XEP-0410: MUC Self-Ping (Schrödinger’s Chat)

Georg Lukas  
mailto:georg@op-co.de  
xmpp:georg@yax.im  

2018-08-31  
Version 0.1.0

<table>
<thead>
<tr>
<th>Status</th>
<th>Type</th>
<th>Short Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experimental</td>
<td>Standards Track</td>
<td>muc-selfping</td>
</tr>
</tbody>
</table>

This protocol extension for XEP-0045 Multi User Chat allows clients to check whether they are still joined to a chatroom.
Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2018 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** .................................................. 1
2. **Requirements** ............................................... 1
3. **Client Self-Presence Check** .................................. 1
   3.1 Possible Protocol Approaches ................................ 1
   3.2 Performing a Self-Ping ....................................... 2
   3.3 Server Optimization .......................................... 3
4. **Implementation Notes** ....................................... 4
5. **Security Considerations** .................................... 4
6. **IANA Considerations** ....................................... 4
7. **XMPP Registrar Considerations** ............................ 4
8. **XML Schema** .................................................. 4
1 Introduction

The Multi-User Chat (XEP-0045) protocol was not designed to handle s2s interruptions or message loss well. Rather often, the restart of a server or a component causes a client to believe that it is still joined to a given chatroom, while the chatroom service does not know of this participant.

Existing approaches for re-synchronization are either inefficient (presence updates and “silent” messages are reflected to all participants, totalling to $O(N^2)$ stanzas per time unit), or mask message / presence losses (the implicit join performed via the deprecated GC1.0 protocol).

This specification aims to provide the most efficient, albeit not the most elegant, way for clients to periodically check whether they are still joined to a chatroom.

2 Requirements

This specification only makes sense in the context of Multi-User Chat (XEP-0045) chatrooms. It makes use of XMPP Ping (XEP-0199) to perform periodic self-pings.

Server support for this extension is optional, but will significantly improve the reliability with Multi-Session Nicks and mobile clients.

3 Client Self-Presence Check

A typical connection between a client and a Multi-User-Chatroom (MUC) goes through the client-to-server link, possibly a server-to-server link and a typically local server-to-component link. If one of the involved servers or the MUC component is restarted, or one of the links is disturbed for some time, this can lead to the removal of some or all participants from the affected MUCs, without the clients being informed.

To a participant, this looks like the MUC is silent (there is no chat activity and no presence changes), making it hard to realize that the connection was interrupted.

To prevent the bad usability effect (message loss, lack of reaction from people in a chatroom), a client needs to actively check whether it is still joined to a MUC.

3.1 Possible Protocol Approaches

There are multiple alternative approaches for a client to test whether it is still joined to a MUC:

1. **Silent message** (e.g. Chat State Notifications (XEP-0085) \(^4\)): this message will be reflected to all MUC participants, causing unwanted traffic and potentially waking up mobile devices without reason. If implemented by all clients, this will result in \(O(N^2)\) messages to the MUC.

2. **Presence update**: if the MUC service implements the legacy GC1.0 protocol, this will be treated as a join attempt, and the MUC will return the full list of participants and full room history. The user’s client will however miss partial history (other participants leaving, potentially also messages), and this has the same drawbacks as the first solution.

3. **Private message to self**: the client can send a MUC private message to itself. However, not all MUCs support / allow private messages, and there is no way to differentiate that from the error responses.

4. **Private IQ to self**: the client can send an IQ to its own participant JID. MUCs typically do not forbid those, and reflect the IQ request to the client (or another client of the same user). Once that client responds to the reflected IQ, the response is delivered to the initiating client as a sign of still being joined.

5. **Dedicated MUC IQ**: a new type of IQ can be deployed to let the client explicitly check whether it is still joined to a MUC. However, this needs to be supported by the server, and the client needs to implement a fallback solution.

The private IQ is the most robust and traffic-efficient solution, and it does not rely on server support. The [XMPP Ping (XEP-0199)](https://xmpp.org/extensions/xep-0199) protocol is appropriate to use for this use case.

### 3.2 Performing a Self-Ping

After an adequate amount of silence from a given MUC (e.g. 15 minutes), or from all MUCs from a given service domain, a client should initiate a self-ping. If Juliet is joined as JuliC in the characters@chat.shakespeare.lit MUC, her client will send the following ping IQ:

![Listing 1: Self-Ping by Juliet’s Client](https://example.com/listing1.png)

If Juliet’s client is not joined, the MUC service will respond with a `<not-acceptable>` error. Thus, her client can automatically attempt a rejoin.

---


If her client is joined, the IQ request will be forwarded to any one of Juliet’s joined clients.

Depending on the other client implementation and its connection status, the IQ will be responded to eventually, in one of these ways, as delivered to the “client” resource:

- **Successful IQ response**: the client is still joined.
- **Error (<service-unavailable> or <feature-not-implemented>)**: the client is joined, but the pinged client does not implement XMPP Ping (XEP-0199)
- **Error (<item-not-found>)**: the client is joined, but the participant just changed their name (e.g. initiated by a different client).
- **Any other error**: the client is probably not joined.
- **Timeout (no response)**: the MUC service (or another client) is unreachable. The client may indicate the status to the user and re-attempt the self-ping after some timeout, until it receives either an error or a success response.

### 3.3 Server Optimization

The normal routing rules of the self-ping impose two round-trips: first the initial ping from the client to the MUC, then the reflection of the ping and its response (possibly to another client), and finally the response to the initial IQ. If the other client is experiencing network connectivity issues, which is often the case with mobile devices, the ping request might never be responded to.

Therefore, a MUC service supporting this protocol may directly respond to a participant’s Ping request to the participant’s own nickname, as opposed to routing it to any of the participant’s

---

4 Implementation Notes

In Multi-Session-Nick scenarios, where multiple clients of the same user are joined as the same participant, it is possible that another client initiates a nickname change while a ping request is pending. In that case, the ping might be responded to with <item-not-found>. A client should not perform a self-ping after initiating a nickname change, and before receiving the response to the nickname change from the service, as it is not yet clear whether the new nickname will be accepted.

5 Security Considerations

A MUC service implementation should not allow a non-participant to obtain information about participants. This is however true irregardless of implementing this specification.

6 IANA Considerations

REQUIRED.

7 XMPP Registrar Considerations

REQUIRED.

8 XML Schema

REQUIRED for protocol specifications.