XEP-0355: Namespace Delegation

Jérôme Poisson
mailto:goffi@goffi.org
xmpp:goffi@jabber.fr

2021-10-15
Version 0.5

Status	Type	Short Name
Experimental	Standards Track	NOT_YET_ASSIGNED

This specification provides a way for XMPP server to delegate treatments for a namespace to an other entity

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Some XMPP features must be offered by the server itself, or can't be available, that's the case of Personal Eventing Protocol (XEP-0163) [1] which is used in several places (e.g. bookmarks storage). But it can be desirable to use an external entity to manage some of these features, because it implements things that the server don't, or because it uses a special implementation useful in a particular case. Some people may also want to decentralize a feature on an entity under their control. This XEP try to solve these cases.Additionaly, a method to do generic treatments (independent of server) on stanza is also provided.
This XEP is complementary to Privileged Entity (XEP-0356) [2] (and works in a similar way), although they can be used together or separately.
Here are some use cases of namespace delegation:

- use an external component for a PEP service because the server doesn't implement it or lacks some features
- decentralize a server feature to an entity under client control
- make a component which react on new user registration, independent of server implementation
- server agnostic roster filtering

2 Requirements

Namespace delegation can be used in two modes:

- **admin** mode, where delegation is specified by the server administrator.
- **client** mode, where it can be requested by any user.

In *admin* mode, the managing entity manages stanza of the delegated namespace for all users registered on the server. The namespace delegation MUST be totally transparent for the managed entities.
In *client* mode, a managing entity MUST have an explicit authorization for any namespace he wants to use. Client SHOULD be able to check and revoke granted permissions, and if it's not possible, permissions MUST be revoked after a disconnection.

[1] XEP-0163: Personal Eventing Protocol <https://xmpp.org/extensions/xep-0163.html>.
[2] XEP-0356: Privileged Entity <https://xmpp.org/extensions/xep-0356.html>.

3 Glossary

- **Delegated namespace** — the namespace being managed by an external entity.
- **Filtering attribute** — an attribute which must be present in first <iq/> child element to delegate the namespace.
- **Managing entity** — the entity which actualy manages the delegated namespace.
- **Delegating server** — the server which delegate the stanzas to the managing entity.
- **Managed entity** — an entity which wants to have a namespace of its server delegated to a managing entity.

4 Admin Mode Use Cases

4.1 Server Allows Namespaces Delegations

Namespaces delegations are granted in the server configuration. Only <iq/> stanza namespaces can be delegated.
A feature is delegated using:

1. its namespace: e.g. *'urn:xmpp:mam:2'*
2. zero or more filtering attribute (attributes which must be present in the initial <iq/> child element): e.g. *'node'*
3. the jid of the managing entity: e.g. *'managing.capulet.lit'*

4.2 Delegation Request Use Case

Once the managing entity is authenticated and stream is started, the server send it a <message/> stanza with a <delegation/> elements which MUST have the **'urn:xmpp:delegation:2'** namespace. This element contains <delegated/> elements which MUST contain a 'namespace' attribute indicating the delegated namespace. If there is additional attribute filtering, the <delegated/> can have children <attribute/> elements which MUST contain a 'name' attribute with the name of the filtering attribute.

Listing 1: server advertise delegated namespaces

```
<message from='capulet.lit' to='pubub.capulet.lit' id='12345'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <delegated namespace='urn:xmpp:mam:2'>
            <attribute name='node' />
        </delegated>
        <delegated namespace='http://jabber.org/protocol/pubsub'/>
```

```
    </delegation>
</message>
```

Here *pubsub.capulet.lit* will received all stanzas of pubsub namespace sent to *capulet.lit*. It will also receive MAM stanzas, but only if the 'node' attribute is present in <query/>.

4.3 Server Forwards Delegated <iq/> Stanza

When a server receives a stanza for a delegated namespace which is either directed to him (no 'to' attribute, or 'to' attribute with its own JID), or directed to any bare jid (and only bare jid) that it manages (i.e. the *domainpart* is a domain handled by the server where delegation is activated), it MUST forwards it to the managing entity:

Listing 2: Juliet sends her mood to her server via PEP

```
<iq from='juliet@capulet.lit/balcony'
    id='pep1'
    type='set'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
        <publish node='http://jabber.org/protocol/mood'>
        <item>
            <mood xmlns='http://jabber.org/protocol/mood'>
                <annoyed/>
                <text>curse my nurse!</text>
            </mood>
        </item>
        </publish>
    </pubsub>
</iq>
```

The server gets this stanza, sees that this namespace is delegated to *pubsub.capulet.lit*, so it forwards it.
To forward, an <iq/> stanza of type "set" is used which contain a <delegation/> element (with namespace *'urn:xmpp:delegation:2'*) which in turn contain a <forwarded/> element encapsulating the initial stanza, according to Stanza Forwarding (XEP-0297) [3]:

Listing 3: Server Delegates The Stanza To pubsub.capulet.lit

```
<iq from='capulet.lit'
    to='pubsub.capulet.lit'
    id='delegate1'
    type='set'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq from='juliet@capulet.lit/balcony'
```

[3] XEP-0297: Stanza Forwarding <https://xmpp.org/extensions/xep-0297.html>.

```
                id='pep1'
                type='set'
                xmlns='jabber:client'>
                <pubsub xmlns='http://jabber.org/protocol/pubsub'>
                    <publish node='http://jabber.org/protocol/mood'>
                    <item>
                        <mood xmlns='http://jabber.org/protocol/mood'>
                            <annoyed/>
                            <text>curse my nurse!</text>
                        </mood>
                    </item>
                    </publish>
                </pubsub>
            </iq>
        </forwarded>
    </delegation>
</iq>
```

The managing entity replies to the stanza by encapsulating its <iq/> result in the same way:

Listing 4: pubsub.capulet.lit Replies To Juliet

```
<iq from='pubsub.capulet.lit'
    to='capulet.lit'
    id='delegate1'
    type='result'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq to='juliet@capulet.lit/balcony'
                id='pep1'
                type='result'
                xmlns='jabber:client'>
                <pubsub xmlns='http://jabber.org/protocol/pubsub' />
            </iq>
        </forwarded>
    </delegation>
</iq>
```

Then the server MUST decapsulate the <iq/> result, MUST insure that the 'to' and the 'from' attribute corresponds to respectively the 'from' and the 'to' attributes of the initial stanza, MUST insure that the 'id' attribute of the decapsulated stanza is the same as the initial 'id' attribute and that 'type' is "result". If everything is alright, it can send the decapsulated stanza to Juliet.
If the forwarded result from managing entity is bad (i.e. wrong 'to', 'from', 'id' or 'result' attributes), the server MUST send an <iq/> error with condition <service-unavailable/> to managed entity, and MAY close the connexion with managing entity.

Listing 5: capulet.lit Replies To Juliet

```
<iq to='juliet@capulet.lit/balcony'
    id='pep1'
    type='result'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub' />
</iq>
```

The workflow is fully transparent for Juliet.
N.B.: If the server encounter a delegated namespace and the managing entity is not available, it MUST return an <iq/> stanza of type "error" with an error condition of <service-unavailable/>
N.B.: Similarly, if the managing entity return an <iq/> stanza of type "error", the server must return itself an <iq/> stanza of type "error" with an error condition of <service-unavailable/>
N.B.: If the server encounter a delegated namespace but the filtering attribute does not match, it MUST follow its normal behaviour, i.e. it must follow the same behaviour it would have had if the namespace was not delegated at all

4.3.1 Stanzas from managing entity

If a stanza is sent by the managing entity on a managed namespace, the server MUST NOT forward it. This way, the managing entity can use privileged entity to do specific treatments, a kind of universal plugin (i.e. working with all servers implementing Namespace Delegation (XEP-0355) [4] and Privileged Entity (XEP-0356) [5]).
In the following examples, *juliet@capulet.lit* has its "*jabber:iq:roster*" namespace delegated to *filter.capulet.lit*. *filter.capulet.lit* is a server agnostic component which filters allowed entities (which can be added to a roster), and sort them in enforced groups.

Listing 6: Juliet adds Romeo to her roster

```
<iq from='juliet@capulet.lit/balcony'
          id='roster1'
          type='set'>
    <query xmlns='jabber:iq:roster'>
        <item jid='romeo@montaigu.lit'
              name='My Romeo'>
        </item>
    </query>
</iq>
```

Listing 7: server forwards stanza to managing entity

```
<iq from='capulet.lit'
    to='pubsub.capulet.lit'
```

[4] XEP-0355: Namespace Delegation <https://xmpp.org/extensions/xep-0355.html>.
[5] XEP-0356: Privileged Entity <https://xmpp.org/extensions/xep-0356.html>.

```
    id='delegate1'
    type='set'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq from='juliet@capulet.lit/balcony'
                id='roster1'
                type='set'
                xmlns='jabber:client'>
                <query xmlns='jabber:iq:roster'>
                    <item jid='romeo@montaigu.lit'
                          name='My Romeo'>
                    </item>
                </query>
            </iq>
        </forwarded>
    </delegation>
</iq>
```

filter.capulet.lit accepts to add Romeo, but all JIDs with a *montaigu.lit* must be in a "Rivals" group, so it first returns a success result (Romeo is accepted).

Listing 8: filtering component accept Romeo

```
<iq from='pubsub.capulet.lit'
    to='capulet.lit'
    id='delegate1'
    type='result'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq to='juliet@capulet.lit/balcony'
                id='roster1'
                type='result'
                xmlns='jabber:client' />
        </forwarded>
    </delegation>
</iq>
```

Listing 9: server decapsulate and send the result with the initial id

```
<iq to='juliet@capulet.lit/balcony'
    id='roster1'
    type='result' />
```

At this stade, the entity is accepted, but not added to the roster. *filter.capulet.lit* is also a privileged entity which can manage "*jabber:iq:roster*", so it uses this ability to add Romeo in the enforced group:

Listing 10: filter.capulet.lit uses privileged entity to add Romeo

```
<iq to='juliet@capulet.lit'
    from='filter.capulet.lit'
    id='roster2'
    type='set'>
    <query xmlns='jabber:iq:roster'>
        <item jid='romeo@montaigu.lit'
              name='My Romeo'>
            <group>Rivals</group>
        </item>
    </query>
</iq>
```

The namespace is delegated, but as the stanza is from the managing entity, the server manages it normally. The entity is also privileged, so it can change the stanza of Juliet, the server accepts:

Listing 11: server accept new entity in roster

```
<iq to='filter.capulet.lit'
    from='juliet@capulet.lit'
    id='roster2'
    type='result'/>
```

The server will then send the roster pushes (with the enforced group) normally.

5 Client Mode Use Cases

5.1 Client Delegation Request Use Case

In *client* mode, the managing entity is not certified by the server administrator, so the delegation MUST be **explicitly** allowed by the managed entity. This is initiated by the managing entity (it can be after an interaction with a managed entity, like a subscription).
To request delegation for a particular entity, the managing entity MUST have an <iq/> stanza with *'urn:xmpp:delegation:2'* namespace. The <query/> element MUST have a 'to' attribute which specify the entity it wants to manage.
Namespace delegations are asked with a <delegate/> element, which MUST contain a 'namespace' attribute set to the requested namespace.
If an entity want to manage PEP service for Juliet, it can ask the delegation like this:

Listing 12: managing entity asks for namespace delegation for one particular entity

```
<iq from='pubsub.montaigu.lit' to='capulet.lit' type='get' id='
   delegation1'>
    <query xmlns='urn:xmpp:delegation:2'
           to='juliet@capulet.lit'>
```

```
        <delegate namespace='http://jabber.org/protocol/pubsub'/>
    </query>
</iq>
```

Once received the delegation request, the server ask to the client if it grants access to the requested namespace using Data Forms (XEP-0004) [6]. The server use a challenge which it MUST have generated itself.

Listing 13: server asks user for the namespace delegation

```
<message from='capulet.lit' to='juliet@capulet.lit/balcony'>
    <body>
        pubsub.montaigu.lit wants to manage a feature normally managed
            by the server.
        Do you allow it to manage the following features?

        Be careful! According management to an entity is a serious
            thing,
        think twice that you can trust the entity before doing this.
    </body>
    <x xmlns='jabber:x:data' type='form'>
        <title>Delegation request</title>
        <instructions>pubsub.montaigu.lit wants to manage the
            following features:
            Do you allow it?</instructions>
        <field type='hidden' var='challenge'><value>5439123</value></
            field>
        <field type='hidden' var='FORM_TYPE'>
            <value>urn:xmpp:delegation:2</value>
        </field>
        <field type='list-single'
            label='Manage␣PubSub␣(http://jabber.org/protocol/pubsub)'
            var='http://jabber.org/protocol/pubsub'>
            <value>0</value>
            <option label='No'><value>0</value></option>
            <option label='Yes'><value>1</value></option>
        </field>
    </x>
</message>
```

The server SHOULD include a warning message, SHOULD translate the namespace to human friendly names (and MAY keep the original namespace in addition) and MUST set the default value to '**0**' (permission refused). The server MUST use namespaces as field var, so a client can use them to have a customized display.
The client can then answer to the form:

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

Listing 14: client answer to the form

```
<message from='juliet@capulet.lit/balcony' to='capulet.lit'>
  <x xmlns='jabber:x:data' type='submit'>
    <field var='FORM_TYPE'>
      <value></value>
    </field>
    <field var='challenge'><value>5439123</value></field>
    <field var='http://jabber.org/protocol/pubsub'><value>1</value></
        field>
  </x>
</message>
```

Here Juliet allows *pubsub.montaigu.lit* to manage the PubSub (and then PEP) service.
Finaly, the server notifies the entity of the granted delegation. It do this in the same way as for admin mode, except that the <delegation/> element has an attitional 'to' attribute set to the managed entity bare jid:

Listing 15: server advertise delegated namespaces for juliet

```
<message from='capulet.net' to='pubub.capulet.lit' id='6789'>
    <delegation xmlns='urn:xmpp:delegation:2' to='juliet@capulet.lit'>
        <delegated namespace='http://jabber.org/protocol/pubsub'/>
    </delegation>
</message>
```

5.2 Server Forward Stanza

The managing entity can now manage the namespace in a similar way as in admin mode but with different rules:

1. if the stanza is directed to the delegating server (no 'to' attribute, or 'to' attribute with the server own JID), **and** the 'from' attribute belong to a managed entity (e.g. *juliet@capulet.lit/balcony* if *juliet@capulet.lit* is a managed entity) **and** the namespace of the stanza has been delegated by the managed entity, then the delegating server MUST forward it to the managing entity accepted by the managed entity, in the same way as in admin mode.

2. if the stanza is directed to the bare jid (and only the bare jid) of the managed entity (e.g. the 'to' attribute is set to *juliet@capulet.lit* and the namespace has been delegated by the managed entity, then the server MUST forward it to the managing entity accepted by the managed entity, in the same way as in admin mode, for any 'from' attribute but the jid of the managing entity.

3. in all other cases, the server MUST NOT forward the stanza

6 Configuration

Server SHOULD provide a way for clients to check already delegated namespaces, and revoke them by using Ad-Hoc Commands (XEP-0050) [7] on the well-defined command node **'urn:xmpp:delegation:2#configure'**.
If present, the configuration commands MUST allow at least to check delegations granted to a managing entity, and to revoke them. A server MAY offer an option to keep delegations from one session to an other (see business rules).

7 Discovering Support

7.1 Announce

If a server or an entity supports the namespace delegation protocol, it MUST report that fact by including a service discovery feature of "*urn:xmpp:delegation:2*" in response to a Service Discovery (XEP-0030) [8] information request:

Listing 16: service discovery information request

```
<iq from='pubsub.capulet.lit'
    id='disco1'
    to='capulet.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 17: service discovery information response

```
<iq from='capulet.lit'
    id='disco1'
    to='pubsub.capulet.lit'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='urn:xmpp:delegation:2'/>
    ...
  </query>
</iq>
```

[7] XEP-0050: Ad-Hoc Commands <https://xmpp.org/extensions/xep-0050.html>.
[8] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

7.2 Nesting

7.2.1 General Case

When a server delegates a namespace to a managing entity, the later can have particular features which must be advertised by the former with disco protocol.
This is done by using a disco node, which is built in the following way: if pubsub.capulet.int manages pubsub namespace, it MUST report that fact in discovery feature, and have a '**urn:xmpp:delegation:2::http://jabber.org/protocol/pubsub**' node which reports the managed features.
The node name is obtained by concatenating this XEP namespace (**urn:xmpp:delegation:2**), a '**::**' separator, and the delegated namespace (here **http://jabber.org/protocol/pubsub**).The server MUST advertise the result in its own discovery answer, and MUST ignore features of its internal component (here internal PubSub service).
In the following example, the capulet.int server delegates its internal PEP component to pubsub.capulet.int. capulet.int only supports REQUIRED PubSub features and auto-create, while pubsub.capulet.int supports REQUIRED PubSub features and publish-options, but not auto-create. juliet@capulet.int asks its server what it is capable of, she is specially interested in PubSub capabilities.

Listing 18: Juliet asks her server its available features

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

Server delegates its PubSub namespace to *pubsub.capulet.lit*, so it asks its available features for this namespace like this:

Listing 19: capulet.lit requests disco infos for pubsub namespace to pubsub.capulet.lit

```
<iq from='capulet.lit'
    id='disco2'
    to='pubsub.capulet.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='urn:xmpp:delegation:2::http://jabber.org/protocol/
            pubsub'/>
</iq>
```

Note that in real situation, server has probably this information already in cache (see Implementation Notes). *pubsub.capulet.lit* returns its available features:

Listing 20: pubsub.capulet.lit returns features to nest

```
<iq from='pubsub.capulet.lit'
    id='disco2'
    to='capulet.lit'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='urn:xmpp:delegation:2::http://jabber.org/protocol/
             pubsub'>
        <feature var='http://jabber.org/protocol/pubsub'/>
        <feature var='http://jabber.org/protocol/pubsub#publish'/>
        <feature var='http://jabber.org/protocol/pubsub#subscribe'/>
        <feature var='http://jabber.org/protocol/pubsub#publish-
            options'/>
  </query>
</iq>
```

Server include the results in its own discovery info results:

Listing 21: capulet.lit return disco info including features from pubsub.capulet.lit

```
<iq from='capulet.lit'
    id='disco1'
    to='juliet@capulet.lit/balcony'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
        <feature var='urn:xmpp:delegation:2'/>
        ...
        <feature var='http://jabber.org/protocol/pubsub'/>
        <feature var='http://jabber.org/protocol/pubsub#publish'/>
        <feature var='http://jabber.org/protocol/pubsub#subscribe'/>
        <feature var='http://jabber.org/protocol/pubsub#publish-
            options'/>
        ...
  </query>
</iq>
```

Note that '*http://jabber.org/protocol/pubsub#auto-create*' is not available.
N.B.: In the special case of attribute filtering, the server still display managing entity's features for the whole delegated namespace instead of its own internal ones.

7.2.2 Nesting of Bare JID Disco Info

As an entity may ask for discovery information on bare JID, which the server would answer, the managing entity must be able to send this kind of information.
To do so, the mechanism is the same as for server features, but the separator is '**:bare:**' instead of '**::**':

Listing 22: Juliet Asks Features for Her Own Bare JID

```
<iq from='juliet@capulet.lit/balcony'
    id='disco3'
    to='juliet@capulet.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Server delegate its PubSub namespace to *pubsub.capulet.lit*, so it ask its available features for this namespace like this:

Listing 23: capulet.lit requests disco infos for pubsub namespace to pubsub.capulet.lit

```
<iq from='capulet.lit'
    id='disco4'
    to='pubsub.capulet.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='urn:xmpp:delegation:2:bare:http://jabber.org/protocol/
            pubsub'/>
</iq>
```

As for general case, server has probably this information already in cache. *pubsub.capulet.lit* returns its available features:

Listing 24: pubsub.capulet.lit returns features to nest

```
<iq from='pubsub.capulet.lit'
    id='disco4'
    to='capulet.lit'
    type='result'>
    <query xmlns='http://jabber.org/protocol/disco#info'
           node='urn:xmpp:delegation:2:bare:http://jabber.org/protocol
              /pubsub'>
        <identity category='pubsub' type='pep'/>
        <feature var='http://jabber.org/protocol/pubsub#access-
           presence'/>
        <feature var='http://jabber.org/protocol/pubsub#auto-create'/>
        <feature var='http://jabber.org/protocol/pubsub#auto-subscribe
           '/>
        <feature var='http://jabber.org/protocol/pubsub#config-node'/>
        <feature var='http://jabber.org/protocol/pubsub#create-and-
           configure'/>
        <feature var='http://jabber.org/protocol/pubsub#create-nodes'/
           >
        <feature var='http://jabber.org/protocol/pubsub#filtered-
           notifications'/>
        <feature var='http://jabber.org/protocol/pubsub#persistent-
           items'/>
```

```
        <feature var='http://jabber.org/protocol/pubsub#publish'/>
        <feature var='http://jabber.org/protocol/pubsub#retrieve-items
            '/>
        <feature var='http://jabber.org/protocol/pubsub#subscribe'/>
        ...
    </query>
</iq>
```

Then the server returns the answer to Juliet, as in general case, with requested bare JID in 'from' field.

Listing 25: capulet.lit returns disco info to Juliet

```
<iq from='juliet@capulet.lit'
    id='disco3'
    to='juliet@capulet.lit/balcony'
    type='result'>
    <query xmlns='http://jabber.org/protocol/disco#info'>
        <identity category='account' type='registered'/>
        <identity category='pubsub' type='pep'/>
        <feature var='http://jabber.org/protocol/pubsub#access-
            presence'/>
        <feature var='http://jabber.org/protocol/pubsub#auto-create'/>
        <feature var='http://jabber.org/protocol/pubsub#auto-subscribe
            '/>
        <feature var='http://jabber.org/protocol/pubsub#config-node'/>
        <feature var='http://jabber.org/protocol/pubsub#create-and-
            configure'/>
        <feature var='http://jabber.org/protocol/pubsub#create-nodes'/
            >
        <feature var='http://jabber.org/protocol/pubsub#filtered-
            notifications'/>
        <feature var='http://jabber.org/protocol/pubsub#persistent-
            items'/>
        <feature var='http://jabber.org/protocol/pubsub#publish'/>
        <feature var='http://jabber.org/protocol/pubsub#retrieve-items
            '/>
        <feature var='http://jabber.org/protocol/pubsub#subscribe'/>
        ...
    </query>
</iq>
```

7.2.3 Service Discovery Extensions

Extensions of Service Discovery as specified in Service Discovery Extensions (XEP-0128) [9] follow the same rules: for the delegated namespace, internal extensions MUST be removed

[9] XEP-0128: Service Discovery Extensions <https://xmpp.org/extensions/xep-0128.html>.

and extensions of managing entity MUST be included instead.

7.2.4 Remaining Discovery Infos

It may be necessary for a managing entity to advertise feature on a node which is unknown at the time of configuration. This is notably the case for a PEP component: features must be advertisable on disco nodes mapping pubsub nodes.
To allows that, the special namespace '**urn:xmpp:delegation:2:bare:disco#info:***' is used, and means *"delegate information discovery requests made on bare jids for any discovery node which is not already managed by the server"*.

Listing 26: Juliet Asks Features for the Microblog Node 'urn:xmpp:microblog:0' for her Own Bare JID

```
<iq from='juliet@capulet.lit/balcony'
    id='disco1'
    to='juliet@capulet.lit'
    type='get'>
    <query xmlns='http://jabber.org/protocol/disco#info' node='
       urn:xmpp:microblog:0'/>
</iq>
```

The node '*urn:xmpp:microblog:0*' is not managed by the server. Normally, this would result as an *<item-not-found>* <iq/> error, but the namespace '**urn:xmpp:delegation:2:bare:disco#info:***' is delegated to *pubsub.capulet.lit*, thus the server delegate the stanza to the managing entity:

Listing 27: Server Delegates Disco Request to pubsub.capulet.lit

```
<iq from='capulet.lit'
    to='pubsub.capulet.lit'
    id='delegate_disco_1'
    type='set'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq from='juliet@capulet.lit/balcony'
                id='disco1'
                to='juliet@capulet.lit'
                type='get'>
                <query xmlns='http://jabber.org/protocol/disco#info'
                    node='urn:xmpp:microblog:0'/>
            </iq>
        </forwarded>
    </delegation>
</iq>
```

The managing entity answers with its own disco features for the requested node:

Listing 28: pubsub.capulet.lit Replies With its Disco Infos for this Node

```
<iq from='pubsub.capulet.lit'
    to='capulet.lit'
    id='delegate_disco_1'
    type='result'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq to='juliet@capulet.lit/balcony'
                id='disco1'
                type='result'
                xmlns='jabber:client'>
                <query xmlns="http://jabber.org/protocol/disco#info"
                    node="urn:xmpp:microblog:0">
                    <identity type="leaf" category="pubsub"/>
                    <x xmlns="jabber:x:data" type="result">
                        <field var="FORM_TYPE" type="hidden">
                            <value>http://jabber.org/protocol/pubsub#
                                meta-data</value>
                        </field>
                        <field var="pubsub#node_type" type="text-
                            single">
                            <value>leaf</value>
                        </field>
                        <field var="pubsub#persist_items" type="
                            boolean">
                            <value>true</value>
                        </field>
                        <field var="pubsub#max_items" type="text-
                            single">
                            <value>max</value>
                        </field>
                        <field var="pubsub#access_model" type="list-
                            single">
                            <value>whitelist</value>
                        </field>
                    </x>
                    <feature var="http://jabber.org/protocol/rsm"/>
                    <feature var="http://jabber.org/protocol/pubsub#
                        rsm"/>
                    <feature var="urn:xmpp:order-by:1"/>
                </query>
            </iq>
        </forwarded>
    </delegation>
</iq>
```

Then the server decapsulate the <iq/> result, validate it as explained in Server Forwards Delegated <iq/> Stanza, and send the result to Juliet:

Listing 29: capulet.lit Sends the Disco Infos to Juliet

```
<iq to='juliet@capulet.lit/balcony'
    id='disco1'
    type='result'
    xmlns='jabber:client'>
    <query xmlns="http://jabber.org/protocol/disco#info" node="
        urn:xmpp:microblog:0">
        <identity type="leaf" category="pubsub"/>
        <x xmlns="jabber:x:data" type="result">
            <field var="FORM_TYPE" type="hidden">
                <value>http://jabber.org/protocol/pubsub#meta-data</
                    value>
            </field>
            <field var="pubsub#node_type" type="text-single">
                <value>leaf</value>
            </field>
            <field var="pubsub#persist_items" type="boolean">
                <value>true</value>
            </field>
            <field var="pubsub#max_items" type="text-single">
                <value>max</value>
            </field>
            <field var="pubsub#access_model" type="list-single">
                <value>whitelist</value>
            </field>
        </x>
        <feature var="http://jabber.org/protocol/rsm"/>
        <feature var="http://jabber.org/protocol/pubsub#rsm"/>
        <feature var="urn:xmpp:order-by:1"/>
    </query>
</iq>
```

7.2.5 Bare JID Disco Items

A managing entity must be able to answer items discovery requests on bare jids. For PEP components, this is used to show available nodes on the service. Furthermore, An information discovery request or an items discovery requests on a node corresponding to an item advertised by the managing entity must be forwarded to the managing entity too.
This is done by using the special namespace '**urn:xmpp:delegation:2:bare:disco#items:***' which means *"delegate items discovery made on bare jids with no node specified and all items discovery requests made on bare jids with a node which is not already managed by the server"*.
Note: When no node is used, and unlike the *disco#info* case, the server SHOULD NOT nest its internal items with the ones from managing entity. The reason is that bare jids disco items are usually used for PEP nodes, and the server would then mix internal PEP nodes with the managing entity nodes, which would be confusing for XMPP clients and end-users. The server implementation MAY choose to nest items anyway, in which case the mechanism is the same as for disco#info.

A server SHOULD NOT cache result of forwarded disco#items requests, as this is highly dynamic.

Listing 30: Juliet Discover Items from her Own Bare JID

```
<iq from='juliet@capulet.lit/balcony'
    id='disco_items_1'
    to='juliet@capulet.lit'
    type='get'>
    <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

The namespace **'urn:xmpp:delegation:2:bare:disco#items:*'** is delegated to *pubsub.capulet.lit*, thus the server delegate the stanza to the managing entity:

Listing 31: Server Delegates Disco Items Request to pubsub.capulet.lit

```
<iq from='capulet.lit'
    to='pubsub.capulet.lit'
    id='delegate_disco_items_1'
    type='set'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq from='juliet@capulet.lit/balcony'
                id='disco_items_1'
                to='juliet@capulet.lit'
                type='get'>
                <query xmlns='http://jabber.org/protocol/disco#items'/
                    >
            </iq>
        </forwarded>
    </delegation>
</iq>
```

The managing entity answers with its own disco items for Juliet's JID:

Listing 32: pubsub.capulet.lit Replies With its Disco Items for this JID

```
<iq from='pubsub.capulet.lit'
    to='capulet.lit'
    id='delegate_disco_items_1'
    type='result'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq to='juliet@capulet.lit/balcony'
                id='disco_items_1'
                type='result'
                xmlns='jabber:client'>
                <query xmlns="http://jabber.org/protocol/disco#items">
```

```
                    <item jid="juliet@capulet.int" node="http://jabber
                        .org/protocol/mood"/>
                    <item jid="juliet@capulet.int" node="http://jabber
                        .org/protocol/activity"/>…

                    <item jid="juliet@capulet.int" node="
                        urn:xmpp:microblog:0"/>…

                </query>
            </iq>
        </forwarded>
    </delegation>
</iq>
```

Then the server decapsulate the <iq/> result, validate it as explained in Server Forwards Delegated <iq/> Stanza, and send the result to Juliet:

Listing 33: capulet.lit Sends the Disco Items to Juliet

```
<iq to='juliet@capulet.lit/balcony'
    id='disco_items_1'
    type='result'
    xmlns='jabber:client'>
    <query xmlns="http://jabber.org/protocol/disco#items">
        <item jid="juliet@capulet.int" node="http://jabber.org/
            protocol/mood"/>
        <item jid="juliet@capulet.int" node="http://jabber.org/
            protocol/activity"/>…

        <item jid="juliet@capulet.int" node="urn:xmpp:microblog:0"/>…

    </query>
</iq>
```

Juliet now wants to know the IDs of blog items which has been published on her blog, to do so she does an items discovery request on her own bare jid with the '*urn:xmpp:microblog:0*' node:

Listing 34: Juliet Discovers Items for the Microblog Node 'urn:xmpp:microblog:0' on her Own Bare JID

```
<iq from='juliet@capulet.lit/balcony'
    id='disco_items_2'
    to='juliet@capulet.lit'
    type='get'>
    <query xmlns='http://jabber.org/protocol/disco#items' node='
        urn:xmpp:microblog:0'/>
</iq>
```

Because the special namespace '**urn:xmpp:delegation:2:bare:disco#items:***' is delegated to *pubsub.capulet.lit*, the server forward the request to this managing entity:

Listing 35: Server Delegates Disco Items Request to pubsub.capulet.lit

```
<iq from='capulet.lit'
    to='pubsub.capulet.lit'
    id='delegate_disco_2'
    type='set'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq from='juliet@capulet.lit/balcony'
                id='disco_items_2'
                to='juliet@capulet.lit'
                type='get'>
                <query xmlns='http://jabber.org/protocol/disco#items'
                    node='urn:xmpp:microblog:0'/>
            </iq>
        </forwarded>
    </delegation>
</iq>
```

The managing entity answers with its own disco items for the requested entity and node (*pubsub.capulet.lit* being a PEP service, the items are actually Juliet's blog items IDs):

Listing 36: pubsub.capulet.lit Replies With its Disco Items for this Entity and Node

```
<iq from='pubsub.capulet.lit'
    to='capulet.lit'
    id='delegate_disco_2'
    type='result'>
    <delegation xmlns='urn:xmpp:delegation:2'>
        <forwarded xmlns='urn:xmpp:forward:0'>
            <iq to='juliet@capulet.lit/balcony'
                id='disco_items_2'
                type='result'
                xmlns='jabber:client'>
                <query xmlns="http://jabber.org/protocol/disco#items"
                    node="urn:xmpp:microblog:0">
                    <item name="o-romeo-romeo-wherefore-art-thou-romeo
                        -G1aF" jid="juliet@capulet.lit"/>
                    <item name="thou-know-st-the-mask-of-night-is-on-
                        my-face-2avF" jid="juliet@capulet.lit"/>
                    <item name="balcony-and-privacy-afFG" jid="
                        juliet@capulet.lit"/>
                </query>
            </iq>
        </forwarded>
    </delegation>
```

```
</iq>
```

The server decapsulates the <iq/> result, validate it as explained in Server Forwards Delegated <iq/> Stanza, and send the result to Juliet:

Listing 37: capulet.lit Sends the Disco Items to Juliet

```
<iq to='juliet@capulet.lit/balcony'
    id='disco_items_2'
    type='result'
    xmlns='jabber:client'>
    <query xmlns="http://jabber.org/protocol/disco#items" node="
        urn:xmpp:microblog:0">
        <item name="o-romeo-romeo-wherefore-art-thou-romeo-G1aF" jid="
            juliet@capulet.lit"/>
        <item name="thou-know-st-the-mask-of-night-is-on-my-face-2avF"
             jid="juliet@capulet.lit"/>
        <item name="balcony-and-privacy-afFG" jid="juliet@capulet.lit"
            />
    </query>
</iq>
```

8 Business Rules

1. In client mode, server MAY keep delegations granted to an entity by a client from one session to an other, but if it does so, it MUST provide configuration like explained in the suitable section. If server offers this feature, it SHOULD add a field directly in configuration commands.

2. If a client can't check or revoke delegations (i.e. it doesn't support Ad-Hoc Commands (XEP-0050) [10]) when granting them, the server MUST NOT keep granted delegations from one session to an other, and delegations will be asked on each new session.

3. If delegations are changed during a session, server MUST notify managing entity of the new delegations, like in client delegation request use case.

4. If delegations are kept between sessions in client mode, and the managing entity is not available or return an error, the server MUST return an <iq/> error with condition <service-unavailable/> when a delegated namespace is requested, like explained in "Server Forwards Delegated <iq/> Stanza" section.

5. The namespace of this XEP (**urn:xmpp:delegation:2**) MUST NOT be delegated. If an entity requests it, the server MUST return a <forbidden/> error.

[10] XEP-0050: Ad-Hoc Commands <https://xmpp.org/extensions/xep-0050.html>.

9 Implementation Notes

1. As admin mode is far more easy to implement than client mode, and client mode may impact performances, a server MAY choose to only implement the former.

2. Because of the performance impact, a server SHOULD ask for disco features to nest to managing entity when delegation is accepted, and keep them in cache.

3. RFC 6120 [11] section 10.1 require in-order processing of stanzas, which may be problematic for this extension: either the server blocks the traffic until the managing entity answers - which can lead to severe performance impact -, or the server doesn't block and may loose order. The recommended way is to not block the traffic while waiting for managing entity answer to avoid performance issues. A future version of this XEP may include an attribute to request traffic blocking. In admin mode the server implementation MAY chooses to have a blocking option (which SHOULD be per namespace, not global).

10 Security Considerations

1. Managing entity can manage sensitive data, *admin* delegation should be granted carefuly, only if you absolutely trust the entity.

2. A server MAY choose to filter allowed namespaces. In this case, it MUST always set the allowed type of filtered namespaces to **0**.

3. In case of filtering, an allowlist system is more secure and SHOULD be prefered to a blocklist (idealy, configuration would allow no filtering, allowlist filtering and blocklist filtering).

4. Managing entity must be careful about not leaking the existence of an account (see XEP-0030 Security Considerations).

11 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [12].

[11] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[12] 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/>.

12 XMPP Registrar Considerations

12.1 Protocol Namespaces

The XMPP Registrar [13] includes 'urn:xmpp:delegation:2' in its registry of protocol namespaces (see <https://xmpp.org/registrar/namespaces.html>).

- urn:xmpp:delegation:2

12.2 Protocol Versioning

If the protocol defined in this specification undergoes a revision that is not fully backwards-compatible with an older version, the XMPP Registrar shall increment the protocol version number found at the end of the XML namespaces defined herein, as described in Section 4 of XEP-0053.

13 XML Schema

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='urn:xmpp:delegation:2'
    xmlns='urn:xmpp:delegation:2'
    elementFormDefault='qualified'>

  <xs:import namespace='urn:xmpp:forward:0'
      schemaLocation='http://xmpp.org/schemas/forward.xsd'/>

  <xs:element name='query'>
      <xs:complexType>
          <xs:attribute name='to' use='required' type='xs:string'/>
          </xs:attribute>
          <xs:element name='delegate'
                      maxOccurs='unbounded'>
              <xs:complexType>
                  <xs:attribute name='namespace' use='required' type='
                     xs:string'/>
              </xs:complexType>
          </xs:element>
      </xs:complexType>
  </xs:element>
```

[13] 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/>.

```
  <xs:element name='delegation'>
      <xs:complexType>
          <xs:element ref='urn:xmpp:forward:0' minOccurs='0' maxOccurs
              ='1' />
          <xs:element name='delegated'
                      minOccurs='0'
                      maxOccurs='unbounded'>
              <xs:complexType>
                  <xs:attribute name='namespace' use='required' type='
                      xs:string'/>
                  <xs:element name='attribute'
                              minOccurs='0'
                              maxOccurs='unbounded'>
                      <xs:complexType>
                          <xs:attribute name='name' use='required'
                              type='xs:string'/>
                      </xs:complexType>
                  </xs:element>
              </xs:complexType>
          </xs:element>
      </xs:complexType>
  </xs:element>

</xs:schema>
```

14 Acknowledgements

This XEP is linked with Privileged Entity (XEP-0356) [14] and works in a similar way.
The client mode delegation mechanism is inspired from Remote Roster Management (XEP-0321) [15] permission request.
Thanks to Adrien Cossa for his typos/style corrections
Thanks to Philipp Hancke, Dave Cridland, Kurt Zeilenga, Sergey Dobrov and □□□□ □□□□□□□□ for their feedbacks.

[14] XEP-0356: Privileged Entity <https://xmpp.org/extensions/xep-0356.html>.
[15] XEP-0321: Remote Roster Management <https://xmpp.org/extensions/xep-0321.html>.