XEP-0337: Event Logging over XMPP

Peter Waher
mailto:peterwaher@hotmail.com
xmpp:peter.waher@jabber.org
http://www.linkedin.com/in/peterwaher

2017-09-11
Version 0.3

Status	Type	Short Name
Deferred	Standards Track	eventlogging

This specification provides a common framework for sending events to event logs over XMPP networks.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

This XEP provides a common framework for sending events over an XMPP network. These events can then be logged in event logs or analyzed by network monitors to analyze the status or operation of the network and its participants.
The basic principle behind interoperable event logging over XMPP is the definition of a common XML element defining the event. This payload is then sent in a normal message stanza (i.e. with **type='normal'**) to the recipient. The recipient in turn, if it understands the payload, can choose to store it in an event log, forward it or analyze it accordingly.
There are various event log packages available, but none yet defined for XMPP or using a well-defined and known XML format. Therefore, this document defines such an XML format. This format is able to store Syslog [1] compliant event information, even though the Syslog event model has been somewhat extended. Also, in the use of the facility attribute, this XEP does not have the same restrictions compared to the Syslog specification.
This document does not restrict the use of event messages to directed message stanzas alone. It may be envisioned that some would like to publish event information through Publish-Subscribe (XEP-0060) [2] or other mechanisms. It is not in the scope of this document to specify such transports however, as it only deals with direct messages, but a brief list is provided in the Security Considerations section.

2 Glossary

The following table lists common terms and corresponding descriptions.

Event A piece of information describing an event in time.

Event ID An attribute providing a machine readable ID to the type of event in question without having to parse the message text itself.

Event Level Provides an additional level on the previous classification (Minor, Medium, Major).

Event Type Coarse classification of the event (Debug, Informational, Notice, Warning, Error, Critical, Alert, Emergency).

Facility What type of device, server, application, etc., is sending the message.

Message A (human readable) text message describing what has occurred.

Module The module reporting the event. Larger software packages are often divided into modules. Keeping track of which modules report which events can be useful when analyzing system performance.

[1] RFC-5424: The Syslog Protocol <https://tools.ietf.org/html/rfc5424>
[2] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.

Object The object to which the event message refers to, on which the current action is performed.

Stack Trace Exact position in the code from which the event was reported or where the error occurred. Reporting it in a separate attribute unclutters the message, and removes the need to define custom tags.

Subject The subject causing the event to take place or performing the action (for instance, user, process, etc.)

Tag A custom tag or parameter attached to an event. Each tag has a name and a value and an optional data type.

Timestamp When the event occurred.

3 Use Cases

3.1 Sending a simple event message

The following example shows how to send a simple event using a normal message to an event log. Only two parameters are required: The timestamp of the message goes into the **timestamp** attribute, and the actual messages goes into a child element named **message**. This event will be treated as a **Minor Informational** event by the recipient.

Listing 1: Simple event message

```
<message from='device@example.org/device' to='eventlog@example.org'
    type='normal' xml:lang='en'>
      <log xmlns='urn:xmpp:eventlog' timestamp='2013-11-10T15:52:23Z'>
        <message>Something happened.</message>
      </log>
   </message>
```

3.2 Sending a multi-line event message

The following example shows how to send a multi-line event.

Listing 2: Multi-line event message

```
<message from='device@example.org/device' to='eventlog@example.org'
    type='normal' xml:lang='en'>
      <log xmlns='urn:xmpp:eventlog' timestamp='2013-11-12T11:47:12Z'>
        <message>10 objects deleted:
Object 1
...
Object 10</message>
```

```
      </log>
   </message>
```

3.3 Specifying type and level

The following example shows how an event message can be categorized with an event type and level.

Listing 3: Specifying type and level

```
<message from='device@example.org/device' to='eventlog@example.org'
   type='normal' xml:lang='en'>
      <log xmlns='urn:xmpp:eventlog' timestamp='2013-11-10T15:54:55Z'
          type='Warning' level='Major'>
         <message>Low on memory.</message>
      </log>
   </message>
```

3.4 Specifying object and subject

The following example shows how an event message can be further enhanced by providing object and subject information.

Listing 4: Specifying object and subject

```
<message from='device@example.org/device' to='eventlog@example.org'
   type='normal' xml:lang='en'>
      <log xmlns='urn:xmpp:eventlog' timestamp='2013-11-10T15:58:12Z'
          type='Informational' level='Major' object='Towel' subject='
          Arthur␣Dent'>
         <message>Object deleted.</message>
      </log>
   </message>
```

3.5 Specifying an event ID

The following example shows how to send an event message with an event ID that can be singled out later to be analyzed by administrators, for instance.

Listing 5: Specifying an event ID

```
<message from='device@example.org/device' to='eventlog@example.org'
   type='normal' xml:lang='en'>
      <log xmlns='urn:xmpp:eventlog' id='LoginFailed' timestamp='
          2013-11-10T16:04:45Z' type='Warning' level='Minor' object='
          user1' subject='10.0.0.1'>
```

```
        <message>User attempted to login but provided incorrect
            password.</message>
      </log>
   </message>
```

3.6 Tagging events with custom information

The following example shows how to tag an event using custom information in a way that is easy to interpret and process.

Listing 6: Tagging events with custom information

```
<message from='device@example.org/device' to='eventlog@example.org'
    type='normal' xml:lang='en'>
      <log xmlns='urn:xmpp:eventlog' timestamp='2013-11-10T16:07:01Z'
          type='Informational' level='Minor'>
          <message>Current resources.</message>
          <tag name='RAM' value='1655709892' type='xs:long'/>
          <tag name='CPU' value='75.45' type='xs:double'/>
          <tag name='HardDrive' value='163208757248' type='xs:long'/>
      </log>
   </message>
```

Note: Any **tag** elements must come after the **message** element.

3.7 Specifying program module

The following example shows how module information can be provided in events to more easily be able to single out information relating to the same application running on different machines.

Listing 7: Specifying program module

```
<message from='device@example.org/device' to='eventlog@example.org'
    type='normal' xml:lang='en'>
      <log xmlns='urn:xmpp:eventlog' timestamp='2013-11-10T16:17:56Z'
          type='Error' level='Major' object='object1' subject='user1'
          module='application1'>
          <message>Something horrible happened.</message>
      </log>
   </message>
```

3.8 Sending debug information

The following example shows how to send a debug message with a stack trace, module and custom information.

Listing 8: Sending debug information

```
<message from='device@example.org/device' to='eventlog@example.org'
    type='normal' xml:lang='en'>
      <log xmlns='urn:xmpp:eventlog' timestamp='2013-11-10T16:12:25Z'
          type='Debug' level='Major' module='My new application'
          stackTrace='file1, line 1, ...'>
          <message>Something is rotten in the state of Denmark.</
              message>
          <tag name='a' value='1' type='xs:int'/>
          <tag name='b' value='10' type='xs:int'/>
          <tag name='s' value='Hello World!' type='xs:string'/>
          <stackTrace>File1, Line1, ...
File2, Line2, ...
...</stackTrace>
      </log>
   </message>
```

Note: Any **stackTrace** element must come after the **message** element and any **tag** elements.

3.9 Sending multiple events

The following example shows how multiple events can be sent in a single message. The receiver should interpret this as two different events having been received.

Listing 9: Sending multiple events

```
<message from='device@example.org/device' to='eventlog@example.org'
    type='normal' xml:lang='en'>
      <log  xmlns='urn:xmpp:eventlog' timestamp='2013-11-10T15:52:23Z'
          >
          <message>Something happened.</message>
      </log>
      <log  xmlns='urn:xmpp:eventlog' timestamp='2013-11-10T15:54:23Z'
          >
          <message>Something else happened.</message>
      </log>
   </message>
```

4 Determining Support

If an entity supports the reception of events as specified herein, it MUST advertise that fact by returning a feature of "urn:xmpp:eventlog" in response to Service Discovery (XEP-0030) [3]

[3] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

information requests.

Listing 10: Service discovery information request

```
<iq type='get'
    from='device@example.org/device'
    to='eventlog@example.org'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 11: Service discovery information response

```
<iq type='result'
    from='eventlog@example.org'
    to='device@example.org/device'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='urn:xmpp:eventlog'/>
    ...
  </query>
</iq>
```

In order for an application to determine whether an entity supports this protocol, where possible it SHOULD use the dynamic, presence-based profile of service discovery defined in Entity Capabilities (XEP-0115) [4]. However, if an application has not received entity capabilities information from an entity, it SHOULD use explicit service discovery instead.

5 Implementation Notes

5.1 Event Type

The following table lists possible event types that can be used. If none is specified for an event, it is assumed that the event is **Informational**. It is largely based on the severity levels of Syslog.

Type	Description
Debug	Developers can ask applications to send debug messages during development or testing to more easily see what happens in a system.
Informational	An informational message describing a normal event.
Notice	Represents a significant condition or change that administrators should be aware of.
Warning	A warning condition. If not taken into account, the condition could turn into an error.

[4] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

Type	Description
Error	An error condition. A condition has been detected that is considered to be an error or a fault.
Critical	A critical condition. An error so great that it could escalate into something graver if not addressed.
Alert	An alert condition. Action must be taken immediately.
Emergency	System is unusable.

5.2 Event Level

Given an Event Type, an event level can provide additional information about the extent or importance of the event (a second dimension).

Level	Description
Minor	Minor events, concerning normal operating procedures.
Medium	Medium events.
Major	More substantial events or events that are affecting larger parts of the system.

5.3 Event ID

Using Event IDs, the application can provide a machine understandable classification of the event. Examples could be ”Login”-events, ”ConnectionProblem”-events, etc. It is easier to group, parse or interpret events and their tags if you know what type of event it is. Event IDs are manufacturer specific, and only provide a means to more easily extract subsets of events for processing without having to parse message texts (which should be allowed to be localizable).
Note: To avoid problems when running applications using different locales, event IDs should never be localized.

5.4 Object

An event is often linked to an object, e.g. on which object an action was performed, or which object is reporting a condition. The object field permits the tagging of objects in a common way. It is later easy to extract all events relating to a specific object by using this attribute.

5.5 Subject

An event is often also linked to a subject, i.e. who or what performed a given action resulting in the event or condition. The subject field permits the tagging of subjects in a common way. It is later easy to extract all events relating to a specific subject by using this attribute.

5.6 Facility

Facility can be either a facility in the network sense or in the system sense. This document does not restrict its use to the possible choices defined by other protocols such as Syslog, and leaves it open. However, it is left as a special attribute since it is important in monitoring applications.

5.7 Module

A module is part of a larger software package. Using the module attribute makes it easier to attribute events to specific parts of a distributed application and analyze them separately.

5.8 Stack Trace

Stack Traces can be important information to developers and correlate events to actual locations in the code that generated the event. This document does not specify any particular format for stack traces.

5.9 Tag

Any event can have a custom set of tags attached to it. A tag is required to have a name and a value. It can also optionally specify a data type. Data types are specified using Qualified Names (QNames). If possible, they should adhere to the list of Data Forms Validation Datatypes [5] that is maintained by the XMPP Registrar.
Note: To avoid problems when running applications using different locales, tag names should never be localized.

5.10 Normalized tables

If persisting received events in a database, care should be taken if normalized tables are used for storage of tags names and values, event IDs, objects, subjects, facilities and modules. If this is the case, the receiver should look for types of values that can be incompatible with normalized tables (such as floating point values or numbers in general, GUIDs, resource names in JIDs etc.) and replace them with some descriptive text and append the corresponding value

[5] Data Forms Validation Datatypes <http://xmpp.org/registrar/xdv-datatypes.html>

in the message text instead. This to avoid problems with indexes in databases because of devices implemented by third parties.
It is still valid to send information like sequence numbers, unique GUIDs, measurements, resource names in JIDs etc. in tag names and values, but such information should be avoided in event IDs, objects, subjects, facilities and modules, as they can cause problems further down the line.

5.11 Multi-line messages and stack traces

The messag text and stack trace parts of an event message lie as simple type valued child elements (xs:string). This allows for simple encoding of multi-line text information into these two parameters. However, do not indent new lines when serializing multi-line text to these parameters to make the XML look nicer. The recipient cannot know what whitespace is indenting and what is part of the actual information.

6 Internationalization Considerations

6.1 Time Zones

All timestamps and dateTime values use the XML data type xs:dateTime to specify values. These values include a date, an optional time and an optional time zone.
Note: If time zone is not available, it is supposed to be undefined. The recipient of an event message without time zone information should assume the sender has the same time zone as the received, if not explicitly configured otherwise on the recipient side.
If devices report time zone, this information should be propagated throughout the system. Otherwise, comparing timestamps from different time zones will be impossible.

6.2 xml:lang

Event messages SHOULD contain an **xml:lang** attribute on the message stanza to specify the language used in message texts, etc. If language information is not available, e.g. if relaying messages are not created by the device itself, the **xml:lang** attribute can be omitted.

7 Security Considerations

7.1 Zero configuration

Even though this document permits zero-configuration of devices to event logs, this might not always be the best option. Event information might be sensitive and should not be sent to

anybody just because they support the event log feature as defined in this document.

7.2 Sensitive information

Never log information that should be handled securely or encrypted, such as passwords.

7.3 Transport options

The following subsections lists different transport options together with security considerations for each one.

7.3.1 Direct messages

This document explicitly describes how to send event messages in direct messages. If sensitive information is being sent, end-to-end encryption should be considered.

7.3.2 Publish/Subscribe

Event messages could be published using Publish-Subscribe. But, even more care should be taken to log only information that can be published openly. If there's risk for sensitive information to be logged, the publish/subscribe pattern should be avoided.

8 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [6].

9 XMPP Registrar Considerations

The protocol schema needs to be added to the list of XMPP protocol schemas.

[6] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

10 XML Schema

```
<?xml version='1.0' encoding='UTF-8'?>
    <xs:schema
        xmlns:xs='http://www.w3.org/2001/XMLSchema'
        targetNamespace='urn:xmpp:eventlog'
        xmlns='urn:xmpp:eventlog'
        elementFormDefault='qualified'>

        <xs:element name='log'>
            <xs:complexType>
                <xs:sequence>
                    <xs:element name='message' type='xs:string'
                        minOccurs='1' maxOccurs='1'/>
                    <xs:element name='tag' minOccurs='0' maxOccurs='
                        unbounded'>
                        <xs:complexType>
                            <xs:attribute name='name' type='xs:string'
                                use='required'/>
                            <xs:attribute name='value' type='xs:string
                                ' use='required'/>
                            <xs:attribute name='type' type='xs:QName'
                                use='optional' default='xs:string'/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name='stackTrace' type='xs:string'
                        minOccurs='0' maxOccurs='1'/>
                </xs:sequence>
                <xs:attribute name='timestamp' type='xs:dateTime' use=
                    'required'/>
                <xs:attribute name='id' type='xs:string' use='optional
                    '/>
                <xs:attribute name='type' type='EventType' use='
                    optional' default='Informational'/>
                <xs:attribute name='level' type='EventLevel' use='
                    optional' default='Minor'/>
                <xs:attribute name='object' type='xs:string' use='
                    optional'/>
                <xs:attribute name='subject' type='xs:string' use='
                    optional'/>
                <xs:attribute name='facility' type='xs:string' use='
                    optional'/>
                <xs:attribute name='module' type='xs:string' use='
                    optional'/>
            </xs:complexType>
        </xs:element>

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

```
                <xs:enumeration value='Debug'/>
                <xs:enumeration value='Informational'/>
                <xs:enumeration value='Notice'/>
                <xs:enumeration value='Warning'/>
                <xs:enumeration value='Error'/>
                <xs:enumeration value='Critical'/>
                <xs:enumeration value='Alert'/>
                <xs:enumeration value='Emergency'/>
            </xs:restriction>
        </xs:simpleType>

        <xs:simpleType name='EventLevel'>
            <xs:restriction base='xs:string'>
                <xs:enumeration value='Major'/>
                <xs:enumeration value='Medium'/>
                <xs:enumeration value='Minor'/>
            </xs:restriction>
        </xs:simpleType>

    </xs:schema>
```

11 Acknowledgements

Thanks in alphabetical order to Dave Cridland, Joachim Lindborg, Karin Forsell, Ludovic Bocquet, Markus Kohlhase, Matthew Wild, Mike Taylor, Philipp Hancke, Robert Kosten, Steffen Larsen, and Yusuke DOI for all valuable feedback.