XEP-0016: Privacy Lists

Peter Millard

Peter Saint-Andre
mailto:xsf@stpeter.im
xmpp:peter@jabber.org
http://stpeter.im/

2017-05-20
Version 1.7

Status	Type	Short Name
Deprecated	Standards Track	privacy

This specification defines an XMPP protocol extension for enabling or disabling communication with other entities on a network. The protocol, which was first standardized in Section 10 of RFC 3921, can be used to block communication with unknown or undesirable entities. Blocking can be based on Jabber Identifier, subscription state, or roster group. The blocked stanzas can be messages, IQs, inbound or outbound presence stanzas, or all stanzas. The protocol also enables an entity to create, modify, or delete its privacy lists, apply different lists to different connected resources, define a default list, and decline the use of any privacy list during a particular communications session.

Legal

Copyright

Permissions

Warranty

Liability

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 Protocol **1**

2.1 Syntax and Semantics . . . 2
2.2 Business Rules . . . 3
2.3 Retrieving One's Privacy Lists . . . 5
2.4 Managing Active Lists . . . 7
2.5 Managing the Default List . . . 8
2.6 Editing a Privacy List . . . 10
2.7 Adding a New Privacy List . . . 11
2.8 Removing a Privacy List . . . 12
2.9 Blocking Messages . . . 12
2.10 Blocking Inbound Presence Notifications . . . 14
2.11 Blocking Outbound Presence Notifications . . . 16
2.12 Blocking IQ Stanzas . . . 17
2.13 Blocking All Communication . . . 19
2.14 Blocked Entity Attempts to Communicate with User . . . 20
2.15 Higher-Level Heuristics . . . 21

3 Discovering Support **22**

4 Implementation Notes **23**

5 Security Considerations **23**

6 IANA Considerations **23**

7 XMPP Registrar Considerations **24**

7.1 Protocol Namespaces . . . 24

8 XML Schema **24**

9 Author Note **26**

1 Introduction

Almost all types of Instant Messaging (IM) applications have found it necessary to develop some method for a user to block the receipt of messages and packets from other users (the rationale for such blockage depends on the needs of the individual user). This document defines a flexible method for communications blocking.
Note: The protocol specified herein MAY be used in conjunction with Blocking Command (XEP-0191) [1]; see XEP-0191 for details.

2 Protocol

This section has been copied without modification from Section 10 of RFC 3921 [2], with the exception of the message stanza handling rule in the Blocked Entity Attempts to Communicate with User subsection.
Most instant messaging systems have found it necessary to implement some method for users to block communications from particular other users (this is also required by sections 5.1.5, 5.1.15, 5.3.2, and 5.4.10 of RFC 2779 [3]. In XMPP this is done by managing one's privacy lists using the 'jabber:iq:privacy' namespace.
Server-side privacy lists enable successful completion of the following use cases:

- Retrieving one's privacy lists.
- Adding, removing, and editing one's privacy lists.
- Setting, changing, or declining active lists.
- Setting, changing, or declining the default list (i.e., the list that is active by default).
- Allowing or blocking messages based on JID, group, or subscription type (or globally).
- Allowing or blocking inbound presence notifications based on JID, group, or subscription type (or globally).
- Allowing or blocking outbound presence notifications based on JID, group, or subscription type (or globally).
- Allowing or blocking IQ stanzas based on JID, group, or subscription type (or globally).
- Allowing or blocking all communications based on JID, group, or subscription type (or globally).

[1] XEP-0191: Blocking Command <https://xmpp.org/extensions/xep-0191.html>.
[2] RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc3921>.
[3] RFC 2779: A Model for Presence and Instant Messaging <http://tools.ietf.org/html/rfc2779>.

Note: Presence notifications do not include presence subscriptions, only presence information that is broadcasted to entities that are subscribed to a user's presence information. Thus this includes presence stanzas with no 'type' attribute or of type='unavailable' only.

2.1 Syntax and Semantics

A user MAY define one or more privacy lists, which are stored by the user's server. Each <list/> element contains one or more rules in the form of <item/> elements, and each <item/> element uses attributes to define a privacy rule type, a specific value to which the rule applies, the relevant action, and the place of the item in the processing order.
The syntax is as follows:

```
<iq>
  <query xmlns='jabber:iq:privacy'>
    <list name='foo'>
      <item
          type='[jid|group|subscription]'
          value='bar'
          action='[allow|deny]'
          order='unsignedInt'>
        [<message/>]
        [<presence-in/>]
        [<presence-out/>]
        [<iq/>]
      </item>
    </list>
  </query>
</iq>
```

If the type is "jid", then the 'value' attribute MUST contain a valid Jabber ID. JIDs SHOULD be matched in the following order:

1. <user@domain/resource> (only that resource matches)
2. <user@domain> (any resource matches)
3. <domain/resource> (only that resource matches)
4. <domain> (the domain itself matches, as does any user@domain or domain/resource)

If the type is "group", then the 'value' attribute SHOULD contain the name of a group in the user's roster. (If a client attempts to update, create, or delete a list item with a group that is not in the user's roster, the server SHOULD return to the client an <item-not-found/> stanza error.)
If the type is "subscription", then the 'value' attribute MUST be one of "both", "to", "from",

or ”none” as defined RFC 6121, where ”none” includes entities that are totally unknown to the user and therefore not in the user’s roster at all. These values are exact matches, so that ”both” means a bidirectional subscription (not ”from” or ”to” only).
If no ’type’ attribute is included, the rule provides the ”fall-through” case.
The ’action’ attribute MUST be included and its value MUST be either ”allow” or ”deny”. [4]
The ’order’ attribute MUST be included and its value MUST be a non-negative integer that is unique among all items in the list. (If a client attempts to create or update a list with non-unique order values, the server MUST return to the client a <bad-request/> stanza error.)
The <item/> element MAY contain one or more child elements that enable an entity to specify more granular control over which kinds of stanzas are to be blocked (i.e., rather than blocking all stanzas). The allowable child elements are:

- <message/> -- blocks incoming message stanzas
- <iq/> -- blocks incoming IQ stanzas
- <presence-in/> -- blocks incoming presence notifications
- <presence-out/> -- blocks outgoing presence notifications [5]

Within the ’jabber:iq:privacy’ namespace, the <query/> child of an IQ stanza of type ”set” MUST NOT include more than one child element (i.e., the stanza MUST contain only one <active/> element, one <default/> element, or one <list/> element); if a sending entity violates this rule, the receiving entity MUST return a <bad-request/> stanza error.
When a client adds or updates a privacy list, the <list/> element SHOULD contain at least one <item/> child element; when a client removes a privacy list, the <list/> element MUST NOT contain any <item/> child elements.
When a client updates a privacy list, it must include all of the desired items (i.e., not a ”delta”).

2.2 Business Rules

1. If there is an active list set for a session, it affects only the session(s) for which it is activated, and only for the duration of the session(s); the server MUST apply the active list only and MUST NOT apply the default list (i.e., there is no ”layering” of lists).

2. The default list applies to the user as a whole, and is processed if there is no active list set for the target session/resource to which a stanza is addressed, or if there are no current sessions for the user.

[4] An implementation MUST NOT block communications from one of a user’s resources to another, even if the user happens to define a rule that would otherwise result in that behavior.
[5] When a <presence-out/> element is included, the server might want to also block presence-like notifications, such as those which use Personal Eventing Protocol (XEP-0163) [6].

3. If there is no active list set for a session (or there are no current sessions for the user), and there is no default list, then all stanzas SHOULD BE accepted or appropriately processed by the server on behalf of the user in accordance with the server rules for handling XML stanzas defined in RFC 6121.

4. Privacy lists MUST be the first delivery rule applied by a server, superseding (1) the routing and delivery rules specified in server rules for handling XML stanzas defined in RFC 6121 and (2) the handling of subscription-related presence stanzas (and corresponding generation of roster pushes) specified in RFC 6121.

5. The order in which privacy list items are processed by the server is important. List items MUST be processed in ascending order determined by the integer values of the 'order' attribute for each <item/>.

6. As soon as a stanza is matched against a privacy list rule, the server MUST appropriately handle the stanza in accordance with the rule and cease processing.

7. If no fall-through item is provided in a list, the fall-through action is assumed to be "allow".

8. If a user updates the definition for an active list, subsequent processing based on that active list MUST use the updated definition (for all resources to which that active list currently applies).

9. If a change to the subscription state or roster group of a roster item defined in an active or default list occurs during a user's session, subsequent processing based on that list MUST take into account the changed state or group (for all resources to which that list currently applies).

10. When the definition for a rule is modified, the server MUST send an IQ stanza of type "set" to all connected resources, containing a <query/> element with only one <list/> child element, where the 'name' attribute is set to the name of the modified privacy list. These "privacy list pushes" adhere to the same semantics as the "roster pushes" used in roster management, except that only the list name itself (not the full list definition or the "delta") is pushed to the connected resources. It is up to the receiving resource to determine whether to retrieve the modified list definition, although a connected resource SHOULD do so if the list currently applies to it.

11. When a resource attempts to remove a list or specify a new default list while that list applies to a connected resource other than the sending resource, the server MUST return a <conflict/> error to the sending resource and MUST NOT make the requested change.

2.3 Retrieving One's Privacy Lists

Listing 1: Client requests names of privacy lists from server

```
<iq from='romeo@example.net/orchard' type='get' id='getlist1'>
<query xmlns='jabber:iq:privacy'/>
</iq>
```

Listing 2: Server sends names of privacy lists to client, preceded by active list and default list

```
<iq type='result' id='getlist1' to='romeo@example.net/orchard'>
<query xmlns='jabber:iq:privacy'>
  <active name='private'/>
  <default name='public'/>
  <list name='public'/>
  <list name='private'/>
  <list name='special'/>
</query>
</iq>
```

Listing 3: Client requests a privacy list from server

```
<iq from='romeo@example.net/orchard' type='get' id='getlist2'>
<query xmlns='jabber:iq:privacy'>
  <list name='public'/>
</query>
</iq>
```

Listing 4: Server sends a privacy list to client

```
<iq type='result' id='getlist2' to='romeo@example.net/orchard'>
<query xmlns='jabber:iq:privacy'>
  <list name='public'>
    <item type='jid'
          value='tybalt@example.com'
          action='deny'
          order='1'/>
    <item action='allow' order='2'/>
  </list>
</query>
</iq>
```

Listing 5: Client requests another privacy list from server

```
<iq from='romeo@example.net/orchard' type='get' id='getlist3'>
<query xmlns='jabber:iq:privacy'>
  <list name='private'/>
</query>
</iq>
```

Listing 6: Server sends another privacy list to client

```
<iq type='result' id='getlist3' to='romeo@example.net/orchard'>
<query xmlns='jabber:iq:privacy'>
  <list name='private'>
    <item type='subscription'
          value='both'
          action='allow'
          order='10'/>
    <item action='deny' order='15'/>
  </list>
</query>
</iq>
```

Listing 7: Client requests yet another privacy list from server

```
<iq from='romeo@example.net/orchard' type='get' id='getlist4'>
<query xmlns='jabber:iq:privacy'>
  <list name='special'/>
</query>
</iq>
```

Listing 8: Server sends yet another privacy list to client

```
<iq type='result' id='getlist4' to='romeo@example.net/orchard'>
<query xmlns='jabber:iq:privacy'>
  <list name='special'>
    <item type='jid'
          value='juliet@example.com'
          action='allow'
          order='6'/>
    <item type='jid'
          value='benvolio@example.org'
          action='allow'
          order='7'/>
    <item type='jid'
          value='mercutio@example.org'
          action='allow'
          order='42'/>
    <item action='deny' order='666'/>
  </list>
</query>
</iq>
```

In this example, the user has three lists: (1) 'public', which allows communications from everyone except one specific entity (this is the default list); (2) 'private', which allows communications only with contacts who have a bidirectional subscription with the user (this is the active list); and (3) 'special', which allows communications only with three specific entities. If the user attempts to retrieve a list but a list by that name does not exist, the server MUST return an <item-not-found/> stanza error to the user:

Listing 9: Client attempts to retrieve non-existent list

```
<iq to='romeo@example.net/orchard' type='error' id='getlist5'>
<query xmlns='jabber:iq:privacy'>
  <list name='The Empty Set'/>
</query>
<error type='cancel'>
  <item-not-found
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
</error>
</iq>
```

The user is allowed to retrieve only one list at a time. If the user attempts to retrieve more than one list in the same request, the server MUST return a <bad request/> stanza error to the user:

Listing 10: Client attempts to retrieve more than one list

```
<iq to='romeo@example.net/orchard' type='error' id='getlist6'>
<query xmlns='jabber:iq:privacy'>
  <list name='public'/>
  <list name='private'/>
  <list name='special'/>
</query>
<error type='modify'>
  <bad-request
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
</error>
</iq>
```

2.4 Managing Active Lists

In order to set or change the active list currently being applied by the server, the user MUST send an IQ stanza of type "set" with a <query/> element qualified by the 'jabber:iq:privacy' namespace that contains an empty <active/> child element possessing a 'name' attribute whose value is set to the desired list name.

Listing 11: Client requests change of active list

```
<iq from='romeo@example.net/orchard' type='set' id='active1'>
<query xmlns='jabber:iq:privacy'>
  <active name='special'/>
</query>
</iq>
```

The server MUST activate and apply the requested list before sending the result back to the client.

Listing 12: Server acknowledges success of active list change

```
<iq type='result' id='active1' to='romeo@example.net/orchard'/>
```

If the user attempts to set an active list but a list by that name does not exist, the server MUST return an <item-not-found/> stanza error to the user:

Listing 13: Client attempts to set a non-existent list as active

```
<iq to='romeo@example.net/orchard' type='error' id='active2'>
<query xmlns='jabber:iq:privacy'>
  <active name='The␣Empty␣Set'/>
</query>
<error type='cancel'>
  <item-not-found
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
</error>
</iq>
```

In order to decline the use of any active list, the connected resource MUST send an empty <active/> element with no 'name' attribute.

Listing 14: Client declines the use of active lists

```
<iq from='romeo@example.net/orchard' type='set' id='active3'>
<query xmlns='jabber:iq:privacy'>
  <active/>
</query>
</iq>
```

Listing 15: Server acknowledges success of declining any active list

```
<iq type='result' id='active3' to='romeo@example.net/orchard'/>
```

2.5 Managing the Default List

In order to change its default list (which applies to the user as a whole, not only the sending resource), the user MUST send an IQ stanza of type "set" with a <query/> element qualified by

the 'jabber:iq:privacy' namespace that contains an empty <default/> child element possessing a 'name' attribute whose value is set to the desired list name.

Listing 16: User requests change of default list

```
<iq from='romeo@example.net/orchard' type='set' id='default1'>
<query xmlns='jabber:iq:privacy'>
  <default name='special'/>
</query>
</iq>
```

Listing 17: Server acknowledges success of default list change

```
<iq type='result' id='default1' to='romeo@example.net/orchard'/>
```

If the user attempts to change which list is the default list but the default list is in use by at least one connected resource other than the sending resource, the server MUST return a <conflict/> stanza error to the sending resource:

Listing 18: Client attempts to change the default list but that list is in use by another resource

```
<iq to='romeo@example.net/orchard' type='error' id='default1'>
<query xmlns='jabber:iq:privacy'>
  <default name='special'/>
</query>
<error type='cancel'>
  <conflict
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
</error>
</iq>
```

If the user attempts to set a default list but a list by that name does not exist, the server MUST return an <item-not-found/> stanza error to the user:

Listing 19: Client attempts to set a non-existent list as default

```
<iq to='romeo@example.net/orchard' type='error' id='default1'>
<query xmlns='jabber:iq:privacy'>
  <default name='The␣Empty␣Set'/>
</query>
<error type='cancel'>
  <item-not-found
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
</error>
</iq>
```

In order to decline the use of a default list (i.e., to use the domain's stanza routing rules at all times), the user MUST send an empty <default/> element with no 'name' attribute.

Listing 20: Client declines the use of the default list

```
<iq from='romeo@example.net/orchard' type='set' id='default2'>
<query xmlns='jabber:iq:privacy'>
  <default/>
</query>
</iq>
```

Listing 21: Server acknowledges success of declining any default list

```
<iq type='result' id='default2' to='romeo@example.net/orchard'/>
```

If one connected resource attempts to decline the use of a default list for the user as a whole but the default list currently applies to at least one other connected resource, the server MUST return a <conflict/> error to the sending resource:

Listing 22: Client attempts to decline a default list but that list is in use by another resource

```
<iq to='romeo@example.net/orchard' type='error' id='default3'>
<query xmlns='jabber:iq:privacy'>
  <default/>
</query>
<error type='cancel'>
  <conflict
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
</error>
</iq>
```

2.6 Editing a Privacy List

In order to edit a privacy list, the user MUST send an IQ stanza of type "set" with a <query/> element qualified by the 'jabber:iq:privacy' namespace that contains one <list/> child element possessing a 'name' attribute whose value is set to the list name the user would like to edit. The <list/> element MUST contain one or more <item/> elements, which specify the user's desired changes to the list by including all elements in the list (not the "delta").

Listing 23: Client edits a privacy list

```
<iq from='romeo@example.net/orchard' type='set' id='edit1'>
<query xmlns='jabber:iq:privacy'>
  <list name='public'>
    <item type='jid'
          value='tybalt@example.com'
          action='deny'
          order='3'/>
    <item type='jid'
          value='paris@example.org'
          action='deny'
```

```
          order='5'/>
    <item action='allow' order='68'/>
  </list>
</query>
</iq>
```

Listing 24: Server acknowledges success of list edit

```
<iq type='result' id='edit1' to='romeo@example.net/orchard'/>
```

Note: The value of the 'order' attribute for any given item is not fixed. Thus in the foregoing example if the user would like to add 4 items between the "tybalt@example.com" item and the "paris@example.org" item, the user's client MUST renumber the relevant items before submitting the list to the server.
The server MUST now send a "privacy list push" to all connected resources:

Listing 25: Privacy list push on list edit

```
<iq to='romeo@example.net/orchard' type='set' id='push1'>
<query xmlns='jabber:iq:privacy'>
  <list name='public'/>
</query>
</iq>

<iq to='romeo@example.net/home' type='set' id='push2'>
<query xmlns='jabber:iq:privacy'>
  <list name='public'/>
</query>
</iq>
```

In accordance with the semantics of IQ stanzas defined in XMPP Core [7], each connected resource MUST return an IQ result to the server as well:

Listing 26: Acknowledging receipt of privacy list pushes

```
<iq from='romeo@example.net/orchard'
  type='result'
  id='push1'/>

<iq from='romeo@example.net/home'
  type='result'
  id='push2'/>
```

2.7 Adding a New Privacy List

The same protocol used to edit an existing list is used to create a new list. If the list name matches that of an existing list, the request to add a new list will overwrite the old one. As

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

with list edits, the server MUST also send a ”privacy list push” to all connected resources.

2.8 Removing a Privacy List

In order to remove a privacy list, the user MUST send an IQ stanza of type ”set” with a <query/> element qualified by the ’jabber:iq:privacy’ namespace that contains one empty <list/> child element possessing a ’name’ attribute whose value is set to the list name the user would like to remove.

Listing 27: Client removes a privacy list

```
<iq from='romeo@example.net/orchard' type='set' id='remove1'>
<query xmlns='jabber:iq:privacy'>
  <list name='private'/>
</query>
</iq>
```

Listing 28: Server acknowledges success of list removal

```
<iq type='result' id='remove1' to='romeo@example.net/orchard'/>
```

If a user attempts to remove a list that is currently being applied to at least one resource other than the sending resource, the server MUST return a <conflict/> stanza error to the user; i.e., the user MUST first set another list to active or default before attempting to remove it. If the user attempts to remove a list but a list by that name does not exist, the server MUST return an <item-not-found/> stanza error to the user. If the user attempts to remove more than one list in the same request, the server MUST return a <bad request/> stanza error to the user.

2.9 Blocking Messages

Server-side privacy lists enable a user to block incoming messages from other entities based on the entity’s JID, roster group, or subscription status (or globally). The following examples illustrate the protocol. (Note: For the sake of brevity, IQ stanzas of type ”result” are not shown in the following examples, nor are ”privacy list pushes”.)

Listing 29: User blocks based on JID

```
<iq from='romeo@example.net/orchard' type='set' id='msg1'>
<query xmlns='jabber:iq:privacy'>
  <list name='message-jid-example'>
    <item type='jid'
          value='tybalt@example.com'
          action='deny'
          order='3'>
      <message/>
```

```
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive messages from the entity with the specified JID.

Listing 30: User blocks based on roster group

```
<iq from='romeo@example.net/orchard' type='set' id='msg2'>
<query xmlns='jabber:iq:privacy'>
  <list name='message-group-example'>
    <item type='group'
          value='Enemies'
          action='deny'
          order='4'>
      <message/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive messages from any entities in the specified roster group.

Listing 31: User blocks based on subscription type

```
<iq from='romeo@example.net/orchard' type='set' id='msg3'>
<query xmlns='jabber:iq:privacy'>
  <list name='message-sub-example'>
    <item type='subscription'
          value='none'
          action='deny'
          order='5'>
      <message/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive messages from any entities with the specified subscription type.

Listing 32: User blocks globally

```
<iq from='romeo@example.net/orchard' type='set' id='msg4'>
<query xmlns='jabber:iq:privacy'>
  <list name='message-global-example'>
```

```
    <item action='deny' order='6'>
      <message/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive messages from any other users.

2.10 Blocking Inbound Presence Notifications

Server-side privacy lists enable a user to block incoming presence notifications from other entities based on the entity's JID, roster group, or subscription status (or globally). The following examples illustrate the protocol.
Note: Presence notifications do not include presence subscriptions, only presence information that is broadcasted to the user because the user is currently subscribed to a contact's presence information. Thus this includes presence stanzas with no 'type' attribute or of type='unavailable' only.

Listing 33: User blocks based on JID

```
<iq from='romeo@example.net/orchard' type='set' id='presin1'>
<query xmlns='jabber:iq:privacy'>
  <list name='presin-jid-example'>
    <item type='jid'
          value='tybalt@example.com'
          action='deny'
          order='7'>
      <presence-in/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive presence notifications from the entity with the specified JID.

Listing 34: User blocks based on roster group

```
<iq from='romeo@example.net/orchard' type='set' id='presin2'>
<query xmlns='jabber:iq:privacy'>
  <list name='presin-group-example'>
    <item type='group'
          value='Enemies'
          action='deny'
          order='8'>
```

```
      <presence-in/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive presence notifications from any entities in the specified roster group.

Listing 35: User blocks based on subscription type

```
<iq from='romeo@example.net/orchard' type='set' id='presin3'>
<query xmlns='jabber:iq:privacy'>
  <list name='presin-sub-example'>
    <item type='subscription'
          value='to'
          action='deny'
          order='9'>
      <presence-in/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive presence notifications from any entities with the specified subscription type.

Listing 36: User blocks globally

```
<iq from='romeo@example.net/orchard' type='set' id='presin4'>
<query xmlns='jabber:iq:privacy'>
  <list name='presin-global-example'>
    <item action='deny' order='11'>
      <presence-in/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive presence notifications from any other users.
Note: If a client blocks incoming presence notifications from an entity that has been available before, the user's server SHOULD send unavailable presence to the user on behalf of that contact.

2.11 Blocking Outbound Presence Notifications

Server-side privacy lists enable a user to block outgoing presence notifications to other entities based on the entity's JID, roster group, or subscription status (or globally). The following examples illustrate the protocol.
Note: Presence notifications do not include presence subscriptions, only presence information that is broadcasted to contacts because those contacts are currently subscribed to the user's presence information. Thus this includes presence stanzas with no 'type' attribute or of type='unavailable' only.

Listing 37: User blocks based on JID

```
<iq from='romeo@example.net/orchard' type='set' id='presout1'>
<query xmlns='jabber:iq:privacy'>
  <list name='presout-jid-example'>
    <item type='jid'
          value='tybalt@example.com'
          action='deny'
          order='13'>
      <presence-out/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not send presence notifications to the entity with the specified JID.

Listing 38: User blocks based on roster group

```
<iq from='romeo@example.net/orchard' type='set' id='presout2'>
<query xmlns='jabber:iq:privacy'>
  <list name='presout-group-example'>
    <item type='group'
          value='Enemies'
          action='deny'
          order='15'>
      <presence-out/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not send presence notifications to any entities in the specified roster group.

Listing 39: User blocks based on subscription type

```
<iq from='romeo@example.net/orchard' type='set' id='presout3'>
<query xmlns='jabber:iq:privacy'>
  <list name='presout-sub-example'>
    <item type='subscription'
          value='from'
          action='deny'
          order='17'>
      <presence-out/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not send presence notifications to any entities with the specified subscription type.

Listing 40: User blocks globally

```
<iq from='romeo@example.net/orchard' type='set' id='presout4'>
<query xmlns='jabber:iq:privacy'>
  <list name='presout-global-example'>
    <item action='deny' order='23'>
      <presence-out/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not send presence notifications to any other users.
Note: When a user blocks outbound presence to a contact, the user's server MUST send unavailable presence information to the contact (but only if the contact is allowed to receive presence notifications from the user in accordance with the rules defined in RFC 6121).

2.12 Blocking IQ Stanzas

Server-side privacy lists enable a user to block incoming IQ stanzas from other entities based on the entity's JID, roster group, or subscription status (or globally). The following examples illustrate the protocol.

Listing 41: User blocks based on JID

```
<iq from='romeo@example.net/orchard' type='set' id='iq1'>
<query xmlns='jabber:iq:privacy'>
  <list name='iq-jid-example'>
    <item type='jid'
```

```
          value='tybalt@example.com'
          action='deny'
          order='29'>
      <iq/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive IQ stanzas from the entity with the specified JID.

Listing 42: User blocks based on roster group

```
<iq from='romeo@example.net/orchard' type='set' id='iq2'>
<query xmlns='jabber:iq:privacy'>
  <list name='iq-group-example'>
    <item type='group'
          value='Enemies'
          action='deny'
          order='31'>
      <iq/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive IQ stanzas from any entities in the specified roster group.

Listing 43: User blocks based on subscription type

```
<iq from='romeo@example.net/orchard' type='set' id='iq3'>
<query xmlns='jabber:iq:privacy'>
  <list name='iq-sub-example'>
    <item type='subscription'
          value='none'
          action='deny'
          order='17'>
      <iq/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive IQ stanzas from any entities with the specified subscription type.

Listing 44: User blocks globally

```
<iq from='romeo@example.net/orchard' type='set' id='iq4'>
<query xmlns='jabber:iq:privacy'>
  <list name='iq-global-example'>
    <item action='deny' order='1'>
      <iq/>
    </item>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive IQ stanzas from any other users.

2.13 Blocking All Communication

Server-side privacy lists enable a user to block all stanzas from and to other entities based on the entity's JID, roster group, or subscription status (or globally). Note that this includes subscription-related presence stanzas, which are excluded by Blocking Inbound Presence Notifications. The following examples illustrate the protocol.

Listing 45: User blocks based on JID

```
<iq from='romeo@example.net/orchard' type='set' id='all1'>
<query xmlns='jabber:iq:privacy'>
  <list name='all-jid-example'>
    <item type='jid'
          value='tybalt@example.com'
          action='deny'
          order='23'/>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive any communications from, nor send any stanzas to, the entity with the specified JID.

Listing 46: User blocks based on roster group

```
<iq from='romeo@example.net/orchard' type='set' id='all2'>
<query xmlns='jabber:iq:privacy'>
  <list name='all-group-example'>
    <item type='group'
          value='Enemies'
          action='deny'
          order='13'/>
  </list>
```

```
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive any communications from, nor send any stanzas to, any entities in the specified roster group.

Listing 47: User blocks based on subscription type

```
<iq from='romeo@example.net/orchard' type='set' id='all3'>
<query xmlns='jabber:iq:privacy'>
  <list name='all-sub-example'>
    <item type='subscription'
          value='none'
          action='deny'
          order='11'/>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive any communications from, nor send any stanzas to, any entities with the specified subscription type.

Listing 48: User blocks globally

```
<iq from='romeo@example.net/orchard' type='set' id='all4'>
<query xmlns='jabber:iq:privacy'>
  <list name='all-global-example'>
    <item action='deny' order='7'/>
  </list>
</query>
</iq>
```

As a result of creating and applying the foregoing list, the user will not receive any communications from, nor send any stanzas to, any other users.

2.14 Blocked Entity Attempts to Communicate with User

If a blocked entity attempts to send a stanza to the user (i.e., an inbound stanza from the user's perspective), the user's server shall handle the stanza according to the following rules:

- For presence stanzas (including notifications, subscriptions, and probes), the server MUST NOT respond and MUST NOT return an error.
- For message stanzas, the server SHOULD return an error, which SHOULD be <service-unavailable/>. [8]

[8] Until version 1.5 of this document (therefore also in RFC 6121), it was recommended to silently ignore message stanzas, which unfortunately resulted in a delivery "black hole" regarding message stanzas.

- For IQ stanzas of type ”get” or ”set”, the server MUST return an error, which SHOULD be <service-unavailable/>. IQ stanzas of other types MUST be silently dropped by the server.

If the foregoing suggestions are followed, the user will appear offline to the contact.

Listing 49: Blocked entity attempts to send IQ get

```
<iq type='get'
  to='romeo@example.net'
  from='tybalt@example.com/pda'
  id='probing1'>
<query xmlns='jabber:iq:version'/>
</iq>
```

Listing 50: Server returns error to blocked entity

```
<iq type='error'
  from='romeo@example.net'
  to='tybalt@example.com/pda'
  id='probing1'>
<query xmlns='jabber:iq:version'/>
<error type='cancel'>
  <service-unavailable
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
</error>
</iq>
```

If the user attempts to send an outbound stanza to a contact and that stanza type is blocked, the user's server MUST NOT route the stanza to the contact but instead MUST return a <not-acceptable/> error:

Listing 51: Error: contact is blocked

```
<message type='error' from='romeo@montague.net' to='juliet@capulet.com
    '>
  <body>Can you hear me now?</body>
  <error type='cancel'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</message>
```

2.15 Higher-Level Heuristics

When building a representation of a higher-level privacy heuristic, a client SHOULD use the simplest possible representation.
For example, the heuristic ”block all communications with any user not in my roster” could

be constructed in any of the following ways:

- allow communications from all JIDs in my roster (i.e., listing each JID as a separate list item), but block communications with everyone else
- allow communications from any user who is in one of the groups that make up my roster (i.e., listing each group as a separate list item), but block communications from everyone else
- allow communications from any user with whom I have a subscription of 'both' or 'to' or 'from' (i.e., listing each subscription value separately), but block communications from everyone else
- block communications from anyone whose subscription state is 'none'

The final representation is the simplest and SHOULD be used; here is the XML that would be sent in this case:

```
<iq type='set' id='heuristic1'>
<query xmlns='jabber:iq:privacy'>
  <list name='heuristic-example'>
    <item type='subscription'
          value='none'
          action='deny'
          order='437'/>
  </list>
</query>
</iq>
```

3 Discovering Support

If an entity supports the privacy lists protocol, it MUST report that fact by including a service discovery feature of "jabber:iq:privacy" (see Protocol Namespaces regarding issuance of one or more permanent namespaces) in response to a Service Discovery (XEP-0030) [9] information request:

Listing 52: Service Discovery information request

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

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

Listing 53: Service Discovery information response

```
<iq from='example.com'
    id='disco1'
    to='juliet@example.com/chamber'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='jabber:iq:privacy'/>
    ...
  </query>
</iq>
```

4 Implementation Notes

When a server receives a block command from a user, it MAY cancel any existing presence subscriptions between the user and the blocked user and MAY send a message to the blocked user; however, it is RECOMMENDED to deploy so-called "polite blocking" instead (i.e., to not cancel the presence subscriptions or send a notification). Which approach to follow is a matter of local service policy.
A service MAY also filter blocking users out of searches performed on user directories (see, for example, Jabber Search (XEP-0055) [10]); however, that functionality is out of scope for this specification.

5 Security Considerations

If properly implemented, this protocol extension does not introduce any new security concerns above and beyond those defined in RFC 6120 and RFC 6121.

6 IANA Considerations

No interaction with the Internet Assigned Numbers Authority (IANA) [11] is required as a result of this specification.

[10] XEP-0055: Jabber Search <https://xmpp.org/extensions/xep-0055.html>.
[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/>.

7 XMPP Registrar Considerations

7.1 Protocol Namespaces

The XMPP Registrar [12] already includes 'jabber:iq:privacy' in its registry of protocol namespaces (see <https://xmpp.org/registrar/namespaces.html>).

8 XML Schema

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

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

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0016: http://www.xmpp.org/extensions/xep-0016.html
    </xs:documentation>
  </xs:annotation>

<xs:element name='query'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='active'
                  minOccurs='0'/>
      <xs:element ref='default'
                  minOccurs='0'/>
      <xs:element ref='list'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name='active'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:NMTOKEN'>
        <xs:attribute name='name'
```

[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/>.

```
                    type='xs:string'
                    use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='default'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:NMTOKEN'>
        <xs:attribute name='name'
                    type='xs:string'
                    use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='list'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='item'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='name'
                  type='xs:string'
                  use='required'/>
  </xs:complexType>
</xs:element>

<xs:element name='item'>
  <xs:complexType>
    <xs:sequence>
      <xs:element name='iq'
                  minOccurs='0'
                  type='empty'/>
      <xs:element name='message'
                  minOccurs='0'
                  type='empty'/>
      <xs:element name='presence-in'
                  minOccurs='0'
                  type='empty'/>
      <xs:element name='presence-out'
                  minOccurs='0'
                  type='empty'/>
    </xs:sequence>
    <xs:attribute name='action' use='required'>
```

```
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='allow'/>
          <xs:enumeration value='deny'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name='order'
                  type='xs:unsignedInt'
                  use='required'/>
    <xs:attribute name='type' use='optional'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='group'/>
          <xs:enumeration value='jid'/>
          <xs:enumeration value='subscription'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name='value'
                  type='xs:string'
                  use='optional'/>
  </xs:complexType>
</xs:element>

<xs:simpleType name='empty'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value=''/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>
```

9 Author Note

Peter Millard, the author of this specification from version 0.1 through version 0.3, died on April 26, 2006.