XEP-0142: Workgroup Queues

Matt Tucker
mailto:matt@jivesoftware.com
xmpp:jivematt@jabber.org

2018-11-03
Version 0.3.1

Status	Type	Short Name
Deferred	Standards Track	Not yet assigned

This document defines an XMPP protocol extension that enables a user to communicate with a representative of an organization, department, or workgroup.

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 **1**
- 1.1 Overview . . . 1
- 1.2 Motivation . . . 1
- 1.3 Concepts . . . 1
- 1.4 Prerequisites . . . 2

2 Roles and Responsibilities **2**
- 2.1 Roles . . . 2
- 2.2 Responsibilities . . . 3

3 User Protocol **3**
- 3.1 User States . . . 4
- 3.2 User Packet Exchanges . . . 4
  - 3.2.1 User Join Protocol . . . 5
  - 3.2.2 User Depart Protocol . . . 9
  - 3.2.3 User Status Update Protocol . . . 11
  - 3.2.4 User Invite Protocol . . . 13

4 Agent Protocol **14**
- 4.1 Agent States . . . 14
- 4.2 Agent Packet Exchanges . . . 15
  - 4.2.1 Agent Presence Protocol . . . 15
  - 4.2.2 Workgroup Status Update Protocol . . . 17
  - 4.2.3 Queue Status Update Protocol . . . 18
  - 4.2.4 Agent Status Update Protocol . . . 20
  - 4.2.5 Agent Offer Protocol . . . 21
  - 4.2.6 Agent Offer Accept/Reject Protocol . . . 23
  - 4.2.7 Agent Offer Revoke Protocol . . . 24
  - 4.2.8 Agent Invite Protocol . . . 26

5 Service Discovery **27**

6 Implementation Notes **29**

7 Security Considerations **29**

8 IANA Considerations **29**

9 XMPP Registrar Considerations **30**
- 9.1 Protocol Namespaces . . . 30
- 9.2 Service Discovery Category/Type . . . 30

10 XML Schema **30**

11 Acknowledgements 34

1 Introduction

1.1 Overview

The protocol defined herein enables users to contact a representative of an organization or workgroup without knowing the address of a particular member of that organization or workgroup. This functionality is similar to an 'email alias' with the addition of queuing pending communication requests and quality of service negotiation to accommodate the real-time nature of IM/chat. Although this protocol is generic enough to handle many use cases, specific features have been added that make it particularly suitable for customer support environments.

1.2 Motivation

This protocol addresses the need of starting a private XMPP conversation with a qualified member of a workgroup. In a standard XMPP exchange of messages, users either connect directly to another user for a one on one conversation, or connect to a chat room for a conversation between many people. The current protocols do not allow users to initiate a private conversation with any person playing a particular role in an organization or workgroup.
For example, a customer has a question and needs to talk to a support representative. The conversation is private and therefore cannot be conducted in a well-known chat room. Using the workgroup protocol, the user requests a chat with support@workgroup.example.com. The chat request is put into a queue and the server routes the chat request to individual support representatives in the support@workgroup.example.com workgroup. The support representative can accept or reject the chat request. Once the request is accepted, the conversation takes place through standard XMPP messaging protocols.

1.3 Concepts

The namespace governing this protocol is "http://jabber.org/protocol/workgroup". This namespace relies on the <iq/> element for execution, and uses the <presence/> element for announcing status updates.
This protocol depends on Service Discovery (XEP-0030) [1] for reporting and announcing available workgroup services. However, support for service discovery is entirely optional and workgroup services may be made known through other means (e.g. web pages or word of mouth .
The end result of a workgroup interaction is to negotiate and route a user and workgroup member (a.k.a. agent) to an appropriate chat room for a chat conversation using the multi-user chat (MUC) protocol. However, multi-user chat essentially 'takes over' when the workgroup protocol successfully completes so there is no overlap between the two protocols. It is RECOMMENDED that groupchat implementations support basic groupchat

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

(a.k.a. Groupchat 1.0) for maximum client compatibility.

1.4 Prerequisites

There are no requirements for supporting the workgroup protocol beyond XMPP Core [2] and Multi-User Chat (XEP-0045) [3]. Support for Data Forms (XEP-0004) [4] is optional if users need to submit additional data before joining (see User Join section of this document).

2 Roles and Responsibilities

This protocol has clearly defined roles and responsibilities for its participants.

2.1 Roles

The workgroup protocol involves three distinct participants that fill the following roles:

- **User** - The user requests a private conversation with a member of a workgroup.
- **Service** - The workgroup service receives and sends messages using the workgroup address. The workgroup address represents a general contact address which allows users to find workgroup members to talk to without the need to know any particular workgroup member's individual address. The workgroup service manages the interactions between users and agents.
- **Agent** - The agent is a member of the workgroup and can carry out conversations with users on behalf of their workgroup organization or company.

In the examples shown throughout this document, the user address <user@example.net/home>, the service address is <support@workgroup.example.com>, and the agent addresses are <alice@example.com/work> and <bob@example.com/work>.
Note: A service MAY contain several queues to help organize, route and handle incoming user chat requests. Implementations supporting multiple queues in a workgroup will respond differently to requests, and send different status information for each queue. Workgroup queues are identified by a unique resource name: e.g. support@workgroup.example.com/platinum-plan or support@workgroup.example.com/xmpp-products. Implementations should gracefully handle services with only one queue (using support@workgroup.example.com) or multiple queues. Users should only be aware of one workgroup (users should never see workgroup queue resource names).

[2] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[3] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[4] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

2.2 Responsibilities

Each participant is responsible for certain behaviors in the workgroup protocol.
Users should:

- Know the status of the workgroup queue before requesting a conversation. This information allows users to see if the workgroup is available, and how long a wait they may have before a chat is initiated.
- Know the status of their request while in the request queue.
- Be able to cancel a chat request at any time.

Workgroup agents should:

- Know the status of the workgroup queue(s).
- Be able to accept or reject chat requests.
- Indicate their availability for handling workgroup chats.

The workgroup service:

- Controls the workgroup request queue(s).
- Manages the updating of queue status information.
- Determines how users are queued and how queue requests are routed to workgroup members. The queue routing algorithm is beyond the scope of this document and left to implementers (simple round-robin, priority based, rules based, etc).
- Maintains its presence, indicating the availability of the workgroup service.

3 User Protocol

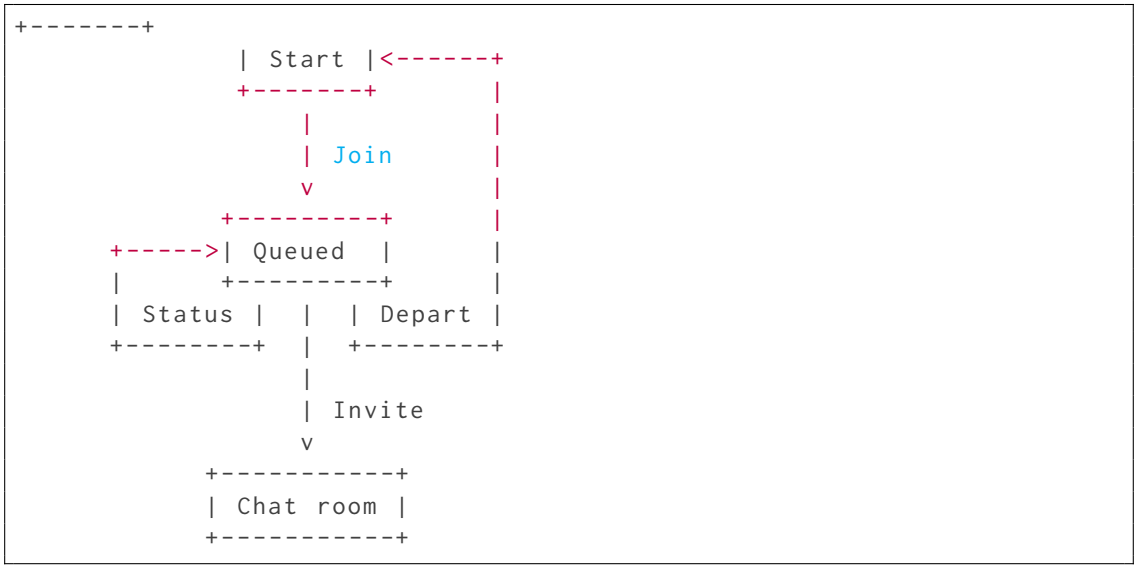
The workgroup protocol consists of several XMPP packet exchanges that occur during the lifetime of the protocol. These packet exchanges change the state of the relationship between user, agent and service.

3.1 User States

Users join the workgroup queue to wait for a chat with an agent. Once they have joined the queue, they may receive zero or more status updates from the workgroup service informing them of their status in the queue. Users have the option to cancel their chat request at any time.
When an agent is ready to chat with the user, the user MUST be sent a standard XMPP groupchat invitation to a chat room. Receipt of the invitation indicates that the user is no longer in the queue and that they should join the chat room using the standard XMPP groupchat protocol in order to converse with an agent. Groupchat is used because it offers several advantages in workgroup conversations including:

- Allowing more than one agent to join the conversation (useful for bringing in experts to join the conversation).
- Allowing managers to monitor conversations for quality of service.
- Creating a simple way of determining what is in a 'conversation' for logging and gathering other statistical information about the conversation.
- Allowing a convenient mechanism for bringing 'chatbot' services into the conversation (e.g. answering FAQs).

The user's states and packet exchanges that cause state transitions are shown below:



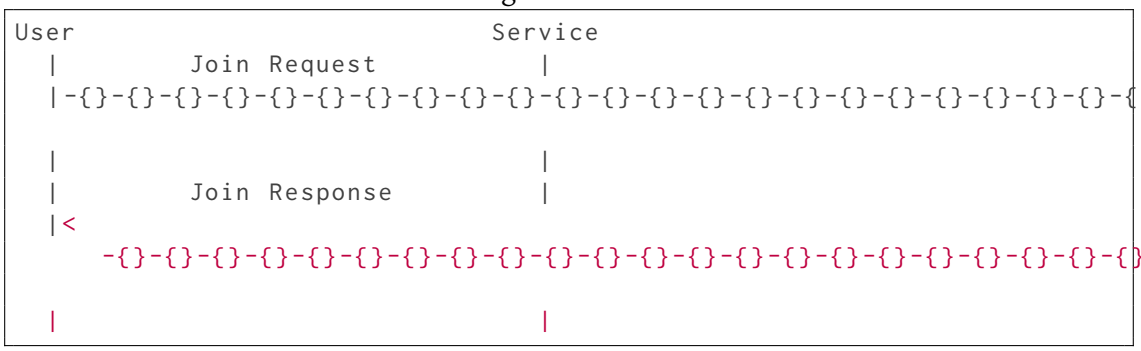
3.2 User Packet Exchanges

Packets are exchanged between the user and service to trigger state changes in the user. These packet exchanges are described next.

3.2.1 User Join Protocol

This section describes the packet exchange allowing users to join a workgroup service queue. This protocol MUST be supported by compliant implementations.

Listing 1: Transactions



The user sends a join request to the workgroup service in order to join the workgroup queue. The workgroup service may either accept or reject the request. A user session (e.g. user@example.net/home) may have only one active join request. Subsequent, simultaneous joins MUST result in an error.
Some workgroups require that the user submit certain information before the user is allowed to join. In these cases, the workgroup MUST reject the initial join request with a <not-acceptable/> error. The user should then use the Data Forms protocol within iq-join-queue to obtain a form, and submit it to join the queue.

Listing 2: Request Element

```
U: <iq to='support@workgroup.example.com' from='user@example.net/home'
     id='id1' type='set'>
U:    <join-queue xmlns='http://jabber.org/protocol/workgroup'>
U:      <queue-notifications/>
U:    </join-queue>
U: </iq>
```

The request may contain application-specific metadata to help the service determine queuing of the user or Data Forms data when submitting form information (definition of such data is out of scope for this document). In addition, the <join-queue> element MAY contain a standard <queue-notifications/> element, which indicates that the user would like to receive user status updates about their state in the queue.
A successful join results in a success response:

Listing 3: Response Element

```
S: <iq to='user@example.net/home' from='support@workgroup.example.com'
     id='id1' type='result'/>
```

If the user indicated interest in their queue status information, the supported status updates MUST be sent by the server. Compliant implementations do not have to support any status update types. Status updates requested by the user and supported by the server MUST be pushed to the user by the service until the user departs or is invited to a chat room.

Condition	Description
<not-authorized/>	The user is not authorized to join the queue. A determination of who has permission to join a queue is left to implementations.
<item-not-found/>	The address the user requests a chat with does not exist or is not a workgroup. Compliant workgroup service implementations MUST NOT return this error if the requested address is a valid workgroup.
<not-acceptable/>	The user must submit valid form data before joining the queue. Note that this error is sent when the user tries to join, or if the user submits form data that is not filled out correctly.
<conflict/>	The user has already joined the queue.
<service-unavailable/>	The workgroup is valid but not accepting new join-queue requests.

The following protocol flows show an example of a user successfully joining a workgroup queue for support@workgroup.example.com.

Listing 4: Successful Join

```
U: <iq type='set'
U:     from='user@example.net/home'
U:       to='support@workgroup.example.com'
U:       id='id1'>
U:   <join-queue xmlns='http://jabber.org/protocol/workgroup'>
U:     <queue-notifications/>
U:   </join-queue>
U: </iq>
S: <iq type='result'
S:     from='support@workgroup.example.com'
S:     to='user@example.net/home'
S:     id='id1'/>
```

The following XML is another example where metadata is sent by the user to assist the workgroup server in queuing and routing (naturally, the custom namespace that qualifies the <crm/> element in this example would be defined outside the context of this specification).

Listing 5: Join With Meta-Data

```
U: <iq type='set'
U:     from='user@example.net/home'
U:       to='support@workgroup.example.com'
U:       id='id2'>
U:   <join-queue xmlns='http://jabber.org/protocol/workgroup'>
U:     <crm xmlns='http://www.example.com/xmpp/workgroup'>
U:       <customer-id>the24th498onth</customer-id>
U:       <referrer>
U:         http://www.example.com/portal/
U:       </referrer>
U:       <product>Widget 1.0</product>
U:     </crm>
U:     <queue-notifications/>
U:   </join-queue>
U: </iq>
S: <iq type='result'
S:     from='support@workgroup.example.com'
S:       to='user@example.net/home'
S:       id='id2'/>
```

Finally an example of a required form submission before a user is allowed to the workgroup queue for support@workgroup.example.com. The data form in this example is trivial; please see Data Forms (XEP-0004) [5] for a complete data form example. The example begins as normal, but the workgroup returns a <not-acceptable/> error.

Listing 6: Join With Form

```
U: <iq type='set'
U:     from='user@example.net/home'
U:       to='support@workgroup.example.com'
U:       id='id1'>
U:   <join-queue xmlns='http://jabber.org/protocol/workgroup'>
U:     <queue-notifications/>
U:   </join-queue>
U: </iq>
S: <iq type='error'
S:     from='support@workgroup.example.com'
S:       to='user@example.net/home'
S:       id='id1'>
S:   <error code='406' type='modify'>
S:     <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
S:   </error>
S: </iq>
```

The <not-acceptable/> error indicates that a data form is required. The user requests the required data form from the workgroup.

[5] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

Listing 7: Join With Form (2)

```
U: <iq type='get'
U:     from='user@example.net/home'
U:       to='support@workgroup.example.com'
U:       id='id2'>
U:   <join-queue xmlns='http://jabber.org/protocol/workgroup'/>
U: </iq>
S: <iq type='result'
S:     from='support@workgroup.example.com'
S:       to='user@example.net/home'
S:       id='id2'>
S:   <join-queue xmlns='http://jabber.org/protocol/workgroup'>
S:     <x xmlns='jabber:iq:data' type='form'>
S:       <title>Support.com Chat Customer Information</title>
S:       <instructions>Welcome to example.com! Please provide us with
S:         some information about yourself so we can serve you better.
S:       </instructions>
S:       <field type='text-single' label='First␣Name' var='first' />
S:       <field type='text-single' label='Last␣Name' var='last' />
S:       <field type='list-single' label='Contract␣Type'
var='contract_type'>
S:         <value>0</value>
S:         <option label='None'><value>0</value></option>
S:         <option label='Bronze'><value>1</value></option>
S:         <option label='Silver'><value>2</value></option>
S:         <option label='Gold'><value>3</value></option>
S:       </field>
S:     </x>
S:   </join-queue>
S: </iq>
```

After presenting the form to the user and gathering the form data, the user submits the form data to the workgroup and the workgroup accepts it. The user is now in the queue.

Listing 8: Join With Form (3)

```
U: <iq type='set'
U:     from='user@example.net/home'
U:       to='support@workgroup.example.com'
U:       id='id3'>
U:   <join-queue xmlns='http://jabber.org/protocol/workgroup'>
U:     <queue-notifications/>
U:     <x xmlns='jabber:iq:data' type='submit'>
U:       <field var='first'><value>John</value></field>
U:       <field var='last'><value>Doe</value></field>
U:       <field var='contract_type'><value>2</value></field>
U:     </x>
```

```
U:    </join-queue>
U:  </iq>
S:  <iq type='result'
S:      from='support@workgroup.example.com'
S:        to='user@example.net/home'
S:        id='id3' />
```

3.2.2 User Depart Protocol

This section describes the packet exchange allowing users to depart a workgroup service queue, or for a workgroup service to remove a user from the workgroup queue. This protocol MUST be supported by compliant implementations.
The user no longer wishes to be in the queue and issues a depart queue command.

Listing 9: Transactions

```
Requester                          Service
  |         Depart Request           |
  |-{}-{}-{}
```

Listing 12: Depart Request With JID

```
U: <iq from='admin-jid' to='support@workgroup.example.com' id='id1'
   type='set'>
U:   <depart-queue xmlns='http://jabber.org/protocol/workgroup'>
U:     <jid>user@example.net/home</jid>
U:   </depart-queue>
U: </iq>
```

It is expected that implementations will determine who is allowed to remove other users from the queue based on an implementation specific permissions model. These administrator depart requests may result in <not-authorized/> errors (see error section). A user removing their own queue entry MUST NOT receive unauthorized errors (the workgroup service MUST NOT prevent a user from departing the queue).
The sender of the depart request receives a successful result packet:

Listing 13: Depart Request

```
S: <iq from='support@workgroup.example.com' to='user@example.net/home'
    id='id1' type='result'/>
```

And the user who is departing receives a depart message (the user may not have been the sender of the request):

Listing 14: Depart Message

```
S: <message from='support@workgroup.example.com' to='user@example.net/
   home'>
S:    <depart-queue xmlns='http://jabber.org/protocol/workgroup'/>
S: </message>
```

The user will not be in the queue after a response is received unless the error response code is <not-authorized/>.

Condition	Description
<not-authorized/>	The sender did not have permission to remove the user from the queue. This error code MUST NOT be used when a user is removing their queue entry.
<item-not-found/>	The user was not in the queue.

A user leaves the workgroup queue support@workgroup.example.com.

Listing 15: User Departs

```
U: <iq from='user@example.net/home'
U:       to='support@workgroup.example.com'
U:       id='id1'
U:     type='set'>
U:   <depart-queue xmlns='http://jabber.org/protocol/workgroup'/>
U: </iq>
S: <iq from='support@workgoup.example.com'
S:     to='user@example.net/home'
S:     id='id1'
S:     type='result'/>
S: <message from='support@workgroup.example.com' to='user@example.net/
   home'>
S:    <depart-queue xmlns='http://jabber.org/protocol/workgroup'/>
S: </message>
```

An administrator removes a user from the workgroup queue support@workgroup.example.com. Notice that the depart-queue message is sent to the user that has left the queue.

Listing 16: Administrator Removes User

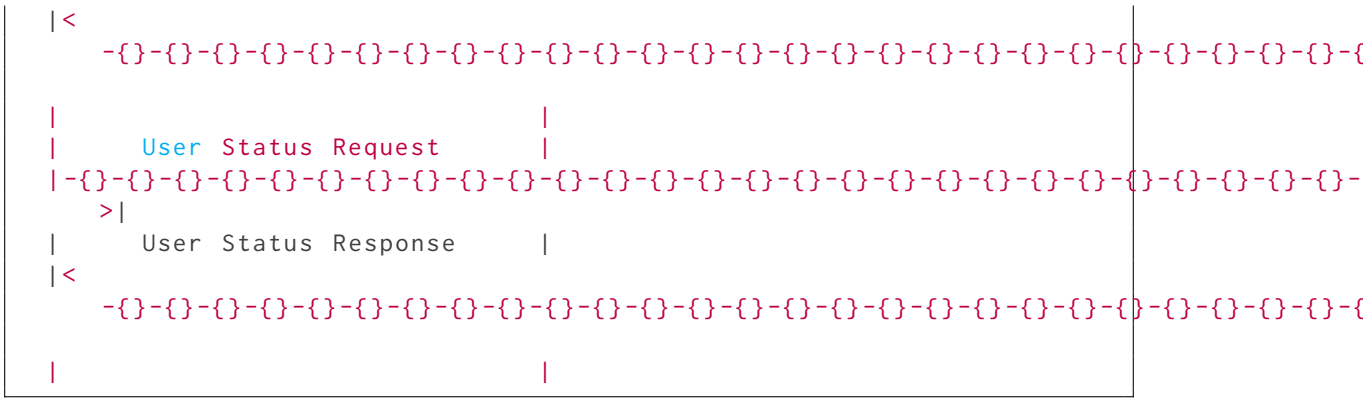
```
U: <iq from='admin@example.com/work'
U:       to='support@workgroup.example.com'
U:       id='id1'
U:     type='set'>
U:   <depart-queue xmlns='http://jabber.org/protocol/workgroup'>
U:     <jid>user@example.net/home</jid>
U:   </depart-queue>
U: </iq>
S: <iq from='support@workgroup.example.com'
S:     to='admin@example.com/work'
S:     id='id1'
S:     type='result'/>
S: <message from='support@workgroupexample.com' to='user@example.net/
   home'>
S:    <depart-queue xmlns='http://jabber.org/protocol/workgroup'/>
S: </message>
```

3.2.3 User Status Update Protocol

This section describes the packet exchange for updating users on their queue status. This protocol MAY be supported by compliant implementations.

Listing 17: Transactions

```
User                          Service
  |      User Status Push        |
```



The workgroup service pushes updates to the user as their queue status changes. Furthermore, the user may request their queue status at any time.
User status updates are contained in a <queue-status/> element that updates the user on their queue position and estimated time. The position contained in a <position> sub-element is a non-negative integer indicating the number of queue entries that must be routed to an agent before the user is routed to an agent. A position of 0 (zero) indicates the user is currently being routed. Clients may use this information to display the current queue position to the user.
The queue time status is contained in a <time/> sub-element that updates the user with the estimated time until they will be routed to an agent. The time is a non-negative integer indicating the estimated number of seconds remaining before being routed. Services should send this update at regular intervals. We recommend every 15 seconds, but the best solution will depend on application dependent factors and the service may decide to send updates at any interval or never (relying on the client to request the information). User clients should assume the estimated time counts down at a rate of one per second between status updates. Clients may use this information to display the running estimated time to the user.
A server 'push' occurs asynchronously to client:

Listing 18: User Status Push

```
S: <message to='user@example.net/home' from='support@workgroup.example
   .com'>
S:   <queue-status xmlns='http://jabber.org/protocol/workgroup'>
S:     <position>4</position>
S:     <time>60</time>
S:   </queue-status>
S: </message>
```

Alternately the client may poll their position:

Listing 19: User Status Poll

```
U: <iq to='support@workgroup.example.com' from='user@example.net/home'
    id='id1' type='get'>
```

```
U:   <queue-status xmlns='http://jabber.org/protocol/workgroup'/>
U: </iq>
S: <iq to='user@example.net/home' from='support@workgroup.example.com'
     id='id1' type='result'>
S:   <queue-status xmlns='http://jabber.org/protocol/workgroup'>
S:     <position>4</position>
S:     <time>60</time>
S:   </queue-status>
S: </iq>
```

Condition	Description
<not-authorized/>	Sent by the server to the user in response to a status query only if the user is not a member of the queue.
<feature-not-implemented/>	Sent only if status updates are not implemented in either the client or server.

3.2.4 User Invite Protocol

This section describes the packet exchange for inviting a queued user to a chat room for conversation with an agent. This protocol MUST be supported by compliant implementations.

Listing 20: Transactions

```
User                              Service
  |          User Invite            |
  |<
     -{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{
  |                                 |
```

The server sends an invitation to the user to begin their conversation with an agent, structured according to the format defined in Multi-User Chat (XEP-0045) [6]. The 'from' attribute of the <invite/> element MUST be set to the JID of the workgroup. The invitation indicates that the user is no longer in the workgroup queue. The user MUST NOT receive any more user queue status updates once they receive an invitation.
There are no defined error conditions for user invitations.
An invitation from the server on behalf of the support@example.net workgroup:

Listing 21: An Invitation

```
S: <message
S:     from='roomname@chatserver.example.com'
```

[6] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

```
S:      to='user@example.net/home'>
S:    <x xmlns='http://jabber.org/protocol/muc#user'>
S:      <invite from='support@workgroup.example.com'>
S:        <reason>
S:         You have been invited to chat with a support@workgroup.
    example.com
           agent.
S:        </reason>
S:      </invite>
S:    </x>
S: </message>
```

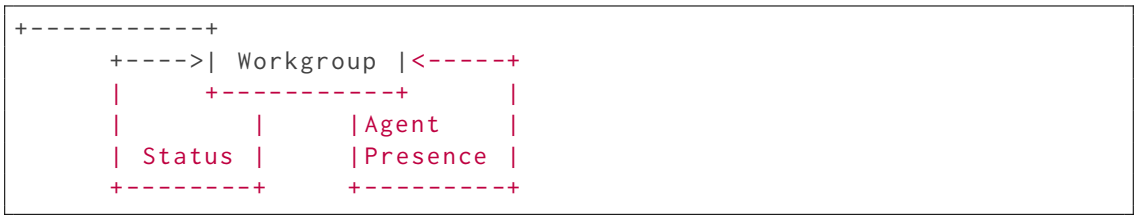
4 Agent Protocol

4.1 Agent States

Agents join a workgroup to indicate they are capable of handling conversations with users. Agent membership in the workgroup is expected to be a long term, persistent relationship similar to roster membership. For example, a customer support agent may join the support@workgroup.example.com workgroup when they begin working at example.com and will only depart when they leave that position. The wide variety of relationships, processes and permissions associated with joining and leaving workgroups lies outside the scope of this document.

Once an agent has joined a workgroup they will receive workgroup status updates to inform them of the status of other members of the workgroup. Agents are responsible for updating the workgroup service with their presence so the service can intelligently route chat requests to the 'best' agent. Workgroup agent presence uses standard XMPP presence packets with optional metadata to help routing of chat requests to agents. Some metadata will be standard and defined later in this document. It is expected that other deployment specific metadata will also be needed to make routing decisions.

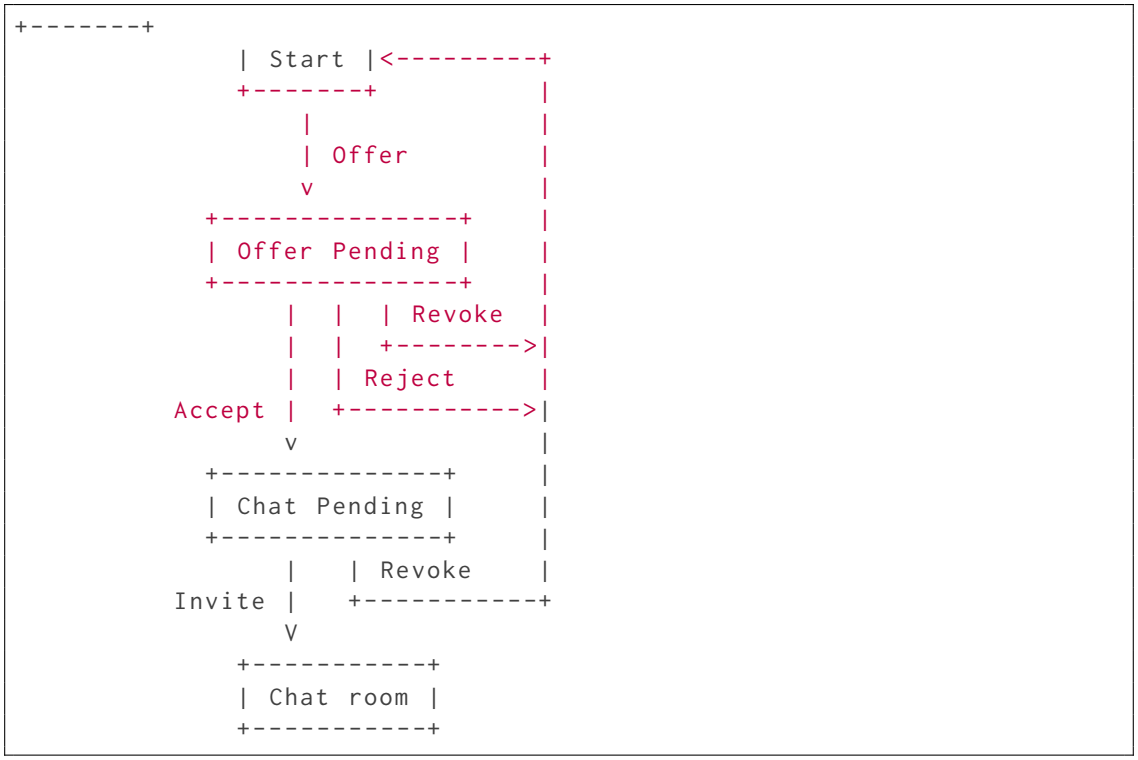
The general agent workgroup state diagram is shown below:



Once an agent has joined a workgroup and is available, the agent will receive offers to chat with users by the workgroup service. Chat offers will be made to the agent and the agent has the opportunity to accept or reject each offer. The workgroup service may also revoke an offer. For example, a service may revoke chat offers if the offer is not responded to within a certain period of time to ensure fast responses to user chat requests.

Once an offer has been accepted, the agent must wait for a standard groupchat invitation

from the workgroup service. The workgroup service may revoke the offer at this stage of the protocol as well. This enables workgroup services to send offers to several agents in parallel, and choose the 'best' agent that accepts. A diagram showing the agent workgroup sub-states and transitions is shown below:



4.2 Agent Packet Exchanges

Packets are exchanged between the agent and service to trigger state changes in the agent client. These packet exchanges are described next.

4.2.1 Agent Presence Protocol

This section describes the packet exchange allowing agents to update a workgroup with their current presence. This protocol MUST be supported by compliant implementations.

Listing 22: Transactions

```
Agent                              Service
  |          Presence Update          |
  |-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-
```


The agent must inform the workgroup of its presence by sending it a directed (not broadcast) presence update packet. Agent presence updates use standard XMPP presence with optional metadata. However, there must always be an agent-status workgroup sub-element in the presence packet to indicate that the presence update relates to agent workgroup presence. Agent workgroup presence is designed to allow a separation between the agent's normal XMPP presence (server-managed via rosters) and their presence with the workgroup.

Listing 23: Presence Update

```
U: <presence from='alice@example.com/work' to='support@workgroup.
   example.com'>
U:   <agent-status xmlns='http://jabber.org/protocol/workgroup'>
U:     <max-chats>count</max-chats>
U:   </agent-status>
U: </presence>
```

Agent presence updates use standard XMPP presence packets and should contain the normal sub elements as needed (e.g. <show/>, <status/>, etc) and can be of type='unavailable' to indicate the agent is not available for workgroup routing or for receiving workgroup agent updates. The standard XMPP show states have specific meaning within the context of the workgroup protocol:

- chat - Indicates the agent is available to chat (is idle and ready to handle more conversations).
- away - The agent is busy (possibly with other chats). The agent may still be able to handle other chats but an offer rejection is likely.
- xa - The agent is physically away from their terminal and should not have a chat routed to them.
- dnd - The agent is busy and should not be disturbed. However, special case, or extreme urgency chats may still be offered to the agent although offer rejection or offer timeouts are highly likely.

Agents MAY also embed metadata to help the workgroup service route chat requests, using the <max-chats> element, which specifies the maximum number of chats the agent can handle. If a presence is sent to the workgroup that does not contain the max-chats value, the "default setting" will be assumed. The value of the default setting for an agent is up to an implementation. [7]

[7] The max-chats value sent from agent to workgroup service is a 'hint' or recommended value. The workgroup service is not obliged to accept this value. The actual max-chats value for the agent will be sent to the agent via the next Agent Status Update. This allows administrators to constrain agent behavior in order to enforce company policy, quality assurance, etc.

There are no defined error conditions for presence updates.
An agent (alice) becomes available to the workgroup support@workgroup.example.com.

Listing 24: Agent Becomes Available

```
U: <presence from='alice@example.com/work'
U:              to='support@workgroup.example.com'>
U:   <show>chat</show>
U:   <agent-status xmlns='http://jabber.org/protocol/workgroup'>
U:     <max-chats>3</max-chats>
U:   </agent-status>
U: </presence>
```

4.2.2 Workgroup Status Update Protocol

This section describes the packet exchange used to update agents on the status of the workgroup. This protocol MAY be supported by compliant implementations.
After an agent announces their presence to the workgroup, they will begin receiving presence updates from the workgroup. All fields are optional:

Listing 25: Notify-Agent Status Type

```
S: <presence to='alice@example.com/work' from='support@workgroup.
   example.com'>
S:   <notify-agents xmlns='http://jabber.org/protocol/workgroup'>
S:     <available>count</available>
S:     <current-chats>count</current-chats>
S:     <max-chats>count</max-chats>
S:   </notify-agents>
S: </presence>
```

The defined sub-elements of <notify-agents> are:

- <available> - The total number of agents available in the workgroup.
- <current-chats> - The current total number of chats being handled by agents in the workgroup.
- <max-chats> - The maximum number of simultaneous conversations that can be handled by agents in the workgroup.

There are no defined error conditions for notify workgroup updates.
An agent (alice) receives an update from workgroup support@workgroup.example.com.

Listing 26: Agent Recives Update

```
S: <presence to='alice@example.com/work' from='support@wokgroup.
   example.com'>
S:   <notify-agents xmlns='http://jabber.org/protocol/workgroup'>
S:     <available>2</available>
S:     <current-chats>2</current-chats>
S:     <max-chats>7</max-chats>
S:   </notify-agents>
S: </presence>
```

4.2.3 Queue Status Update Protocol

This section describes the packet exchange used to update agents on the status of the workgroup queue. This protocol MAY be supported by compliant implementations.
After an agent announces their presence to the workgroup, they will begin receiving presence updates from the workgroup with an overview and details on the queue status.
The <notify-queue/> element updates the agent with a summary of the status of the workgroup queue. All fields are optional:

Listing 27: Notify-Queue Status Type

```
S: <presence to='alice@example.com/work' from='support@workgroup.
   example.com'>
S:   <notify-queue xmlns='http://jabber.org/protocol/workgroup'>
S:     <count>count</count>
S:     <oldest>YYYY-MM-DDTHH:mm:ss</oldest>
S:     <time>average-time-to-chat</time>
S:     <status>open</status>
S:   </notify-queue>
S: </presence>
```

The defined sub-elements of <notify-queue> are:

- <count> - The total number of users in the workgroup queue.
- <oldest> - The date and time when the oldest member of the queue joined (MUST conform to the DateTime profile defined in XMPP Date and Time Profiles (XEP-0082) [8]).
- <time> - The average time in seconds that a user is in the queue before they are routed to an agent for handling.
- <status> - The status of the queue. Queues may be active (requests are being routed and handled by agents) but not accepting new requests for handling. Typical reasons for this state include the queue is shutting down but finishing processing users in the queue, or

[8] XEP-0082: XMPP Date and Time Profiles <https://xmpp.org/extensions/xep-0082.html>.

because the queue has too many requests and should not accept more request until the existing requests are handled. The status field MUST contain one of the following values:

- open - the queue is active and accepting new chat requests
- active - the queue is active but NOT accepting new chat requests
- closed - the queue is NOT active and NOT accepting new chat requests

The <notify-queue-details/> element updates the agent with details of the workgroup queue. All fields are optional:

Listing 28: Notify-Queue-Details Status Type

```
S: <presence to='alice@example.com/work' from='support@workgroup.
   example.com'>
S:   <notify-queue-details xmlns='http://jabber.org/protocol/workgroup
   '>
S:     <user jid='user@example.net/home'>
S:       <position>pos</position>
S:       <time>estimated-time</time>
S:       <join-time>YYYY-MM-DDTHH:mm:SS</join-time>
S:     </user>
S:   </notify-queue-details>
S: </presence>
```

An update may contain one or more <user> entries (one per user in the queue). The defined sub-elements of <user> are:

- <position> - The user's zero-based position in the queue.
- <time> - Estimated time in seconds remaining before the user is routed to an agent.
- <join-time> - The datetime when the user joined the queue (MUST conform to the Date-Time profile defined in XMPP Date and Time Profiles (XEP-0082) [9]).

There are no defined error conditions for workgroup queue status updates.
An agent receives an update from workgroup support@workgroup.example.com.

Listing 29: Agent Receives Updates

```
S: <presence to='alice@example.com/work' from='support@workgroup.
   example.com'>
S:   <notify-queue xmlns='http://jabber.org/protocol/workgroup'>
S:     <count>1</count>
S:     <oldest>20050208T10:00:00</oldest>
S:     <time>30</time>
S:     <status>open</status>
```

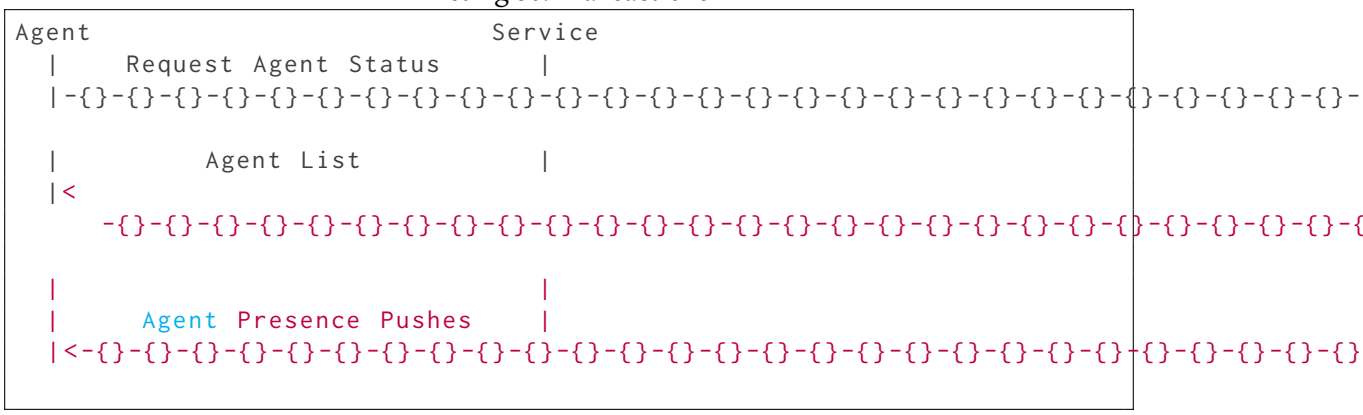
[9] XEP-0082: XMPP Date and Time Profiles <https://xmpp.org/extensions/xep-0082.html>.

```
S:    </notify-queue>
S:  </presence>
S:  <presence to='alice@example.com/work' from='support@workgroup.
    example.com'>
S:    <notify-queue-details xmlns='http://jabber.org/protocol/workgroup
    '>
S:      <user jid='user@example.net/home'>
S:        <position>1</position>
S:        <time>5</time>
S:        <join-time>20050208T10:00:00</join-time>
S:      </user>
S:    </notify-queue-details>
S:  </presence>
```

4.2.4 Agent Status Update Protocol

This section describes the packet exchange used to update agents on the status of other agents in the workgroup. This protocol MAY be supported by compliant implementations.

Listing 30: Transactions



The workgroup service pushes presence updates to the agent as the presence of other agents changes. This will only occur after an agent has requested to receive other agents' information. The server will continue to send presence updates until the agent sends an unavailable presence to the server. This protocol is similar to the standard XMPP roster workflow.
To receive presence updates for other agents in the workgroup, the agent sends an agent info request to the workgroup:

Listing 31: Request Element

```
U:  <iq to='support@workgroup.example.com' from='alice@example.com/work
    '
U:      id='id1' type='get'>
U:    <agent-status-request xmlns='http://jabber.org/protocol/workgroup
    '/>
```

```
U: </iq>
```

The workgroup will then reply with a list of all agents in the workgroup (excluding the agent making the request):

Listing 32: Response Element

```
S: <iq to='alice@example.com/work' from='support@workgroup.example.com
   '
S:     id='id1' type='result'>
S:   <agent-status-request xmlns='http://jabber.org/protocol/workgroup
   '>
S:     <agent jid='bob@example.com' />
S:   </agent-status-request>
S: </iq>
```

The server will then push presence packets for other agents as their presence changes. All fields in the <agent-status> child stanza are optional, but an <agent-status> child stanza must be present:

Listing 33: Agent Status Update

```
S: <presence to='alice@example.com/work' from='bob@example.com/work'>
S:   <agent-status xmlns='http://jabber.org/protocol/workgroup'>
S:       <current-chats>2</current-chats>
S:       <max-chats>4</max-chats>
S:   </agent-status>
S: </presence>
```

The defined sub-elements of <agent-status> are:

- <current-chats> - The number of conversations currently being handled by the agent.
- <max-chats> - The maximum number of simultaneous conversations the agent can handle.

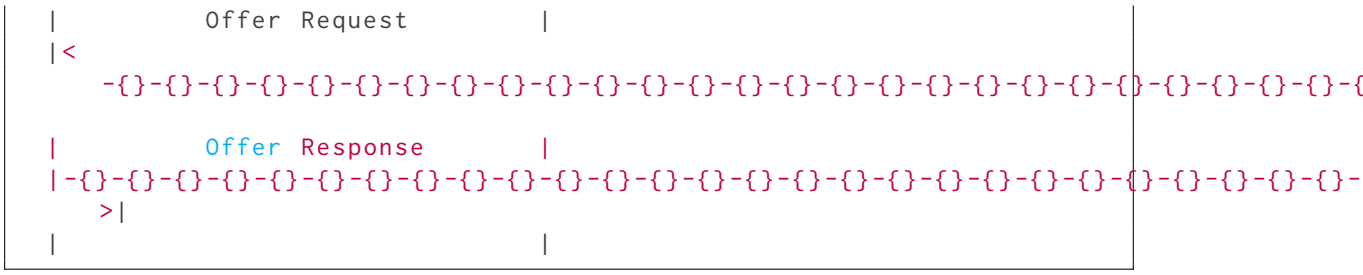
There are no defined error conditions for agent status updates.

4.2.5 Agent Offer Protocol

This section describes the packet exchange involved in a service offering a chat to an agent. This protocol MUST be supported by compliant implementations.

Listing 34: Transactions

```
Agent                           Service
```



The agent is offered a chat with a user. A successful offer results in the agent owning the offer, but does not mean it has accepted the chat. Accepting an offer is handled by the Agent Accept protocol. The separation between offer and acceptance is made so that agents may receive offers while engaged in other activities (busy with other chats) and accept them at a later time.

Listing 35: Offer Request

```
S: <iq from='support@workgroup.example.com' to='alice@example.com/work
   ' id='id1' type='set'>
S:   <offer xmlns='http://jabber.org/protocol/workgroup' jid='
   user@example.net/home'>
S:     <timeout>seconds</timeout>
S:   </offer>
S: </iq>
```

Application specific metadata will normally be added as a sub-element of <offer> to help agents decide whether to accept or not (formats for which are out of scope for this document). An optional <timeout> sub-element may be included indicating the amount of time the offer stands before the service will revoke it.

Listing 36: Offer Response

```
A: <iq from='alice@example.com/work' to='support@workgroup.example.com
   ' id='id1' type='result'/>
```

The agent may respond only with a successful result.
There are no defined error conditions for an offer response.
An agent is offered a chat with a user. The offer will be revoked in 30 seconds.

Listing 37: Agent is Offered a Chat

```
S: <iq to='alice@example.com/work'
S:     from='support@workgroup.example.com'
S:     id='id1'
S:     type='set'>
S:   <offer xmlns='http://jabber.org/protocol/workgroup' jid='
   user@example.net/home'>
S:     <timeout>30</timeout>
S:   </offer>
```

```
S: </iq>
A: <iq to='support@workgroup.example.com'
A:     from='alice@example.com/work'
A:     id='id1'
A:     type='result'/>
```

The following is a more typical offer containing metadata about the user. The offer will be revoked in 30 seconds.

Listing 38: Offer Including Meta-Data

```
S: <iq to='alice@example.com/work'
S:     from='support@workgroup.example.com'
S:     id='id2'
S:     type='set'>
S:   <offer xmlns='http://jabber.org/protocol/workgroup' jid='
   user@example.net/home'>
S:     <timeout>30</timeout>
S:     <crm xmlns='http://www.example.com/xmpp/workgroup'>
S:       <user-id>423498ae84f</user-id>
S:       <product>Widget 1.0</product>
S:     </crm>
S:   </offer>
S: </iq>
A: <iq to='support@workgroup.example.com'
A:     from='alice@example.com/work'
A:     id='id2'
A:     type='result'/>
```

4.2.6 Agent Offer Accept/Reject Protocol

This section describes the packet exchange involved in an agent rejecting an offering a chat to a user. This protocol MUST be supported by compliant implementations.

Listing 39: Transactions

```
Agent                          Service
  |  Offer Accept/Reject Request |
  |-{}-{}-{}
```

Listing 40: Offer Accept Request

```
A: <iq to='support@workgroup.example.com' from='alice@example.com/work
   ' id='id1' type='set'>
A:   <offer-accept jid='user@example.net/home' xmlns='http://jabber.
   org/protocol/workgroup' />
A: </iq>
```

Listing 41: Offer Reject Request

```
A: <iq to='support@workgroup.example.com' from='alice@example.com/work
   ' id='id1' type='set'>
A:   <offer-reject jid='user@example.net/home' xmlns='http://jabber.
   org/protocol/workgroup'/>
A: </iq>
```

Listing 42: Offer Response

```
S: <iq to='alice@example.com/work' from='support@workgroup.example.com
   ' id='id1' type='result'/>
```

The service may respond only with a successful result.
There are no defined error conditions for an accept/reject offer response.

Listing 43: Agent Accepts Chat

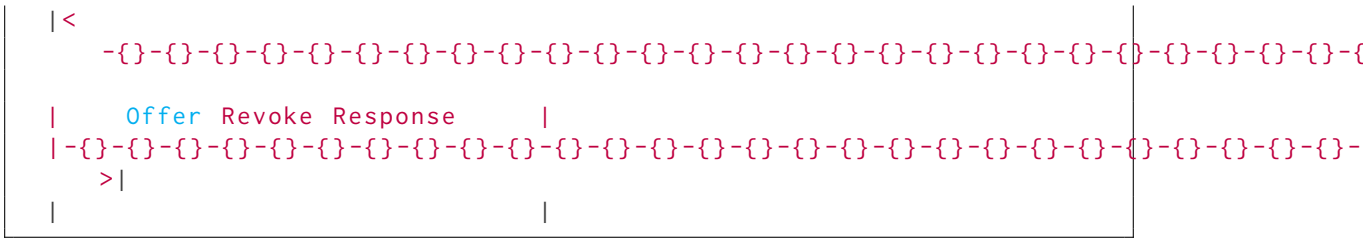
```
A: <iq from='alice@example.com/work'
A:       to='support@workgroup.example.com'
A:       id='id3'
A:     type='set'>
A:   <offer-accept jid='user@example.net/home' xmlns='http://jabber.
   org/protocol/workgroup'/>
A: </iq>
S: <iq from='support@workgroup.example.com'
S:     to='alice@example.com/work'
S:     id='id3'
S:     type='result'/>
```

4.2.7 Agent Offer Revoke Protocol

This section describes the packet exchange involved in a service revoking an offer to an agent to chat to a user. This protocol MUST be supported by compliant implementations.

Listing 44: Transactions

```
Agent                          Service
  |      Offer Revoke Request      |
```



The service revokes an earlier offer to chat to a user. Offer revocations typically occur when the original offer times out, or a better agent was found to handle the chat. Note that offer revocations may occur anytime after an offer has been made, and before an invitation is sent (see agent state diagram). In other words, even though an agent has accepted an offer to chat, the agent may still receive an offer revocation (e.g. a better agent was found to handle the chat).

Listing 45: Offer Revoke Request

```
S: <iq from='support@workgroup.example.com' to='alice@example.com/work
   ' id='id1' type='set'>
S:   <offer-revoke jid='user@example.net/home' xmlns='http://jabber.
   org/protocol/workgroup'>
S:     <reason>
S:       [natural-language text]
S:     </reason>
S:   </offer-revoke>
S: </iq>
```

The reason element may optionally contain free form text explaining the reason the offer was revoked.

Listing 46: Offer Response

```
A: <iq from='alice@example.com/work' to='support@workgroup.example.com
   ' id='id1' type='result'/>
```

The agent may respond only with a successful result.
There are no defined error conditions for an offer response.

Listing 47: Offer Revoked Due to Timeout

```
S: <iq to='alice@example.com/work'
S:     from='support@workgroup.example.com'
S:     id='id4'
S:     type='set'>
S:   <offer-revoke xmlns='http://jabber.org/protocol/workgroup' jid='
   user@example.net/home'>
S:       <reason>
S:         Offer timed out
```

```
S:        </reason>
S:    </offer-revoke>
S: </iq>
A: <iq to='support@workgroup.example.com'
A:     from='alice@example.com/work'
A:       id='id4'
A:     type='result'/>
```

4.2.8 Agent Invite Protocol

This section describes the packet exchange inviting an agent to a chat room for conversation with a user. This protocol MUST be supported by compliant implementations.

Listing 48: Transactions

```
Agent                              Service
  |          Agent Invite            |
  |<
     -{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{}-{
  |                                  |
```

The server sends an invitation to the agent to begin their conversation with the user, structured according to the format defined in Multi-User Chat (XEP-0045) [10]. The 'from' attribute of the <invite/> element MUST be set to the JID of the workgroup.
In order to match invitations to offers, all invitations SHOULD include metadata in the <offer/> element, with the JID of the user specified via the 'jid' attribute. The typical metadata fragment would appear as:

Listing 49: Invitation Meta-Data

```
<offer xmlns='http://jabber.org/protocol/workgroup' jid='user@example.
   net/home'>
```

There are no defined error conditions for agent invitations.
An invitation from the server on behalf of the support@example.net workgroup:

Listing 50: An Invitation

```
S: <message
S:     from='roomname@chatserver.example.com'
S:     to='alice@example.com/work'>
S:   <x xmlns='http://jabber.org/protocol/muc#user'>
S:     <invite from='support@workgroup.example.com'>
S:       <reason>
```

[10] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

```
S:       Please join the chat room to start your chat with
    user@example.net.
S:       </reason>
S:     </invite>
S:   </x>
S:   <offer xmlns='http://jabber.org/protocol/workgroup' jid='
    user@example.net/home'/>
S: </message>
```

5 Service Discovery

Service Discovery support is optional for Workgroup compliant implementations. Workgroup services that do support Service Discovery MUST:

- Use the <identity> category='collaboration'.
- Use the <identity> type='workgroup'.
- Use the <feature> var='http://jabber.org/protocol/workgroup'.

An example of discovery browsing is included. Notice how probing starts at the server (example.com) revealing the workgroup service by its JID (workgroup.example.com) and a simple, human friendly name ("Example.com Live Assistant"). It is only during the discovery probing of the service that it is identified as a workgroup using the <identity> and <feature> tags. Finally individual workgroups (support and sales) can be discovered on the Workgroup service. When individual workgroups are probed, the <identity> and <feature> tags are again presented to identify them as workgroups along with (optional) associated metadata.

Listing 51: Workgroup Service Discovery

```
U: <iq to='example.com' from='user@example.net/home' id='id1' type='
    get'>
U:     <query xmlns='http://jabber.org/protocol/disco#items'/>
U: </iq>
S: <iq from='example.com' to='user@example.net/home' id='id1' type='
    result'>
S:    <query xmlns='http://jabber.org/protocol/disco#items'>
S:        <item jid='workgroup.example.com' name='Example.com␣Live␣
    Assistant'/>
S:    </query>
S: </iq>

U: <iq to='workgroup.example.com' from='user@example.net/home' id='id2
    ' type='get'>
U:    <query xmlns='http://jabber.org/protocol/disco#info'/>
U: </iq>
```

```
S: <iq from='workgroup.example.com' to='user@example.net/home' id='i2'
     type='result'>
S:   <query xmlns='http://jabber.org/protocol/disco#info'>
S:       <identity category='collaboration' name='Live Assistant'
    type='workgroup'/>
S:       <feature var='http://jabber.org/protocol/workgroup'/>
S:       <feature var='http://jabber.org/protocol/disco#info'/>
S:   </query>
S: </iq>

U: <iq to='workgroup.example.com' from='user@example.net/home' id='id3
    ' type='get'>
U:   <query xmlns='http://jabber.org/protocol/disco#items'/>
U: </iq>
S: <iq from='workgroup.example.com' to='user@example.net/home' id='id3
    ' type='result'>
S:   <query xmlns='http://jabber.org/protocol/disco#items'>
S:       <item jid='support@workgroup.example.com' name='Example.com
    Support Live Assistant'/>
S:       <item jid='sales@workgroup.example.com' name='Example.com
    Sales Live Assistant'/>
S:   </query>
S: </iq>

U: <iq to='support@workgroup.example.com' from='user@example.net/home'
     id='id4' type="get'>
U:     <query xmlns='http://jabber.org/protocol/disco#info'/>
U: </iq>
S: <iq from='support@workgroup.example.com' to='user@example.net/home'
    id='id4' type='result'>
S:     <query xmlns='http://jabber.org/protocol/disco#info'>
S:         <identity category='collaboration' name='demo' type='
    workgroup'/>
S:         <feature var='http://jabber.org/protocol/disco#info'/>
S:         <x xmlns='jabber:x:data' type='result'>
S:             <field var='FORM_TYPE' type='hidden'>
S:                 <value>http://jabber.org/protocol/workgroup#
    workgroupinfo</value>
S:             </field>
S:             <field var='workgroup#description' label='Description'>
S:                 <value>Example.com Support Workgroup</value>
S:             </field>
S:             <field var='workgroup#online' label='Status'>
S:                 <value>ready</value>
S:             </field>
S:         </x>
S:     </query>
S: </iq>
```

6 Implementation Notes

- A workgroup is a normal XMPP messaging node and MUST maintain its own presence. It is recommended that a workgroup be able to respond to arbitrary chat messages sent to it (preferably by responding with instructions on how to join the queue). Other users may subscribe to the workgroup service's presence using standard XMPP presence-subscribe and presence-unsubscribe protocols. The workgroup service's presence can be used to determine the workgroup's status without joining the workgroup as a user or agent. For example, a website server-side component can subscribe to the workgroup presence and indicate on web pages whether a workgroup is available to offer live chat to website visitors.
- If workgroup goes offline, all queued users SHOULD be notified using the appropriate workgroup presence, status, and depart protocols.
- An implementation MAY support anonymous login by users, which makes it easier to deploy such a system on a website.
- Generally, client authors only need to implement the "user" portion of this document so that clients can contact workgroups. Implementing the "agent" portion of the document is generally left to specialized clients for agents.
- Coordination of groupchat and workgroup services is beyond the scope of this document. It is RECOMMENDED that implementations use or create standard mechanisms to allow workgroups and groupchat services to interact.

7 Security Considerations

Implementations may wish to restrict who is allowed to join workgroups as users and agents. Details concerning the implementation of this feature is outside the scope of this document.

8 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [11].

[11] 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/>.

9 XMPP Registrar Considerations

9.1 Protocol Namespaces

The XMPP Registrar [12] shall include 'http://jabber.org/protocol/workgroup' in its registry of protocol namespaces.

9.2 Service Discovery Category/Type

The XMPP Registrar shall add a Service Discovvery type of "workgroup" to the existing "collaboration" category. The registry submission is as follows:

```
<category>
  <name>collaboriation</name>
  <type>
    <name>workgroup</name>
    <desc>A workgroup component.</desc>
    <doc>XEP-0142</doc>
  </type>
</category>
```

10 XML Schema

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://jabber.org/protocol/workgroup'
    xmlns='http://jabber.org/protocol/workgroup'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The allowable root elements for the namespace defined
      herein are:
        - agent-status
        - agent-status-request
        - depart-queue
        - join-queue
        - notify-agents
        - notify-queue
```

[12] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

```
        - notify-queue-details
        - offer
        - offer-accept
        - offer-reject
        - offer-revoke
        - queue-status
    </xs:documentation>
  </xs:annotation>

  <xs:import
      namespace='jabber:x:data'
      schemaLocation='http://www.xmpp.org/schemas/x-data.xsd'/>

  <xs:element name='agent-status'>
    <xs:complexType>
      <xs:choice minOccurs='0' maxOccurs='unbounded'>
        <xs:element name='current-chats' type='xs:positiveInteger'/>
        <xs:element name='max-chats' type='xs:positiveInteger'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name='agent-status-request'>
    <xs:complexType>
      <xs:sequence minOccurs='0' maxOccurs='unbounded'>
        <xs:element ref='agent'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='depart-queue'>
    <xs:complexType>
      <xs:choice minOccurs='0' maxOccurs='unbounded'>
        <xs:element name='jid' type='xs:string'/>
        <xs:any namespace='##other' processContents='lax'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name='join-queue'>
    <xs:complexType>
      <xs:choice xmlns:xdata='jabber:x:data' minOccurs='0' maxOccurs='
          unbounded'>
        <xs:element name='queue-notifications' type='empty'/>
        <xs:element ref='xdata:x'/>
        <xs:any namespace='##other' processContents='lax'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
```

```
<xs:element name='notify-agents'>
  <xs:complexType>
    <xs:choice minOccurs='0' maxOccurs='unbounded'>
      <xs:element name='available' type='xs:positiveInteger'/>
      <xs:element name='current-chats' type='xs:positiveInteger'/>
      <xs:element name='max-chats' type='xs:positiveInteger'/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name='notify-queue'>
  <xs:complexType>
    <xs:choice minOccurs='0' maxOccurs='unbounded'>
      <xs:element name='count' type='xs:positiveInteger'/>
      <xs:element name='oldest' type='xs:dateTime'/>
      <xs:element name='time' type='xs:positiveInteger'/>
      <xs:element name='status'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='moderator'/>
            <xs:enumeration value='none'/>
            <xs:enumeration value='participant'/>
            <xs:enumeration value='visitor'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name='notify-queue-details'>
  <xs:complexType>
    <xs:sequence minOccurs='0' maxOccurs='unbounded'>
      <xs:element ref='user'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name='offer'>
  <xs:complexType>
    <xs:choice minOccurs='0' maxOccurs='unbounded'>
      <xs:element name='timeout' type='xs:positiveInteger'/>
      <xs:any namespace='##other' processContents='lax'/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name='offer-accept'>
```

```
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='jid' use='required' type='xs:string'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='offer-reject'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='jid' use='required' type='xs:string'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='offer-revoke'>
  <xs:complexType>
    <xs:sequence minOccurs='0'>
      <xs:element name='reason' type='xs:string'/>
    </xs:sequence>
    <xs:attribute name='jid' type='xs:string' use='required'/>
  </xs:complexType>
</xs:element>

<xs:element name='queue-status'>
  <xs:complexType>
    <xs:choice minOccurs='0' maxOccurs='unbounded'>
      <xs:element name='position' type='xs:positiveInteger'/>
      <xs:element name='time' type='xs:positiveInteger'/>
      <xs:any namespace='##other' processContents='lax'/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name='user'>
  <xs:complexType>
    <xs:sequence>
      <xs:element name='position' type='xs:positiveInteger'/>
      <xs:element name='time' type='xs:positiveInteger'/>
      <xs:element name='join-time' type='xs:dateTime'/>
    </xs:sequence>
    <xs:attribute name='jid' type='xs:string' use='required'/>
  </xs:complexType>
</xs:element>
```

```
  <xs:element name='agent'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='jid' use='required' type='xs:string'/>
        </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>
```

11 Acknowledgements

The author would like to thank Iain Shigeoka for his work on the first version of this document, and Derek DeMoro and Gaston Dombiak for their comments.