WebSocket-Protocol' header in order for an XMPP sub-protocol connection to be established.

If a client receives a handshake response that does not include 'xmpp' in the 'Sec-WebSocket-Protocol' header, then a XMPP sub-protocol WebSocket connection was not established and the client MUST close the WebSocket connection.

Once the handshake has successfully completed, WebSocket messages sent or received MUST conform to the protocol defined in the rest of this document.

The following is an example of a WebSocket handshake, followed by opening an XMPP stream:

```
C:  GET /xmpp-websocket HTTP/1.1
    Host: example.com
    Upgrade: websocket
    Connection: Upgrade
    Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
    Origin: http://example.com
    ...
    Sec-WebSocket-Protocol: xmpp
    Sec-WebSocket-Version: 13


S:  HTTP/1.1 101 Switching Protocols
    Upgrade: websocket
    Connection: Upgrade
    ...
    Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
    Sec-WebSocket-Protocol: xmpp

[WebSocket connection established]

C:  <open xmlns="urn:ietf:params:xml:ns:xmpp-framing"
          to="example.com"
          version="1.0" />

S:  <open xmlns="urn:ietf:params:xml:ns:xmpp-framing"
          from="example.com"
          id="++TR84Sm6A3hnt3Q065SnAbbk3Y="
          xml:lang="en"
          version="1.0" />
```

3.2. WebSocket Messages

Data frame messages in the XMPP sub-protocol MUST be of the text type and contain UTF-8 encoded data.

3.3. XMPP Framing

The framing method for the binding of XMPP to WebSocket differs from the framing method for the TCP binding as defined in [RFC6120]; in particular, the WebSocket binding adopts the message framing provided by WebSocket to delineate the stream open and close headers, stanzas, and other top-level stream elements.

3.3.1. Framed XML Stream

The start of a framed XML stream is marked by the use of an opening "stream header" which is an <open/> element with the appropriate attributes and namespace declarations (see Section 3.3.2). The attributes of the <open/> element are the same as those of the <stream/> element defined defined for the 'http://etherx.jabber.org/streams' namespace in [RFC6120] and with the same semantics and restrictions.

The end of a framed XML stream is denoted by the closing "stream header" which is a <close/> element with its associated attributes and namespace declarations (see Section 3.3.2).

The introduction of the <open/> and <close/> elements is motivated by the parsable XML document framing restriction in Section 3.3.3. As a consequence, note that a framed XML stream does not provided a wrapping <stream:stream/> element encompassing the entirety of the XML stream, as in [RFC6120].

3.3.2. Framed Stream Namespace

The XML stream "headers" (the <open/> and <close/> elements) MUST be qualified by the namespace 'urn:ietf:params:xml:ns:xmpp-framing' (the "framed stream namespace"). If this rule is violated, the entity that receives the offending stream header MUST close the stream with an error, which MUST be <invalid-namespace> (see Section 4.9.3.10 of [RFC6120]).

3.3.3. Stream Frames

The individual frames of a framed XML stream have a one-to-one correspondence with WebSocket messages, and MUST be parsable as standalone XML documents, complete with all relevant namespace and language declarations. The inclusion of XML declarations, however, is NOT RECOMMENDED, as WebSocket messages are already mandated to be UTF-8 encoded. Inclusions of declarations would only add a constant size overhead to each message.

The first character of each frame MUST be a '<' character.

Every XMPP stanza or other XML element (including the stream open and close headers) sent directly over the XML stream MUST be sent in its own frame.

Example of a WebSocket message that contains an independently parsable XML document:

```
<message xmlns="jabber:client" xml:lang="en">
  <body>Every WebSocket message is parsable by itself.</body>
</message>
```

Note that for stream features and errors, there is no parent context element providing the "stream" namespace prefix as in [RFC6120], and thus the stream prefix MUST be declared or use an unprefixed form:

```
<stream:features xmlns:stream="http://etherx.jabber.org/streams">
  <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind"/>
</stream:features>

-- OR --

<error xmlns="http://etherx.jabber.org/streams">
  <host-unknown xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
</error>
```

3.4. Stream Initiation

The first message sent after the WebSocket opening handshake MUST be from the initiating entity, and MUST be an <open/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-framing' namespace and with the same attributes mandated for the <stream> opening tag as described in Section 4.7 of [RFC6120].

The receiving entity MUST respond with either an <open /> element (whose attributes match those described in Section 4.7 of [RFC6120]) or a <close /> element (see Section 3.6.1).

An example of a successful stream initiation exchange:

```
C:  <open xmlns="urn:ietf:params:xml:ns:xmpp-framing"
          to="example.com"
          version="1.0" />

S:  <open xmlns="urn:ietf:params:xml:ns:xmpp-framing"
          from="example.com"
          id="++TR84Sm6A3hnt3Q065SnAbbk3Y="
          xml:lang="en"
          version="1.0" />
```

Clients MUST NOT multiplex XMPP streams over the same WebSocket.

3.5. Stream Errors

Stream level errors in XMPP are fatal. Should such an error occur, the server MUST send the stream error as a complete element in a message to the client.

If the error occurs during the opening of a stream, the server MUST send the initial open element response, followed by the stream level error in a second WebSocket message frame. The server MUST then close the connection as specified in Section 3.6.

3.6. Closing the Connection

The closing process for the XMPP sub-protocol mirrors that of the XMPP TCP binding as defined in Section 4.4 of [RFC6120], except that a <close/> element is used instead of the ending </stream:stream> tag.

Either the server or the client may close the connection at any time. Before closing the connection, the closing party is expected to first close the XMPP stream (if one has been opened) by sending a message with the <close/> element, qualified by the "urn:ietf:params:xml:ns :xmpp-framing" namespace. The stream is considered closed when a corresponding <close/> element is received from the other party, and the XMPP session is ended.

To then close the WebSocket connection, the closing party MUST initiate the WebSocket closing handshake (see Section 7.1.2 of [RFC6455]).

An example of ending an XMPP over WebSocket session by first closing the XMPP stream layer and then the WebSocket connection layer:

```
   Client                         (XMPP WSS)                      Server
   |  |                                                             |  |
   |  | <close xmlns="urn:ietf:params:xml:ns:xmpp-framing />        |  |
   |  |------------------------------------------------------------>|  |
   |  |       <close xmlns="urn:ietf:params:xml:ns:xmpp-framing" /> |  |
   |  |<------------------------------------------------------------|  |
   |  |                                                             |  |
   |  |                      (XMPP Stream Closed)                   |  |
   |  +-------------------------------------------------------------+  |
   |                                                                   |
   | WS CLOSE FRAME                                                    |
   |------------------------------------------------------------------>|
   |                                                    WS CLOSE FRAME |
   |<------------------------------------------------------------------|
   |                                                                   |
   |                         (Connection Closed)                       |
   +-------------------------------------------------------------------+
```

If the WebSocket connection is closed or broken without the XMPP stream having been closed first, then the XMPP stream is considered implicitly closed and the XMPP session ended; however, if the use of stream management resumption was negotiated (see [XEP-0198]), the server SHOULD consider the XMPP session still alive for a period of time based on server policy as specified in [XEP-0198].

3.6.1. see-other-uri

If the server wishes at any point to instruct the client to move to a different WebSocket endpoint (e.g., for load balancing purposes), then a <close/> element is sent with the 'see-other-uri' attribute set to the URI of the new connection endpoint (which MAY be for a different transport method, such as BOSH (see [XEP-0124] and [XEP-0206])).

Clients MUST NOT accept suggested endpoints with a lower security context (e.g., moving from a 'wss://' endpoint to a 'ws://' or 'http://' endpoint).

An example of the server closing a stream and instructing the client to connect at a different WebSocket endpoint:

```
S: <close xmlns="urn:ietf:params:xml:ns:xmpp-framing"
          see-other-uri="wss://otherendpoint.example/xmpp-bind" />
```

3.7. Stream Restarts

Whenever a stream restart is mandated (see Section 4.3.3 of [RFC6120]), both the server and client streams are implicitly closed and new streams MUST be opened, using the same process as in Section 3.4.

The client MUST send a new stream <open/> element and MUST NOT send a closing <close/> element.

An example of restarting the stream after successful SASL negotiation:

```
S: <success xmlns="urn:ietf:params:xml:ns:xmpp-sasl" />

[Streams implicitly closed]

C: <open xmlns="urn:ietf:params:xml:ns:xmpp-framing"
         to="example.com"
         version="1.0" />
```

3.8. Pings and Keepalives

Traditionally, XMPP servers and clients often send "whitespace keepalives" (see Section 4.6.1 of [RFC6120]) between stanzas to maintain an XML stream. However, for the XMPP sub-protocol each message is required to start with a '<' character, and, as such, whitespace keepalives MUST NOT be used.

As alternatives, the XMPP Ping extension [XEP-0199] and the XMPP Stream Management extension [XEP-0198] provide pinging mechanisms. Either of these extensions (or both) MAY be used to determine the state of the connection.

Clients and servers MAY also use WebSocket ping control frames for this purpose, but note that some environments, such as browsers, do not provide access for generating or monitoring ping control frames.

3.9. Use of TLS

TLS cannot be used at the XMPP sub-protocol layer because the sub-protocol does not allow for raw binary data to be sent. Instead, when TLS is used, it MUST be enabled at the WebSocket layer using secure WebSocket connections via the 'wss' URI scheme. (See Section 10.6 of [RFC6455].)

Because TLS is to be provided outside of the XMPP sub-protocol layer, a server MUST NOT advertise TLS as a stream feature (see Section 4.6

of [RFC6120]) when using the XMPP sub-protocol. Likewise, a client MUST ignore any advertised TLS stream feature when using the XMPP sub-protocol.

3.10. Stream Management

In order to alleviate the problems of temporary disconnections, the client MAY use the XMPP Stream Management extension [XEP-0198] to confirm when stanzas have been received by the server.

In particular, the client MAY use session resumption in [XEP-0198] to recreate the same stream session state after a temporary network unavailability or after navigating to a new URL in a browser.

4. Discovering the WebSocket Connection Method

Section 3 of [RFC6120] defines a procedure for connecting to an XMPP server, including ways to discover the TCP/IP address and port of the server using Domain Name System service (DNS SRV) records [RFC2782]. When using the WebSocket binding as specified in this document (instead of the TCP binding as specified in [RFC6120]), a client needs an alternative way to discover information about the server's connection methods, since web browsers and other WebSocket-capable software applications typically cannot obtain such information from the DNS.

The alternative lookup process uses Web Host Metadata [RFC6415] and Web Linking [RFC5988], where the link relation type is "urn:xmpp:alt-connections:websocket" as described in Discovering Alternate XMPP Connection Methods [XEP-0156]. Conceptually, the host-meta lookup process used for the WebSocket binding is analogous to the DNS SRV lookup process used for the TCP binding. The process is as follows.

1. Send a request over secure HTTP to the path "/.well-known/host-meta" at an HTTP origin [RFC6454] that matches the XMPP service domain (e.g., a URL of "https://im.example.org/.well-known/host-meta" if the XMPP service domain is "im.example.org").

2. Retrieve a host-meta document specifying a link relation type of "urn:xmpp:alt-connections:websocket", such as:

```
<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
  <Link rel="urn:xmpp:alt-connections:websocket"
        href="wss://im.example.org:443/ws" />
</XRD>
```

Servers MAY expose discovery information using host-meta documents, and clients MAY use such information to determine the WebSocket endpoint for a server.

In cases where the XMPP service domain does not match the discovered web origin of the WebSocket endpoint, the Web-host metadata SHOULD be used to establish trust between the XMPP server domain and the WebSocket endpoint as long as the host-meta request and response occurred over secure HTTP; this is especially relevant in multi-tenant situations where the same WebSocket endpoint is serving multiple XMPP domains (e.g., the XMPP service domains for both "example.com" and "im.example.org" might be serviced by the same WebSocket endpoint at "hosting.example.net"). See Section 6 for related discussion.

5. IANA Considerations

5.1. WebSocket Subprotocol Name

This specification requests IANA to register the WebSocket XMPP sub-protocol under the "WebSocket Subprotocol Name" Registry with the following data:

Subprotocol Identifier: xmpp

Subprotocol Common Name: WebSocket Transport for the Extensible Messaging and Presence Protocol (XMPP)

Subprotocol Definition: this document

5.2. URN Sub-Namespace

A URN sub-namespace for framing of Extensible Messaging and Presence Protocol (XMPP) streams is defined as follows.

URI: urn:ietf:params:xml:ns:xmpp-framing

Specification: this document

Description: This is the XML namespace name for framing of Extensible Messaging and Presence Protocol (XMPP) streams as defined by RFC XXXX.

Registrant Contact: IESG <iesg@ietf.org>

6. Security Considerations

The WebSocket binding for XMPP differs in several respects from the TCP binding defined in [RFC6120]:

1. As described under Section 4 of this document, the method for discovering a connection endpoint does not use DNS SRV records as in the TCP binding, but instead uses Web Host Metadata files retrieved via HTTPS from a URL at the XMPP service domain. From a security standpoint, this is functionally equivalent to resolution via DNS SRV records (and still relies on the DNS for resolution of the XMPP source domain).

2. The method for authenticating a connection endpoint uses TLS (typically with PKIX certificates) as in the TCP binding, but the identity to be authenticated is the connection endpoint address instead of the XMPP service domain; delegation from the XMPP service domain to the connection endpoint address (if any) is accomplished via the discovery method described under Section 4. Thus the connection endpoint is still authenticated, and the delegation is secure as long as the Web Host Metadata file is retrieved via HTTPS. However, note that in practice this option might not be employed when user agents are configured or deployed for a particular delegated domain.

3. The framing method described under Section 3.3 follows the WebSocket pattern by sending one top-level XML element per WebSocket message, instead of using streaming XML as in the TCP binding. However, the framing method has no impact on the security properties of an XMPP session (e.g., end-to-end encryption of XML stanzas can be accomplished just as easily with WebSocket framing as with streaming XML).

4. In all other respects (e.g., user authentication via SASL, allowable characters in XMPP addresses, and re-use of various technologies such as Base 64, SASL mechanisms, UTF-8, and XML), the WebSocket binding does not differ from the TCP binding, and thus does not modify the security properties of the protocol. In all these respects, the security considerations of [RFC6120] apply directly to the WebSocket binding.

In order to ensure that communications over the WebSocket binding are as secure as communications over the TCP binding, an operator needs to (1) serve the Web Host Metadata file for the XMPP service domain over secure HTTP ('https' URIs) only, (2) configure the WebSocket connection endpoint to use Transport Layer Security ('wss' URIs) only, and (3) deploy certificates that properly identify the XMPP

service domain and WebSocket connection endpoint for usages (1) and (2), respectively.

Since application level TLS cannot be used (see Section 3.9), applications need to protect the privacy of XMPP traffic at the WebSocket or other appropriate layer.

Browser-based applications are not able to inspect and verify, at the application layer, the certificate used for the WebSocket connection to ensure that it corresponds to the domain specified as the 'to' address of the XMPP stream. There are two cases:

1. If the XMPP service domain matches the origin for the WebSocket connection, the relevant check is already performed by the browser. For example, the XMPP service domain might be "foo.example" and the WebSocket endpoint discovered for the link relation type of "urn:xmpp:alt-connections:websocket" might be "wss://foo.example/websocket". As long as the certificate provided over WebSocket or HTTPS is verified according to the rules defined for secure HTTP [RFC2818], then the browser will report the successful establishment of a secure connection to the application. (However, as noted, the application is still not able to independently inspect and verify the certificate, and needs to trust the browser; this is a limitation of existing browser technologies, and thus cannot be worked around by WebSocket applications.)

2. In situations where the user agent has to deal with delegation and the domain of the XMPP server does not match the web origin of the WebSocket endpoint (such as multi-tenant hosting situations), the host-meta process described under Section 4 SHOULD be used to delegate trust from the XMPP server domain to the WebSocket origin, as long as the host-meta request and response occurred over secure HTTP (with appropriate certificate verification as defined in [RFC2818]).

When presented with a new WebSocket endpoint via the 'see-other-uri' attribute of a <close/> element, clients MUST NOT accept the suggestion if the security context of the new endpoint is lower than the current one in order to prevent downgrade attacks from a 'wss://' endpoint to 'ws://'.

The Security Considerations for both WebSocket (see Section 10 of [RFC6455]) and XMPP (see Section 13 of [RFC6120]) apply to the WebSocket XMPP sub-protocol.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6415] Hammer-Lahav, E. and B. Cook, "Web Host Metadata", RFC 6415, October 2011.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

7.2. Informative References

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC6121] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence", RFC 6121, March 2011.

[RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[XEP-0124] Paterson, I., Smith, D., Saint-Andre, P., Moffitt, J., and L. Stout, "Bidirectional-streams Over Synchronous HTTP (BOSH)", XSF XEP 0124, November 2013.

[XEP-0156] Hildebrand, J., Saint-Andre, P., and L. Stout, "Discovering Alternative XMPP Connection Methods", XSF XEP 0156, January 2014.

[XEP-0198]
Karneges, J., Saint-Andre, P., Hildebrand, J., Forno, F., Cridland, D., and M. Wild, "Stream Management", XSF XEP 0198, June 2011.

[XEP-0199]
Saint-Andre, P., "XMPP Ping", XSF XEP 0199, June 2009.

[XEP-0206]
Paterson, I., Saint-Andre, P., and L. Stout, "XMPP Over BOSH", XSF XEP 0206, November 2013.

[XML-SCHEMA]
Thompson, H., Maloney, M., Mendelsohn, N., and D. Beech, "XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

Appendix A. Acknowledgements

The authors wish to thank the following individuals for their feedback: Andreas Guth, Bjoern Hoerhmann, Dave Cridland, Florian Zeitz, Kurt Zeilenga, Matt Miller, Matthew Wild, Paul Aurich, Sergey Dobrov, Waqas Hussain.

Dan Romascanu reviewed the document on behalf of the General Area Review Team.

During IESG review, Barry Leiba Benoit Claise, Dan Romasanu, Jari Arkko, Juergen Schoenwaelder, Spencer Dawkins, Stephen Farrell, Ted Lemon, Kathleen Moriarty, Pete Resnick caught several issues that needed to be addressed.

The authors gratefully acknowledge the assistance of Peter Saint-Andre as document shepherd, Ben Campbell and Joe Hildebrand as the working group chairs, and Richard Barnes as the sponsoring Area Director.

Appendix B. XML Schema

The following schema formally defines the 'urn:ietf:params:xml:ns:xmpp-framing' namespace used in this document, in conformance with W3C XML Schema [XML-SCHEMA]. Because validation of XML streams and stanzas is optional, this schema is not normative and is provided for descriptive purposes only.

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
   <xs:schema
       xmlns:xs='http://www.w3.org/2001/XMLSchema'
       targetNamespace='urn:ietf:params:xml:ns:xmpp-framing'
       xmlns='urn:ietf:params:xml:ns:xmpp-framing'
       elementFormDefault='unqualified'>

     <xs:element name='open'>
       <xs:complexType>
         <xs:simpleContent>
           <xs:extension base='empty'>
             <xs:attribute name='from' type='xs:string'
                           use='optional'/>
             <xs:attribute name='id' type='xs:string'
                           use='optional'/>
             <xs:attribute name='to' type='xs:string'
                           use='optional'/>
             <xs:attribute name='version' type='xs:decimal'
                           use='optional'/>
             <xs:attribute ref='xml:lang'
                           use='optional'/>
           </xs:extension>
         </xs:simpleContent>
       </xs:complexType>
     </xs:element>

     <xs:element name='close'>
       <xs:complexType>
         <xs:simpleContent>
           <xs:extension base='empty'>
             <xs:attribute name='from' type='xs:string'
                           use='optional'/>
             <xs:attribute name='id' type='xs:string'
                           use='optional'/>
             <xs:attribute name='see-other-uri' type='xs:anyURI'
                           use='optional'/>
             <xs:attribute name='to' type='xs:string'
                           use='optional'/>
             <xs:attribute name='version' type='xs:decimal'
                           use='optional'/>
             <xs:attribute ref='xml:lang'
                           use='optional'/>
           </xs:extension>
         </xs:simpleContent>
       </xs:complexType>
     </xs:element>

     <xs:simpleType name='empty'>
       <xs:restriction base='xs:string'>
```

```
         <xs:enumeration value=''/>
       </xs:restriction>
     </xs:simpleType>

   </xs:schema>
```

Authors' Addresses

Lance Stout (editor)
&yet

Email: lance@andyet.net

Jack Moffitt
Mozilla

Email: jack@metajack.im

Eric Cestari
cstar industries

Email: eric@cstar.io