XEP-0060: Publish-Subscribe

Peter Millard

Peter Saint-Andre
mailto:stpeter@stpeter.im
xmpp:stpeter@jabber.org
https://stpeter.im/

Ralph Meijer
mailto:ralphm@ik.nu
xmpp:ralphm@ik.nu

2023-09-07
Version 1.26.0

Status	Type	Short Name
Draft	Standards Track	pubsub

This specification defines an XMPP protocol extension for generic publish-subscribe functionality. The protocol enables XMPP entities to create nodes (topics) at a pubsub service and publish information at those nodes; an event notification (with or without payload) is then broadcasted to all entities that have subscribed to the node. Pubsub therefore adheres to the classic Observer design pattern and can serve as the foundation for a wide variety of applications, including news feeds, content syndication, rich presence, geolocation, workflow systems, network management systems, and any other application that requires event notifications.

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 How It Works . . . 1

2 Glossary **4**

3 Requirements **6**

4 Preliminaries **7**
- 4.1 Affiliations . . . 7
- 4.2 Subscription States . . . 8
- 4.3 Event Types . . . 8
- 4.4 Node Types . . . 9
- 4.5 Node Access Models . . . 10
- 4.6 Addressing . . . 11
  - 4.6.1 JID . . . 11
  - 4.6.2 JID+NodeID . . . 11

5 Entity Use Cases **12**
- 5.1 Discover Features . . . 12
- 5.2 Discover Nodes . . . 13
- 5.3 Discover Node Information . . . 14
- 5.4 Discover Node Metadata . . . 15
- 5.5 Discover Items for a Node . . . 18
- 5.6 Retrieve Subscriptions . . . 18
- 5.7 Retrieve Affiliations . . . 21

6 Subscriber Use Cases **23**
- 6.1 Subscribe to a Node . . . 23
  - 6.1.1 Request . . . 23
  - 6.1.2 Success Case . . . 24
  - 6.1.3 Error Cases . . . 24
  - 6.1.4 Approval Required . . . 29
  - 6.1.5 Configuration Required . . . 29
  - 6.1.6 Multiple Subscriptions . . . 31
  - 6.1.7 Receiving the Last Published Item . . . 32
- 6.2 Unsubscribe from a Node . . . 33
  - 6.2.1 Request . . . 33
  - 6.2.2 Success Case . . . 33
  - 6.2.3 Error Cases . . . 34
- 6.3 Configure Subscription Options . . . 36
  - 6.3.1 Advertising Support . . . 36
  - 6.3.2 Request . . . 36

6.3.3 Success Case . . . 37
6.3.4 Error Cases . . . 38
6.3.5 Form Submission . . . 41
6.3.6 Form Processing . . . 42
6.3.7 Subscribe and Configure . . . 42
6.4 Request Default Subscription Configuration Options . . . 44
6.4.1 Request . . . 44
6.4.2 Success Case . . . 45
6.4.3 Error Cases . . . 45
6.5 Retrieve Items from a Node . . . 46
6.5.1 Permissions . . . 47
6.5.2 Requesting All Items . . . 47
6.5.3 Returning All Items . . . 47
6.5.4 Returning Some Items . . . 49
6.5.5 Returning the Last Published Item . . . 51
6.5.6 Returning Notifications Only . . . 51
6.5.7 Requesting the Most Recent Items . . . 53
6.5.8 Requesting a Particular Item . . . 54
6.5.9 Error Cases . . . 55

7 Publisher Use Cases **59**
7.1 Publish an Item to a Node . . . 59
7.1.1 Request . . . 59
7.1.2 Success Case . . . 61
7.1.3 Error Cases . . . 67
7.1.4 Automatic Node Creation . . . 71
7.1.5 Publishing Options . . . 71
7.2 Delete an Item from a Node . . . 72
7.2.1 Request . . . 72
7.2.2 Success Case . . . 73
7.2.3 Error Cases . . . 74

8 Owner Use Cases **76**
8.1 Create a Node . . . 76
8.1.1 General Considerations . . . 76
8.1.2 Create a Node With Default Configuration . . . 80
8.1.3 Create and Configure a Node . . . 81
8.2 Configure a Node . . . 83
8.2.1 Request . . . 83
8.2.2 Success Case . . . 83
8.2.3 Error Cases . . . 86
8.2.4 Form Submission . . . 88
8.2.5 Form Processing . . . 90

8.3 Request Default Node Configuration Options . . . 92
8.3.1 Request . . . 92
8.3.2 Success Case . . . 92
8.3.3 Error Cases . . . 95
8.4 Delete a Node . . . 96
8.4.1 Request . . . 96
8.4.2 Success Case . . . 97
8.4.3 Error Cases . . . 97
8.5 Purge All Node Items . . . 98
8.5.1 Request . . . 98
8.5.2 Success Case . . . 99
8.5.3 Error Cases . . . 99
8.6 Manage Subscription Requests . . . 101
8.7 Process Pending Subscription Requests . . . 104
8.7.1 Request . . . 104
8.7.2 Success Case . . . 105
8.7.3 Error Cases . . . 105
8.7.4 Per-Node Request . . . 107
8.8 Manage Subscriptions . . . 108
8.8.1 Retrieve Subscriptions List . . . 108
8.8.2 Modify Subscriptions . . . 110
8.8.3 Delete a Subscriber . . . 112
8.8.4 Notifying Subscribers . . . 113
8.9 Manage Affiliations . . . 113
8.9.1 Retrieve Affiliations List . . . 113
8.9.2 Modify Affiliation . . . 115
8.9.3 Delete an Entity . . . 118
8.9.4 Notifying Entities . . . 118

9 IM Account Integration 118
9.1 Auto-Subscribe . . . 119
9.1.1 Account Owner . . . 119
9.1.2 Presence Subscriber . . . 120
9.1.3 Presence Sharer . . . 120
9.2 Filtered Notifications . . . 120

10 Feature Summary 123

11 Error Conditions 127

12 Implementation Notes 128
12.1 Data Model . . . 128
12.2 Notification Triggers . . . 129
12.3 Intended Recipients for Notifications . . . 129

12.4 Handling Notification-Related Errors . . . 130
12.5 Temporary Subscriptions . . . 130
12.6 Presence-Based Delivery of Events . . . 131
12.7 Not Routing Events to Offline Storage . . . 132
12.8 Including a Message Body . . . 132
12.9 Node ID and Item ID Uniqueness . . . 132
12.10 Item Caching . . . 133
12.11 Batch Processing . . . 133
12.12 Auto-Subscribing Owners and Publishers . . . 133
12.13 Authorizing Subscription Requests (Pending Subscribers) . . . 134
12.14 Notification of Subscription State Changes . . . 134
12.15 NodeID Semantics . . . 135
12.16 Inclusion of SHIM Headers . . . 135
12.17 Associating Events and Payloads with the Generating Entity . . . 136
12.18 Chaining . . . 137
12.19 Time-Based Subscriptions (Leases) . . . 137
12.20 Content-Based Pubsub Systems . . . 141
12.21 Singleton Nodes . . . 144
12.22 PubSub URIs . . . 144
12.23 Integration with Data Forms . . . 145

13 Internationalization Considerations **146**
13.1 Field Labels . . . 146

14 Security Considerations **146**
14.1 Private Information . . . 146
14.2 Authorization . . . 146
14.3 Access Models . . . 146
14.4 Presence Leaks . . . 147

15 IANA Considerations **147**

16 XMPP Registrar Considerations **147**
16.1 Protocol Namespaces . . . 147
16.2 Service Discovery Category/Type . . . 148
16.3 Service Discovery Features . . . 148
16.4 Field Standardization . . . 154
16.4.1 pubsub#subscribe_authorization FORM_TYPE . . . 154
16.4.2 pubsub#subscribe_options FORM_TYPE . . . 155
16.4.3 pubsub#meta-data FORM_TYPE . . . 156
16.4.4 pubsub#node_config FORM_TYPE . . . 157
16.5 SHIM Headers . . . 162
16.6 URI Query Types . . . 163
16.7 Data Forms Validation Datatypes . . . 164

17 XML Schemas **165**
17.1 http://jabber.org/protocol/pubsub 165
17.2 http://jabber.org/protocol/pubsub#errors 170
17.3 http://jabber.org/protocol/pubsub#event 172
17.4 http://jabber.org/protocol/pubsub#owner 175

18 Acknowledgements **179**

19 Author Note **179**

1 Introduction

1.1 Overview

The XMPP publish-subscribe extension defined in this document provides a framework for a wide variety of applications, including news feeds, content syndication, extended presence, geolocation, avatar management, shared bookmarks, auction and trading systems, workflow systems, network management systems, NNTP gateways, profile management, and any other application that requires event notifications.
This technology uses the classic "publish-subscribe" or "observer" design pattern: a person or application publishes information, and an event notification (with or without payload) is broadcasted to all authorized subscribers. In general, the relationship between the publisher and subscriber is mediated by a service that receives publication requests, broadcasts event notifications to subscribers, and enables privileged entities to manage lists of people or applications that are authorized to publish or subscribe. The focal point for publication and subscription is a "node" to which publishers send data and from which subscribers receive event notifications. Nodes can also maintain a history of events and provide other services that supplement the pure pubsub model.
This document defines a generic protocol that all pubsub applications can use. Compliant implementations are not required to implement all of the features defined here (see the Feature Summary.) Other specifications may define "subsets" or "profiles" of publish-subscribe for use in specialized contexts, but such profiles are out of scope for this document.

1.2 How It Works

Although this specification is large because it defines many side use cases and possible error flows, the basic idea is simple:

1. An entity publishes information to a node at a publish-subscribe service.
2. The pubsub service pushes an event notification to all entities that are authorized to learn about the published information.

Perhaps the most popular application of pubsub-like functionality is content syndication, which has become familiar from the RSS and Atom (RFC 4287 [1]) feeds associated with weblogs, news sites, and other frequently-updated information available on the Internet. Consider the example of a weblog published by <hamlet@denmark.lit>. When Hamlet writes a new weblog entry, his blogging software publishes the entry to a pubsub node hosted at <pubsub.shakespeare.lit>:

Listing 1: Publisher Publishes a New Weblog Entry

[1] RFC 4287: The Atom Syndication Format <http://tools.ietf.org/html/rfc4287>.

```
<iq type='set'
    from='hamlet@denmark.lit/blogbot'
    to='pubsub.shakespeare.lit'
    id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='princely_musings'>
      <item>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
Whether 'tis␣nobler␣in␣the␣mind␣to␣suffer
The␣slings␣and␣arrows␣of␣outrageous␣fortune,
Or␣to␣take␣arms␣against␣a␣sea␣of␣troubles,
And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/atom03'/>
␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
␣␣␣␣␣␣␣␣␣␣</entry>
␣␣␣␣␣␣␣␣␣␣</item>
␣␣␣␣␣␣␣␣␣␣</publish>
␣␣␣␣␣␣␣␣␣␣</pubsub>
</iq>
```

So that is the "pub" part of pubsub.
Now the pubsub service notifies all the subscribers about the new blog entry:

Listing 2: Service Notifies Subscribers

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        [ ... ENTRY ... ]
      </item>
    </items>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='bernardo@denmark.lit' id='
    bar'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        [ ... ENTRY ... ]
```

```
      </item>
    </items>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='horatio@denmark.lit' id='
    baz'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        [ ... ENTRY ... ]
      </item>
    </items>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='bard@shakespeare.lit' id='
    fez'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        [ ... ENTRY ... ]
      </item>
    </items>
  </event>
</message>
```

Here is an even simpler example: a transient node that sends only event notifications without a payload:

Listing 3: A Transient Notification

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='elsinore/doorbell'/>
  </event>
</message>
```

Naturally, the entities involved may need to complete other use cases in order to enable full pubsub functionality -- for example, the publisher may need to create the node (see Create a Node) and subscribers may need to sign up for event notifications (see Subscribe to a Node). These use cases are fully described in the remainder of this document. (For information about which features are required and which are recommended or optional, consult the Feature Summary.)

2 Glossary

The following terms are used throughout this document to refer to elements, objects, or actions that occur in the context of a pubsub service. (Note: Some of these terms are specified in greater detail within the body of this document.)

Authorize Access Model A node access model under which an entity can subscribe only through having a subscription request approved by a node owner (subscription requests are accepted but only provisionally) and only subscribers may retrieve items.

Address (1) A JID as defined in XMPP Core RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>., or (2) the combination of a JID and a Service Discovery (XEP-0030) XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>. node.

Collection Node A type of node that contains nodes and/or other collections but no published items. Collections make it possible to represent more sophisticated relationships among nodes. Collection nodes are defined in PubSub Collection Nodes (XEP-0248) XEP-0248: PubSub Collection Nodes <https://xmpp.org/extensions/xep-0248.html>..

Entity A JID-addressable XMPP entity (client, service, application, etc.).

Event A change in the state of a node.

Instant Node A node whose NodeID is automatically generated by a pubsub service.

Item An XML fragment which is published to a node, thereby generating an event.

ItemID A unique identifier for an item in the context of a specific node.

Leaf Node A type of node that contains published items only. It is NOT a container for other nodes.

Node A virtual location to which information can be published and from which event notifications and/or payloads can be received (in other pubsub systems, this may be labelled a ”topic”).

NodeID The unique identifier for a node within the context of a pubsub service. The NodeID is either supplied by the node creator or generated by the pubsub service (if the node creator requests an Instant Node). The NodeID MAY have semantic meaning (e.g., in some systems or in pubsub profiles such as PEP the NodeID might be an XML namespace for the associated payload), but such meaning is OPTIONAL. If a document defines a given NodeID as unique within the realm of XMPP pubsub systems, it MUST specify the XML namespace of the associated payload.

Notification A message sent to a subscriber informing them of an event.

Outcast An entity that is disallowed from subscribing or publishing to a node.

Owner The manager of a node, of which there may be more than one; often but not necessarily the node creator.

Open Access Model A node access model under which any entity may subscribe and retrieve items without approval.

Payload The XML data that is contained within the <item/> element of a publication request or an event notification. A given payload is defined by an XML namespace and associated schema. A document that defines a payload format SHOULD specify whether the payload can be used only for a NodeID whose value is the same as the XML namespace, or whether the payload can be used for any NodeID. Such a document also SHOULD specify whether it is suggested that node at which such payloads are published are best configured as Singleton Nodes.

Personal Eventing A simplified subset of Publish-Subscribe for use in the context of instant messaging and presence applications, whereby each IM user's JID is a virtual pubsub service; for details, see Personal Eventing Protocol (XEP-0163) XEP-0163: Personal Eventing Protocol <https://xmpp.org/extensions/xep-0163.html>..

Presence Access Model A node access model under which any entity that is subscribed to the owner's presence with a subscription of type "from" or "both" (see RFC 3921 RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc3921>.) may subscribe to the node and retrieve items from the node; this access model applies mainly to instant messaging systems.

Publisher An entity that is allowed to publish items to a node and that is automatically subscribed to the node.

Publish-Only An entity that is allowed to publish items to a node but that is not allowed to receive notifications. (This affiliation is useful in the context of nodes that do not have an open access model when automated entities need to generate notifications on behalf of the owner.)

Pubsub Service An XMPP server or component that adheres to the protocol defined herein.

Roster Access Model A node access model under which any entity that is subscribed to the owner's presence and in the specified roster group(s) may subscribe to the node and retrieve items from the node; this access model applies mainly to instant messaging systems.

Subscriber An entity that is subscribed to a node.

Whitelist Access Model A node access model under which an entity may subscribe and retrieve items only if explicitly allowed to do so by the node owner (subscription requests from unauthorized entities are rejected).

3 Requirements

Requirements for a pubsub service can be driven by end-user needs as well as the needs of other components and services which can use the service. First, a pubsub service implemented using XMPP MUST provide the basic features that implement a pure publish-subscribe pattern:

- An entity MUST be able to publish events to a service such that all subscribers to a node receive notification of the event. See Publish an Item to a Node.
- An entity MUST be able to subscribe to a node (or be informed that subscription is not allowed). See Subscribe to a Node.
- An entity MUST be allowed to be affiliated with a node. Allowable affiliations are member, none, outcast, owner, publish-only, and publisher. Implementations MUST support affiliations of none and owner, and MAY support affiliations of member, outcast, publisher, and publish-only. See Affiliations.
- An entity MUST be allowed to query the pubsub service (or a specific node) to determine what optional features of this specification the service (or node) implements. This query MUST use the Service Discovery (disco#info) protocol. See Discover Node Information.

Some of the possible uses of an XMPP-based pubsub service will require other features, but these features are OPTIONAL and therefore not mandatory for compliance with this specification. However, if these features are implemented, they MUST adhere to the protocol described herein in to be compliant. These features include:

- A service MAY cache the last item published to a node (even if the "persistent-items" option is set to false); if it does default "cache-last-item" to true, it SHOULD send the last published item (or notification thereof) to subscribed entities based on configuration of the "send_last_published_item" field.
- A node owner SHOULD be able to specify who may subscribe to a node.
- A node owner SHOULD be able to specify who may publish to a node.
- A node MAY be configured to deliver the published payload inside the event notification.
- A node MAY be configured to persist published items to some persistent storage mechanism.
- A node MAY be configured to persist only a limited number of items.
- A service MAY support collections as described in XEP-0248.
- A service or node MAY support extended service discovery information (metadata).

4 Preliminaries

4.1 Affiliations

To manage permissions, the protocol defined herein uses a hierarchy of affiliations, similiar to those introduced in Multi-User Chat (XEP-0045) [2].
All affiliations MUST be based on a bare JID (<localpart@domain.tld> or <domain.tld>) instead of a full JID (<localpart@domain.tld/resource> or <domain.tld/resource>).
Support for the "owner" and "none" affiliations is REQUIRED. Support for all other affiliations is RECOMMENDED. For each non-required affiliation supported by an implementation, it SHOULD return a service discovery feature of "name-affiliation" where "name" is the name of the affiliation, such as "member", "outcast", or "publisher" (see the Feature Summary). Particular kinds of pubsub services MAY enforce additional requirements (e.g., requiring support for a given non-required affiliation or for all affiliations).

Affiliation	Subscribe	Retrieve Items	Publish Items	Delete Single Item	Purge Node	Configure Node	Delete Node
Owner	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Publisher	Yes	Yes	Yes	Yes *	Yes *	No	No
Publish-Only	No	No	Yes	Yes *	No *	No	No
Member	Yes	Yes	No	No	No	No	No
None	Yes	No	No	No	No	No	No
Outcast	No	No	No	No	No	No	No

* Note: A service MAY allow any publisher to delete / purge any item once it has been published to that node instead of allowing only the original publisher to remove it. This behavior is NOT RECOMMENDED for the publish-only affiliation, which SHOULD be allowed to delete only items that the publish-only entity has published.
The ways in which an entity changes its affiliation with a node are well-defined. Typically, action by an owner is required to make an affiliation state transition. Affiliation changes and their triggering actions are specified in the following table.

	Outcast	None	Member	Publisher	Owner
Outcast	--	Owner removes ban	Owner adds entity to member list	Owner adds entity to publisher list	Owner adds entity to owner list
None	Owner bans entity	--	Owner adds entity to member list	Owner adds entity to publisher list	Owner adds entity to owner list

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

	Outcast	None	Member	Publisher	Owner
Member	Owner bans entity	Owner removes entity from member list	--	Owner adds entity to publisher list	Owner adds entity to owner list
Publisher	Owner bans entity	Owner removes entity from publisher list	n/a	--	Owner adds entity to owner list
Owner	n/a	Owner resigns	n/a	n/a	--

4.2 Subscription States

Subscriptions to a node may exist in several states.

Subscription State	Description
None	The node MUST NOT send event notifications or payloads to the Entity.
Pending	An entity has requested to subscribe to a node and the request has not yet been approved by a node owner. The node MUST NOT send event notifications or payloads to the entity while it is in this state.
Unconfigured	An entity has subscribed but its subscription options have not yet been configured. The node MAY send event notifications or payloads to the entity while it is in this state. The service MAY timeout unconfigured subscriptions.
Subscribed	An entity is subscribed to a node. The node MUST send all event notifications (and, if configured, payloads) to the entity while it is in this state (subject to subscriber configuration and content filtering).

4.3 Event Types

The requirements for the publish-subscribe protocol imply that there are two major dimensions along which we can measure events: persistent vs. transient, and pure event notification vs. inclusion of payload. An implementation SHOULD enable an owner to configure a node along both of these dimensions.
No matter whether a node is configured for persistent or transient events, a service MAY cache the last item published to the node, in which case it SHOULD send that item to subscribers based on configuration of the "send_last_published_item" option (see the Item Caching section of this document); if the service supports the "http://jabber.org/protocol/pubsub#last-published" feature then the value of this option MUST default to "on_sub_and_presence"

(though the service SHOULD allow the node owner to override the default).
Note: The "on_sub_and_presence" setting relates to the *subscriber's* presence, not the publisher's presence.
A pubsub service MUST validate publish requests against the configuration of the node along both of these dimensions (see the Publish An Item to a Node section of this document for the relevant error conditions).
The node configuration and desired event type determine whether an item must be provided by the publisher, whether the item includes a payload in the publish request or event notification, and whether an item ID is provided by the publisher or generated by the pubsub service. We can summarize the relevant rules as follows:

	Notification-Only Node *	Payload-Included Node *
Persistent Node **	Publish request MUST include an <item/> element, which MAY be empty or MAY contain a payload; even if publish request contains a payload, pubsub service MUST NOT include the payload in event notifications; if publish request did not include item ID, pubsub service MUST generate item ID	Publish request MUST include an <item/> element, which SHOULD contain a payload; if publish request included a payload, event notifications MUST include the payload; if publish request did not include item ID, pubsub service MUST generate item ID
Transient Node **	Publish request MUST NOT include an <item/> element; payload is not included in publish request or event notifications, although event notifications MUST include an empty <items/> element; item ID is neither provided in publish request nor generated by pubsub service	Publish request MUST include an <item/> element, which SHOULD contain a payload; if publish request included a payload, event notifications MUST include the payload; pubsub service MAY generate an item ID

* Note: Whether the node is notification-only or includes payloads is determined by the "pubsub#deliver_payloads" configuration field.
* Note: Whether the node is persistent or transient is determined by the "pubsub#persist_items" configuration field.

4.4 Node Types

There are two types of nodes:

Node Type	Description
Leaf	A node that contains published items only. It is NOT a container for other nodes. This is the most common node type.
Collection	A node that contains nodes and/or other collections but no published items. Collections make it possible to represent more sophisticated relationships among nodes. For details, refer to XEP-0248.

4.5 Node Access Models

In order to make node creation simpler for clients, we define the following node access models (in order of openness):

Access Model	Description
Open	Any entity may subscribe to the node (i.e., without the necessity for subscription approval) and any entity may retrieve items from the node (i.e., without being subscribed); this SHOULD be the default access model for generic pubsub services.
Presence	Any entity with a subscription of type "from" or "both" may subscribe to the node and retrieve items from the node; this access model applies mainly to instant messaging systems (see RFC 3921).
Roster	Any entity in the specified roster group(s) may subscribe to the node and retrieve items from the node; this access model applies mainly to instant messaging systems (see RFC 3921).
Authorize	The node owner must approve all subscription requests, and only subscribers may retrieve items from the node.
Whitelist	An entity may subscribe or retrieve items only if on a whitelist managed by the node owner. The node owner MUST automatically be on the whitelist. In order to add entities to the whitelist, the node owner SHOULD use the protocol specified in the Manage Affiliated Entities section of this document, specifically by setting the affiliation to "member".

A generic publish-subscribe implementation SHOULD support all of the defined access models, although specialized publish-subscribe implementations MAY support only a subset of the access models. Which access models are provided in a particular deployment is a matter of service provisioning (e.g., some restricted deployments may wish to lock down permissions so that only the "authorize" and "whitelist" access models are provided, or even only the "whitelist" access model).
A node creator or owner can override the default access model by specifying an appropriate value for the 'pubsub#access_model' configuration field (see the Create a Node With Default Configuration and Configure a Node sections of this document).

4.6 Addressing

If a pubsub node is addressable, it MUST be addressable either (1) as a JID or (2) as the combination of a JID and a node. [3]

4.6.1 JID

If a pubsub node is addressable as a JID, the NodeID MUST be the resource identifier, and MUST NOT be specified by the "user" portion (node identifier) of the JID (e.g. "domain.tld/NodeID" and "user@domain.tld/NodeID" are allowed; "NodeID@domain.tld" is not allowed [4]). JID addressing SHOULD be used when interacting with a pubsub node using a protocol that does not support the node attribute. For example, when a service makes it possible for entities to subscribe to nodes via presence, it would address nodes as JIDs. If a pubsub node is addressable as a JID, the pubsub service MUST ensure that the NodeID conforms to the Resourceprep profile of Stringprep as described in RFC 3920.
Consider the following example, in which the pubsub service is located at the hostname pubsub.shakespeare.lit.

Listing 4: Node addressed as domain.tld/NodeID

```
<iq to='pubsub.shakespeare.lit/news announcements'>
  ...
</iq>
```

Now consider the following example, in which the pubsub service is located at pubsub@shakespeare.lit.

Listing 5: Node addressed as user@domain.tld/NodeID

```
<iq to='pubsub@shakespeare.lit/news announcements'>
  ...
</iq>
```

4.6.2 JID+NodeID

If a pubsub node is addressable as a JID plus node, the NodeID MUST be the value of both the Service Discovery 'node' attribute and the pubsub 'node' attribute; i.e., for discovery purposes, a pubsub node is equivalent to a Service Discovery node. If a pubsub node is addressable as a JID plus node, the pubsub service SHOULD ensure that the NodeID conforms to the Resourceprep profile of Stringprep as described in RFC 3920.

[3] These nodes are equivalent to those used in XEP-0030: Service Discovery.
[4] This rule does not apply to the root collection node, if any.

Consider the following example, in which the (virtual) pubsub service is located at hamlet@denmark.lit.

Listing 6: Node addressed as JID+NodeID

```
<iq to='hamlet@denmark.lit'>
  <query node='princely_musings'/>
</iq>
```

5 Entity Use Cases

This section defines the use cases for and protocols to be used by any entity that wishes to interact with a publish-subscribe service, mainly focused on Service Discovery use cases.

5.1 Discover Features

A service MUST respond to service discovery information requests qualified by the 'http://jabber.org/protocol/disco#info' namespace. The "disco#info" result returned by a pubsub service MUST indicate the identity of the service and indicate which pubsub features are supported.

Listing 7: Entity Queries Pubsub Service Regarding Supported Features

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='feature1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 8: Pubsub Service Returns Set of Supported Features

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='feature1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='pubsub' type='service'/>
    <feature var='http://jabber.org/protocol/pubsub'/>
  </query>
</iq>
```

The possible pubsub features are noted throughout this document and have been registered as described in the XMPP Registrar Considerations section of this document. For information regarding which features are required, recommended, and optional, see the Feature Summary

section of this document.

5.2 Discover Nodes

If a service implements a hierarchy of nodes (by means of Collection Nodes), it MUST also enable entities to discover the nodes in that hierarchy by means of the **Service Discovery** protocol, subject to the recommendations in XEP-0030 regarding large result sets (for which Jabber Search (XEP-0055) [5] or some other protocol SHOULD be used). The following examples show the use of service discovery in discovering the nodes available at a hierarchical pubsub service.
Note: Node hierarchies and collection nodes are OPTIONAL. For details, refer to the NodeID Semantics and Collection Nodes sections of this document.
In the first example, an entity sends a service discovery items ("disco#items") request to the root node (i.e., the service itself), which is a Collection Node:

Listing 9: Entity asks service for all first-level nodes

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='nodes1'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

Listing 10: Service returns all first-level nodes

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='nodes1'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='pubsub.shakespeare.lit'
          node='blogs'
          name='Weblog␣updates'/>
    <item jid='pubsub.shakespeare.lit'
          node='news'
          name='News␣and␣announcements'/>
  </query>
</iq>
```

In the second example, an entity sends a disco#items request to one of the first-level nodes, which is also a collection node:

Listing 11: Entity requests second-level nodes

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

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='nodes2'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='blogs'/>
</iq>
```

Listing 12: Service returns second-level nodes

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='nodes2'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='blogs'>
    <item jid='pubsub.shakespeare.lit'
          node='princely_musings'/>
    <item jid='pubsub.shakespeare.lit'
          node='kingly_ravings'/>
    <item jid='pubsub.shakespeare.lit'
          node='starcrossed_stories'/>
    <item jid='pubsub.shakespeare.lit'
          node='moorish_meanderings'/>
  </query>
</iq>
```

If a node is a leaf node rather than a collection node and items have been published to the node, the service MAY return one <item/> element for each published item as described in the Discover Items for a Node section of this document, however such items MUST NOT include a 'node' attribute (since they are published items, not nodes).

5.3 Discover Node Information

A pubsub service MUST allow entities to query individual nodes for the information associated with that node. The Service Discovery protocol MUST be used to discover this information. The "disco#info" result MUST include an identity with a category of "pubsub" and a type of either "leaf" or "collection".
Note: If a node has an identity type of "leaf", then it MUST NOT contain other nodes or collections (only items); if a node has an identity type of "collection", then it MUST NOT contain items (only other nodes or collections).

Listing 13: Entity queries collection node for information

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
```

```
    to='pubsub.shakespeare.lit'
    id='info2'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='blogs'/>
</iq>
```

Listing 14: Service responds with identity of pubsub/collection

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='meta1'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='blogs'>
    <identity category='pubsub' type='collection'/>
  </query>
</iq>
```

Listing 15: Entity queries leaf node for information

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='princely_musings'/>
</iq>
```

Listing 16: Service responds with identity of pubsub/leaf

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='princely_musings'>
    ...
    <identity category='pubsub' type='leaf'/>
    ...
  </query>
</iq>
```

5.4 Discover Node Metadata

The "disco#info" result MAY include detailed metadata about the node, encapsulated in the Data Forms (XEP-0004) [6] format as described in Service Discovery Extensions

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

(XEP-0128) [7], where the data form context is specified by including a FORM_TYPE of "http://jabber.org/protocol/pubsub#meta-data" in accordance with Field Standardization for Data Forms (XEP-0068) [8]. If metadata is provided, it SHOULD include values for all configured options as well as "automatic" information such as the node creation date, a list of publishers, and the like.

Listing 17: Entity queries a node for information

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='meta1'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='princely_musings'/>
</iq>
```

Listing 18: Service responds with information and metadata

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='meta1'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='princely_musings'>
    <identity category='pubsub' type='leaf'/>
    <feature var='http://jabber.org/protocol/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#type' label='Payload␣semantic␣type␣
          information' type='text-single'>
        <value>urn:example:e2ee:bundle</value>
      </field>
      <field var='pubsub#creator' label='Node␣creator' type='jid-
          single'>
        <value>hamlet@denmark.lit</value>
      </field>
      <field var='pubsub#creation_date' label='Creation␣date' type='
          text-single'>
        <value>2003-07-29T22:56:10Z</value>
      </field>
      <field var='pubsub#title' label='A␣short␣name␣for␣the␣node' type
          ='text-single'>
        <value>Princely Musings (Atom)</value>
      </field>
```

[7] XEP-0128: Service Discovery Extensions <https://xmpp.org/extensions/xep-0128.html>.
[8] XEP-0068: Field Data Standardization for Data Forms <https://xmpp.org/extensions/xep-0068.html>.

```
      <field var='pubsub#description' label='A description of the node
          ' type='text-single'>
        <value>Updates for Hamlet&apos;s Princely Musings weblog.</
            value>
      </field>
      <field var='pubsub#language' label='Default language' type='list
          -single'>
        <value>en</value>
      </field>
      <field var='pubsub#contact' label='People to contact with
          questions' type='jid-multi'>
        <value>bard@shakespeare.lit</value>
      </field>
      <field var='pubsub#access_model' label='Access model' type='list
          -single'>
        <value>open</value>
      </field>
      <field var='pubsub#publish_model' label='Publish model' type='
          list-single'>
        <value>publishers</value>
      </field>
      <field var='pubsub#max_items' label='Max # of items to persist.
          `max` for no specific limit other than a server imposed
          maximum.' type='text-single'>
        <value>120</value>
      </field>
      <field var='pubsub#owner' label='Node owners' type='jid-multi'>
        <value>hamlet@denmark.lit</value>
      </field>
      <field var='pubsub#publisher' label='Publishers to this node'
          type='jid-multi'>
        <value>hamlet@denmark.lit</value>
      </field>
      <field var='pubsub#num_subscribers' label='Number of subscribers
          to this node' type='text-single'>
        <value>1066</value>
      </field>
    </x>
  </query>
</iq>
```

Note: Node metadata can be set in many ways. Some of it is based on node configuration (e.g., the owner's JID) whereas some of it may be dynamic (e.g., the number of subscribers). Any static information to be provided in the node metadata SHOULD be provided as fields in the node configuration form.

Note: The pubsub#language field SHOULD be list-single so that the pubsub service can present an appropriate list of languages and language codes.

5.5 Discover Items for a Node

To discover the published items which exist on the service for a specific node, an entity MAY send a "disco#items" request to the node itself, and the service MAY return each item as a Service Discovery <item/> element. The 'name' attribute of each Service Discovery item MUST contain its ItemID and the item MUST NOT possess a 'node' attribute. This ItemID MAY then be used to retrieve the item using the protocol defined in the Retrieve Items from a Node section of this document.

Listing 19: Entity requests all of the items for a node

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='items1'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='princely_musings'/>
</iq>

<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='princely_musings'>
    <item jid='pubsub.shakespeare.lit' name='368866411
       b877c30064a5f62b917cffe'/>
    <item jid='pubsub.shakespeare.lit' name='3300659945416
       e274474e469a1f0154c'/>
    <item jid='pubsub.shakespeare.lit' name='4
       e30f35051b7b8b42abe083742187228'/>
    <item jid='pubsub.shakespeare.lit' name='
       ae890ac52d0df67ed7cfdf51b644e901'/>
  </query>
</iq>
```

5.6 Retrieve Subscriptions

An entity may want to query the service to retrieve its subscriptions for all nodes at the service. Support for this feature ("retrieve-subscriptions") is RECOMMENDED.
In order to make the request, the requesting entity MUST send an IQ-get whose <pubsub/> child contains an empty <subscriptions/> element with no attributes.

Listing 20: Entity requests all current subscriptions

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
```

```
    to='pubsub.shakespeare.lit'
    id='subscriptions1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscriptions/>
  </pubsub>
</iq>
```

If the service returns a list of subscriptions, it MUST return all subscriptions for all JIDs that match the bare JID (<localpart@domain.tld> or <domain.tld>) portion of the 'from' attribute on the request.
For each subscription, a <subscription/> element is returned specifying the NodeID, the JID that is affiliated (which MAY include a resource, depending on how the entity subscribed), and the current subscription state. If subscription identifiers are supported by the service, the 'subid' attribute MUST be present as well.

Listing 21: Service returns all current subscriptions

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit'
    id='subscriptions1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscriptions>
      <subscription node='node1' jid='francisco@denmark.lit'
          subscription='subscribed'/>
      <subscription node='node2' jid='francisco@denmark.lit'
          subscription='subscribed'/>
      <subscription node='node5' jid='francisco@denmark.lit'
          subscription='unconfigured'/>
      <subscription node='node6' jid='francisco@denmark.lit'
          subscription='subscribed' subid='123-abc'/>
      <subscription node='node6' jid='francisco@denmark.lit'
          subscription='subscribed' subid='004-yyy'/>
    </subscriptions>
  </pubsub>
</iq>
```

If the requesting entity has no subscriptions, the pubsub service MUST return an empty <subscriptions/> element.

Listing 22: No subscriptions

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='subscriptions1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscriptions/>
```

```
  </pubsub>
</iq>
```

If the service does not support subscriptions retrieval, the service MUST respond with a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "retrieve-subscriptions".

Listing 23: Subscriptions retrieval not supported

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='subscriptions1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='retrieve-subscriptions'/>
  </error>
</iq>
```

An entity MAY also request all of its subscriptions at a specific node (e.g., if it has subscriptions with multiple SubIDs) by including a 'node' attribute on the <subscriptions/> element.

Listing 24: Entity requests current subscriptions from a specific node

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='subscriptions2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscriptions node='princely_musings'/>
  </pubsub>
</iq>
```

The service would then return only the entity's subscriptions to that specific node; this acts as a filter on the entity's subscriptions.

Listing 25: Service returns all current subscriptions to a specific node

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit'
    id='subscriptions2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscriptions node='node6'>
      <subscription node='node6' jid='francisco@denmark.lit'
          subscription='subscribed' subid='123-abc'/>
```

```
      <subscription node='node6' jid='francisco@denmark.lit'
          subscription='subscribed' subid='004-yyy'/>
    </subscriptions>
  </pubsub>
</iq>
```

5.7 Retrieve Affiliations

An entity may want to query the service to retrieve its affiliations for all nodes at the service, or query a specific node for its affiliation with that node. Support for this feature ("retrieve-affiliations") is RECOMMENDED.
In order to make the request of the service, the requesting entity includes an empty <affiliations/> element with no attributes.

Listing 26: Entity requests all current affiliations

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='affil1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <affiliations/>
  </pubsub>
</iq>
```

If the service returns a list of affiliations, it MUST return all affiliations for all JIDs that match the bare JID (<localpart@domain.tld> or <domain.tld>) portion of the 'from' attribute on the request.
For each affiliation, an <affiliation/> element is returned containing the NodeID and the affiliation state (owner, publisher, publish-only, member, or outcast).

Listing 27: Service replies with all current affiliations

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit'
    id='affil1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <affiliations>
      <affiliation node='node1' affiliation='owner'/>
      <affiliation node='node2' affiliation='publisher'/>
      <affiliation node='node5' affiliation='outcast'/>
      <affiliation node='node6' affiliation='owner'/>
    </affiliations>
  </pubsub>
</iq>
```

If the requesting entity has no affiliations, the pubsub service MUST return an empty <affiliations/> element.

Listing 28: No affiliations

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='affil1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <affiliations/>
  </pubsub>
</iq>
```

If the service does not support affiliations retrieval, the service MUST respond with a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "retrieve-affiliations".

Listing 29: Affiliations retrieval not supported

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='affil1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='retrieve-affiliations'/>
  </error>
</iq>
```

In order to make an affiliations request of a specific node, the requesting entity includes an empty <affiliations/> element with a 'node' attribute.

Listing 30: Entity requests affiliation at a specific node

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='affil2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <affiliations node='node6'/>
  </pubsub>
</iq>
```

Listing 31: Service replies with current affiliation

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit'
    id='affil2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <affiliations>
      <affiliation node='node6' affiliation='owner'/>
    </affiliations>
  </pubsub>
</iq>
```

6 Subscriber Use Cases

This section defines the use cases for and protocols to be used by potential and actual subscribers. (Note: The Implementation Notes section of this document describes many important factors and business rules which a pubsub service MUST observe. In addition, the examples throughout assume the existence of a separate pubsub component and include any relevant 'from' addresses as stamped by a server or network edge.)

6.1 Subscribe to a Node

6.1.1 Request

When an XMPP entity wishes to subscribe to a node, it sends a subscription request to the pubsub service. The subscription request is an IQ-set where the <pubsub/> element contains one and only one <subscribe/> element. The <subscribe/> element SHOULD possess a 'node' attribute specifying the node to which the entity wishes to subscribe. The <subscribe/> element MUST also possess a 'jid' attribute specifying the exact XMPP address to be used as the subscribed JID -- often a bare JID (<localpart@domain.tld> or <domain.tld>) or full JID (<localpart@domain.tld/resource> or <domain.tld/resource>).
Here is an example of a subscription request.

Listing 32: Entity subscribes to a node

```
<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe
        node='princely_musings'
        jid='francisco@denmark.lit'/>
  </pubsub>
</iq>
```

6.1.2 Success Case

If the subscription request is successfully processed, the server MUST inform the requesting entity that it is now subscribed (which MAY include a service-generated SubID).

Listing 33: Service replies with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='princely_musings'
        jid='francisco@denmark.lit'
        subid='ba49252aaa4f5d320c24d3766f0bdcade78c78d3'
        subscription='subscribed'/>
  </pubsub>
</iq>
```

6.1.3 Error Cases

There are several reasons why the subscription request might fail:

1. The bare JID portions of the JIDs do not match.
2. The node has an access model of "presence" and the requesting entity is not subscribed to the owner's presence.
3. The node has an access model of "roster" and the requesting entity is not in one of the authorized roster groups.
4. The node has an access model of "whitelist" and the requesting entity is not on the whitelist.
5. The service requires payment for subscriptions to the node.
6. The requesting entity is anonymous and the service does not allow anonymous entities to subscribe.
7. The requesting entity has a pending subscription.
8. The requesting entity is blocked from subscribing (e.g., because having an affiliation of outcast).
9. The requesting entity has attempted to establish too many subscriptions.
10. The node does not support subscriptions.

11. The node has moved.
12. The node does not exist.

These error cases are described more fully in the following sections.
If the specified JID is a bare JID or full JID, the service MUST at a minimum check the bare JID portion against the bare JID portion of the 'from' attribute on the received IQ request to make sure that the requesting entity has the same identity as the JID which is being requested to be added to the subscriber list.
If the bare JID portions of the JIDs do not match as described above and the requesting entity does not have some kind of admin or proxy privilege as defined by the implementation, the service MUST return a <bad-request/> error, which SHOULD also include a pubsub-specific error condition of <invalid-jid/>.

Listing 34: JIDs do not match

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <invalid-jid xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

Note: An implementation MAY enable the service administrator to configure a list of entities that are excluded from this check; those entities may be considered "trusted proxies" that are allowed to subscribe on behalf of other entities. In the same way, implementations MAY enable listing of entities that are not allowed to perform specific operations (such as subscribing or creating nodes).
For nodes with an access model of "presence", if the requesting entity is not subscribed to the owner's presence then the pubsub service MUST respond with a <not-authorized/> error, which SHOULD also include a pubsub-specific error condition of <presence-subscription-required/>.

Listing 35: Entity is not authorized to create a subscription (presence subscription required)

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='auth'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <presence-subscription-required xmlns='http://jabber.org/protocol/
        pubsub#errors'/>
  </error>
</iq>
```

For nodes with an access model of ”roster”, if the requesting entity is not in one of the authorized roster groups then the pubsub service MUST respond with a <not-authorized/> error, which SHOULD also include a pubsub-specific error condition of <not-in-roster-group/>.

Listing 36: Entity is not authorized to create a subscription (not in roster group)

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='auth'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <not-in-roster-group xmlns='http://jabber.org/protocol/pubsub#
       errors'/>
  </error>
</iq>
```

For nodes with a node access model of ”whitelist”, if the requesting entity is not on the whitelist then the service MUST return a <not-allowed/> error, specifying a pubsub-specific error condition of <closed-node/>.

Listing 37: Node has whitelist access model

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <closed-node xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

Commercial deployments may wish to link subscribers to a database of paying customers. If the subscriber needs to provide payment in order to subscribe to the node (e.g., if the subscriber is not in the customer database or the customer’s account is not paid up), the service SHOULD return a <payment-required/> error to the subscriber.

Listing 38: Payment is required for a subscription

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='auth'>
    <payment-required xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Some XMPP servers may allow authentication using SASL ANONYMOUS; however, because the resulting entity is unstable (the assigned JID may not be owned by the same principal in a persistent manner), a service MAY prevent anonymous entities from subscribing to nodes and SHOULD use service discovery to determine if an entity has an identity of "account/anonymous". If a requesting entity is anonymous but the service does not allow anonymous entities to subscribe, the service SHOULD return a <forbidden/> error to the subscriber.

Listing 39: Requesting entity is anonymous

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='anonymous@denmark.lit/foo'
    id='sub1'>
  <error type='cancel'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requesting entity has a pending subscription, the service MUST return a <not-authorized/> error to the subscriber, specifying a pubsub-specific error condition of <pending-subscription/>.

Listing 40: Requesting entity has pending subscription

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='auth'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <pending-subscription xmlns='http://jabber.org/protocol/pubsub#
        errors'/>
  </error>
</iq>
```

If the requesting entity is blocked from subscribing (e.g., because having an affiliation of outcast), the service MUST return a <forbidden/> error to the subscriber.

Listing 41: Requesting entity is blocked

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requesting entity has attempted to establish too many subscriptions (where the definition of "too many" is a matter of local service policy), the service MUST return a <policy-violation/> error to the subscriber, specifying a pubsub-specific error condition of <too-many-subscriptions/>.

Listing 42: Requesting entity has exceeded limit on number of subscriptions

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='wait'>
    <policy-violation xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <too-many-subscriptions xmlns='http://jabber.org/protocol/pubsub#
        errors'/>
  </error>
</iq>
```

The service can match on bare JID or full JID in determining which subscribing entities match for the purpose of determining if an entity has requested too many subscriptions.
If the node does not allow entities to subscribe, the service SHOULD return a <feature-not-implemented/> error to the subscriber, specifying a pubsub-specific error condition of <unsupported/> and a feature of "subscribe".

Listing 43: Subscribing not supported

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='subscribe'/>
  </error>
</iq>
```

If the node has, the service SHOULD return a <gone/> error (if the node has moved permanently) or a <redirect/> error (if the node has moved temporarily).

Listing 44: Node has moved

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='modify'>
```

```
    <gone xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
      xmpp:pubsub.shakespeare.lit?;node=some-other-node
    </gone>
  </error>
</iq>
```

If the node does not exist, the service SHOULD return an <item-not-found/> error to the subscriber.

Listing 45: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

6.1.4 Approval Required

For nodes with an access model of "authorize", subscription requests MUST be approved by one of the node owners unless service policy allows entities with affiliations other than "none" to auto-subscribe (e.g., members and publishers might be allowed to auto-subscribe); therefore the pubsub service sends a message to the node owner(s) requesting authorization (see the Manage Subscription Requests section of this document). Because the subscription request may or may not be approved, the service MUST return a pending notification to the subscriber.

Listing 46: Service replies with pending

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='princely_musings'
        jid='francisco@denmark.lit'
        subscription='pending'/>
  </pubsub>
</iq>
```

6.1.5 Configuration Required

If the entity must configure its subscription options (see the Configure Subscription Options section of this document) before receiving event notifications, the service MUST so inform

the entity. It SHOULD do so by returning an IQ-result to the requesting entity with a notation that configuration of subscription options is required.

Listing 47: Service replies with success and indicates that subscription configuration is required

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='princely_musings'
        jid='francisco@denmark.lit'
        subscription='unconfigured'>
      <subscribe-options>
        <required/>
      </subscribe-options>
    </subscription>
  </pubsub>
</iq>
```

Note: The node shall include the <required/> child element only if the subscriber must configure the subscription before receiving any event notifications. A service MAY time out subscription requests if configuration is required and a configuration request is not submitted within a reasonable amount of time (which shall be determined by the service or node configuration).
Alternatively, if the service is unable to create the subscription without simultaneous configuration, the service MAY return a <not-acceptable/> error, specifying a pubsub-specific error condition of <configuration-required/>.

Listing 48: Service returns error specifying that subscription configuration is required

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe
        node='princely_musings'
        jid='francisco@denmark.lit'/>
    <options node='princely_musings' jid='francisco@denmark.lit'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        <field var='pubsub#deliver'><value>1</value></field>
```

```
        <field var='pubsub#digest'><value>0</value></field>
        <field var='pubsub#include_body'><value>false</value></field>
        <field var='pubsub#show-values'>
          <value>chat</value>
          <value>online</value>
          <value>away</value>
        </field>
      </x>
    </options>
  </pubsub>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <configuration-required xmlns='http://jabber.org/protocol/pubsub#
        errors'/>
  </error>
</iq>
```

If the <required/> element is not included and no error is returned, the subscription takes effect immediately and the entity may configure the subscription at any time (the service MAY indicate that subscription options are supported by including an empty <subscribe-options/> element in the IQ-result, as shown in the following example).

Listing 49: Service replies with success and indicates that subscription options are supported but not required

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='princely_musings'
        jid='francisco@denmark.lit'
        subscription='unconfigured'>
      <subscribe-options/>
    </subscription>
  </pubsub>
</iq>
```

6.1.6 Multiple Subscriptions

An entity may wish to subscribe using different subscription options, which it can do by subscribing multiple times to the same node. Support for this feature ("multi-subscribe") is OPTIONAL.

If multiple subscriptions for the same JID are allowed, the service MUST use the 'subid' attribute to differentiate between subscriptions for the same entity (therefore the SubID MUST be unique for each node+JID combination and the SubID MUST be present on the

<subscription/> element any time it is sent to the subscriber). It is NOT RECOMMENDED for clients to generate SubIDs, since collisions might result; therefore a service SHOULD generate the SubID on behalf of the subscriber and MAY overwrite SubIDs if they are provided by subscribers. If the service does not allow multiple subscriptions for the same entity and it receives an additional subscription request, the service MUST return the current subscription state (as if the subscription was just approved).
When the pubsub service generates event notifications, it SHOULD send only one event notification to an entity that has multiple subscriptions, rather than one event notification for each subscription. By "entity" here is meant the JID specified for the subscription, whether bare JID or full JID; however, if the same bare JID has multiple subscriptions but those subscriptions are for different full JIDs (e.g., one subscription for user@domain.tld./foo and another subscription for user@domain.tld/bar), the service MUST treat those as separate JIDs for the purpose of generating event notifications.

6.1.7 Receiving the Last Published Item

When a subscription request is successfully processed, the service MAY send the last published item to the new subscriber. The message containing this item SHOULD be stamped with extended information qualified by the 'urn:xmpp:delay' namespace (see Delayed Delivery (XEP-0203) [9]) to indicate it is sent with delayed delivery. (Note that in this example the event notification is sent to the bare JID since that is the subscribed JID.)

Listing 50: Service sends last published item

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
Whether 'tis␣nobler␣in␣the␣mind␣to␣suffer
The␣slings␣and␣arrows␣of␣outrageous␣fortune,
Or␣to␣take␣arms␣against␣a␣sea␣of␣troubles,
And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/atom03'/>
␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
␣␣␣␣␣␣␣␣␣␣</entry>
␣␣␣␣␣␣␣␣</item>
```

[9] XEP-0203: Delayed Delivery <https://xmpp.org/extensions/xep-0203.html>.

```
    </items>
  </event>
  <delay xmlns='urn:xmpp:delay' stamp='2003-12-13T23:58:37Z'/>
</message>
```

If the service sends the last published item by default for all nodes (subject to overriding by node configuration), it MUST return a feature of "http://jabber.org/protocol/pubsub#last-published" in its responsess to disco#info requests.

6.2 Unsubscribe from a Node

6.2.1 Request

To unsubscribe from a node, the subscriber sends an IQ-set whose <pubsub/> child contains an <unsubscribe/> element that specifies the node and the subscribed JID.

Listing 51: Entity unsubscribes from a node

```
<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='unsub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <unsubscribe
        node='princely_musings'
        jid='francisco@denmark.lit'/>
  </pubsub>
</iq>
```

6.2.2 Success Case

If the request can be successfully processed, the service MUST return an IQ result and MAY include a <pubsub/> child element with the updated <subscription/> element for that node. If subscription identifiers are supported by the service, the 'subid' attribute MUST be present as well.

Listing 52: Service replies with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='unsub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='princely_musings'
        jid='francisco@denmark.lit'
```

```
        subscription='none'
        subid='ba49252aaa4f5d320c24d3766f0bdcade78c78d3'/>
  </pubsub>
</iq>
```

6.2.3 Error Cases

There are several reasons why the unsubscribe request might fail:

1. The requesting entity has multiple subscriptions to the node but does not specify a subscription ID.
2. The request does not specify an existing subscriber.
3. The requesting entity does not have sufficient privileges to unsubscribe the specified JID.
4. The node does not exist.
5. The request specifies a subscription ID that is not valid or current.

These error cases are described more fully in the following sections.
If the requesting entity has multiple subscriptions to the node but does not specify a subscription ID, the service MUST return a <bad-request/> error, which SHOULD also include a pubsub-specific error condition of <subid-required/>.

Listing 53: Entity did not specify SubID

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='unsub1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <subid-required xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

If the value of the 'jid' attribute does not specify an existing subscriber, the pubsub service MUST return an error stanza, which SHOULD be <unexpected-request/> and which SHOULD also include a pubsub-specific error condition of <not-subscribed/>.

Listing 54: Requesting entity is not a subscriber

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
```

```
    id='unsub1'>
  <error type='cancel'>
    <unexpected-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <not-subscribed xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

If the requesting entity is prohibited from unsubscribing the specified JID, the service MUST return a <forbidden/> error. The service MUST validate that the entity making the request is authorized to unsubscribe the entity. If the subscriber's JID is of the form (<localpart@domain.tld/resource> or <domain.tld/resource>), a service MUST perform this check by comparing the (<localpart@domain.tld> or <domain.tld>) part of the two JIDs to ensure that they match. If the bare JID portions of the JIDs do not match and the requesting entity is not authorized to unsubscribe the JID (e.g., because it is not a service-wide admin or authorized proxy), the service MUST return a <forbidden/> error.

Listing 55: Requesting entity is prohibited from unsubscribing entity

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='unsub1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node does not exist, the pubsub service MUST return an <item-not-found/> error.

Listing 56: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='unsub1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If a subscription identifier is associated with the subscription, the unsubscribe request MUST include an appropriate 'subid' attribute. If the unsubscribe request includes a SubID but SubIDs are not supported for the node (or the subscriber did not subscribe using a SubID in the first place), the service SHOULD ignore the SubID and simply unsubscribe the entity. If the subscriber originally subscribed with a SubID but the unsubscribe request includes a SubID that is not valid or current for the subscriber, the service MUST return a <not-acceptable/> error, which SHOULD also include a pubsub-specific error condition of <invalid-subid/>.

Listing 57: Invalid subscription identifier

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='unsub1'>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <invalid-subid xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

6.3 Configure Subscription Options

An implementation MAY allow subscribers to configure subscription options. Implementations SHOULD use the Data Forms protocol to accomplish this configuration (however, an out-of-band mechanism such as a web interface could be offered as well).

6.3.1 Advertising Support

If a service supports subscription options it MUST advertise that fact in its response to a "disco#info" query by including a feature whose 'var' attribute is "pubsub#subscription-options".

Listing 58: Pubsub service indicates support for subscription options

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='feature1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='http://jabber.org/protocol/pubsub#subscription-
        options'/>
    ...
  </query>
</iq>
```

6.3.2 Request

A subscriber requests the subscription options by including an <options/> element inside an IQ-get stanza.

Listing 59: Subscriber requests subscription options form

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='options1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'/>
  </pubsub>
</iq>
```

6.3.3 Success Case

If the request can be successfully processed, the service MUST respond with the options.

Listing 60: Service responds with the options form

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='options1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'>
      <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
             value>
        </field>
        <field var='pubsub#deliver' type='boolean'
               label='Enable␣delivery?'>
          <value>1</value>
        </field>
        <field var='pubsub#digest' type='boolean'
               label='Receive␣digest␣notifications␣(approx.␣one␣per␣
                  day)?'>
          <value>0</value>
        </field>
        <field var='pubsub#include_body' type='boolean'
               label='Receive␣message␣body␣in␣addition␣to␣payload?'>
          <value>false</value>
        </field>
        <field
            var='pubsub#show-values'
            type='list-multi'
            label='Select␣the␣presence␣types␣which␣are
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣allowed␣to␣receive␣event␣notifications'>
          <option label='Want␣to␣Chat'><value>chat</value></option>
          <option label='Available'><value>online</value></option>
          <option label='Away'><value>away</value></option>
          <option label='Extended␣Away'><value>xa</value></option>
```

```
          <option label='Do_Not_Disturb'><value>dnd</value></option>
          <value>chat</value>
          <value>online</value>
        </field>
      </x>
    </options>
  </pubsub>
</iq>
```

Note: The foregoing example shows some (but by no means all) of the possible configuration options that MAY be provided. If an implementation provides these options using the **Data Forms** protocol, it MUST use the field variables that are registered with the XMPP Registrar in association with the 'http://jabber.org/protocol/pubsub' namespace (a preliminary representation of those field variables is shown above and in the pubsub#subscribe_options FORM_TYPE section of this document, but MUST NOT be construed as canonical since the XMPP Registrar may standardize additional fields at a later date without changes to this document).
Note: Many of the relevant data form fields are of type "boolean" and MUST be handled accordingly. [10]

6.3.4 Error Cases

There are several reasons why the options request might fail:

1. The requesting entity does not have sufficient privileges to modify subscription options for the specified JID.
2. The requesting entity (or specified subscriber) is not subscribed.
3. The request does not specify both the NodeID and the subscriber's JID.
4. The request does not specify a subscription ID but one is required.
5. The request specifies a subscription ID that is not valid or current.
6. Subscription options are not supported.
7. The node does not exist.

These error cases are described more fully in the following sections.
When requesting subscription options, the subscriber MUST specify the JID that is subscribed to the node and SHOULD specify a node (if no node is specified, i.e. via node="", the service MUST assume that the requesting entity wishes to request subscription options for its

[10] In accordance with Section 3.2.2.1 of XML Schema Part 2: Datatypes, the allowable lexical representations for the xs:boolean datatype are the strings "0" and "false" for the concept 'false' and the strings "1" and "true" for the concept 'true'; implementations MUST support both styles of lexical representation.

subscription to the root collection node; refer to XEP-0248 for details).
The service MUST validate that the entity making the request is authorized to set the subscription options for the subscribed entity. If the subscriber's JID is of the form (<localpart@domain.tld/resource> or <domain.tld/resource>), a service MUST perform this check by comparing the (<localpart@domain.tld> or <domain.tld>) part of the two JIDs to ensure that they match. If the bare JID portions of the JIDs do not match and the requesting entity is not authorized to modify subscription options for the JID (e.g., because it is not a service-wide admin or authorized proxy), the service MUST return a <forbidden/> error.

Listing 61: Requesting entity does not have sufficient privileges to modify subscription options

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requesting entity (or specified subscriber, if different) is not subscribed, the service MUST return an <unexpected-request/> error, which SHOULD also include a pubsub-specific error condition of <not-subscribed/>.

Listing 62: No such subscriber

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='options1'>
  <error type='modify'>
    <unexpected-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <not-subscribed xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

If the subscriber does not specify a subscriber JID, the service MUST return a <bad-request/> error, which SHOULD also include a pubsub-specific error condition of <jid-required/>.

Listing 63: Subscriber JID not specified

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='options1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <jid-required xmlns='http://jabber.org/protocol/pubsub#errors'/>
```

```
  </error>
</iq>
```

If a subscription identifier is associated with the subscription, the 'subid' attribute MUST be present on the request in order for the service to differentiate subscriptions for the same entity. If the 'subid' is required but not provided, the service MUST return a <bad-request/> error, which SHOULD also include a pubsub-specific error condition of <subid-required/>.

Listing 64: SubID required

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <subid-required xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

If a subscription identifier is associated with the subscription but the request includes a SubID that is not valid or current for the subscriber, the service MUST return a <not-acceptable/> error, which SHOULD also include a pubsub-specific error condition of <invalid-subid/>.

Listing 65: Invalid subscription identifier

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='unsub1'>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <invalid-subid xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

If the node or service does not support subscription options, the service MUST respond with a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "subscription-options".

Listing 66: Subscription options not supported

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='options1'>
  <error type='cancel'>
```

```
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='subscription-options'/>
  </error>
</iq>
```

If the node does not exist, the pubsub service MUST return an <item-not-found/> error.

Listing 67: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='options1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

6.3.5 Form Submission

After receiving the configuration form, the requesting entity SHOULD submit the form in order to update the entity's subscription options for that node.

Listing 68: Subscriber submits completed options form

```
<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='options2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        <field var='pubsub#deliver'><value>1</value></field>
        <field var='pubsub#digest'><value>0</value></field>
        <field var='pubsub#include_body'><value>false</value></field>
        <field var='pubsub#show-values'>
          <value>chat</value>
          <value>online</value>
          <value>away</value>
        </field>
      </x>
    </options>
```

```
  </pubsub>
</iq>
```

6.3.6 Form Processing

If the service can successfully process the submission, it MUST respond with success.

Listing 69: Service responds with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='options2'/>
```

If the subscriber attempts to set an invalid group of options, the service MUST respond with a <bad-request/> error.

Listing 70: Service responds with Bad Request for invalid options

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='options2'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <invalid-options xmlns='http://jabber.org/protocol/pubsub#errors'/
        >
  </error>
</iq>
```

The other errors already mentioned for getting subscription options also apply to setting subscription options.

6.3.7 Subscribe and Configure

As noted, if a service supports subscription options, an entity MAY subscribe and provide the subscription options in the same stanza.
Note: The <options/> element MUST follow the <subscribe/> element and MUST NOT possess a 'node' attribute or 'jid' attribute, since the value of the <subscribe/> element's 'node' attribute specifies the desired NodeID and the value of the <subscribe/> element's 'jid' attribute specifies the subscriber's JID; if any of these rules are violated, the service MUST return a <bad-request/> error.

Listing 71: Entity subscribes to node and sets configuration options

```
<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe node='princely_musings' jid='francisco@denmark.lit'/>
    <options>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        <field var='pubsub#deliver'><value>1</value></field>
        <field var='pubsub#digest'><value>0</value></field>
        <field var='pubsub#include_body'><value>false</value></field>
        <field var='pubsub#show-values'>
          <value>chat</value>
          <value>online</value>
          <value>away</value>
        </field>
      </x>
    </options>
  </pubsub>
</iq>
```

When the service informs the client of success, it SHOULD include a data form of type "result" informing the client of the resulting configuration options.

Listing 72: Service replies with success (including configuration options)

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='princely_musings'
        jid='francisco@denmark.lit'
        subid='ba49252aaa4f5d320c24d3766f0bdcade78c78d3'
        subscription='subscribed'/>
    <options>
      <x xmlns='jabber:x:data' type='result'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        <field var='pubsub#deliver'><value>1</value></field>
        <field var='pubsub#digest'><value>0</value></field>
        <field var='pubsub#include_body'><value>false</value></field>
```

```
        <field var='pubsub#show-values'>
          <value>chat</value>
          <value>online</value>
          <value>away</value>
        </field>
      </x>
    </options>
  </pubsub>
</iq>
```

6.4 Request Default Subscription Configuration Options

An entity might want to request information about the default subscription configuration. Support for this feature is OPTIONAL.

6.4.1 Request

To get the default subscription options for a node, the entity MUST send an empty <default/> element to the node; in response, the node SHOULD return the default subscription options.

Listing 73: Entity requests default subscription configuration options

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='def1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <default node='princely_musings'/>
  </pubsub>
</iq>
```

Note: Here the namespace is 'http://jabber.org/protocol/pubsub' (not 'http://jabber.org/protocol/pubsub#owner' as for retrieval of the default node configuration options).
The service itself MAY also have default subscription configuration options. To get the default subscription configuration options all (leaf) nodes at a service, the entity MUST send an empty <default/> element but not specifiy a node; in response, the service SHOULD return the default subscription options.

Listing 74: Entity requests default subscription configuration options

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='def2'>
```

```
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <default/>
  </pubsub>
</iq>
```

The process for retrieving the default subscription configuration options for collection nodes is described in XEP-0248.

6.4.2 Success Case

If no error occurs, the node MUST return the default subscription configuration options.

Listing 75: Service responds with default subscription configuration options

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='def1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <default node='princely_musings'>
      <x xmlns='jabber:x:data' type='result'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        <field var='pubsub#deliver'><value>1</value></field>
        <field var='pubsub#digest'><value>0</value></field>
        <field var='pubsub#include_body'><value>false</value></field>
        <field var='pubsub#show-values'>
          <value>chat</value>
          <value>online</value>
          <value>away</value>
        </field>
      </x>
    </default>
  </pubsub>
</iq>
```

6.4.3 Error Cases

There are several reasons why the default subscription configuration options request might fail:

1. The service does not support subscription configuration.
2. The service does not support retrieval of default subscription configuration.

These error cases are described more fully in the following sections.
If the node does not support subscription configuration, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "subscription-options".

Listing 76: Service does not support subscription configuration

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='def1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='subscription-options'/>
  </error>
</iq>
```

If the node does not support retrieval of default subscription configuration options, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "retrieve-default-sub".

Listing 77: Service does not support retrieval of default subscription configuration options

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='def1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='retrieve-default-sub'/>
  </error>
</iq>
```

6.5 Retrieve Items from a Node

Implementations of pubsub that choose to persist items MAY allow entities to request existing items from a node (e.g., an entity may wish to do this after successfully subscribing in order to receive all the items in the publishing history for the node).

6.5.1 Permissions

The service MUST conform to the node's access model in determining whether to return items to the entity that requests them. Specifically:

- If the access model is "open", the service SHOULD allow any entity (whether or not it is subscribed) to retrieve items.
- If the access model is "presence", the service SHOULD allow any entity that is subscribed to the owner's presence to retrieve items.
- If the access model is "roster", the service SHOULD allow any entity that is subscribed to the owner's presence and contained in the relevant roster group(s) to retrieve items.
- If the access model is "authorize" or "whitelist", the service MUST allow only subscribed entities to retrieve items.

The only exception foreseen to the SHOULD requirements for the foregoing access models is the enforcement of local privacy and security policies as specified more fully in the Security Considerations section of this document. (In addition, a service MUST always allow the node owner to retrieve items from a node and SHOULD always allow a publisher to do so.)

6.5.2 Requesting All Items

The subscriber may request all items by specifying only the Node ID without restrictions.

Listing 78: Subscriber requests all items

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='items1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'/>
  </pubsub>
</iq>
```

6.5.3 Returning All Items

The service then SHOULD return all available items at the node, although it MAY truncate the result set if a large number of items has been published (see next section) and naturally it

cannot return items that have been deleted, expired, etc.

Listing 79: Service returns all items

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'>
      <item id='368866411b877c30064a5f62b917cffe'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>The Uses of This World</title>
          <summary>
O, that this too too solid flesh would melt
Thaw and resolve itself into a dew!
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-12T17:47:23Z</published>
          <updated>2003-12-12T17:47:23Z</updated>
        </entry>
      </item>
      <item id='3300659945416e274474e469a1f0154c'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Ghostly Encounters</title>
          <summary>
O all you host of heaven! O earth! what else?
And shall I couple hell? O, fie! Hold, hold, my heart;
And you, my sinews, grow not instant old,
But bear me stiffly up. Remember thee!
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-12T23:21:34Z</published>
          <updated>2003-12-12T23:21:34Z</updated>
        </entry>
      </item>
      <item id='4e30f35051b7b8b42abe083742187228'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Alone</title>
          <summary>
Now I am alone.
O, what a rogue and peasant slave am I!
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
```

```
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-13T11:09:53Z</published>
          <updated>2003-12-13T11:09:53Z</updated>
        </entry>
      </item>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
Whether 'tis␣nobler␣in␣the␣mind␣to␣suffer
The␣slings␣and␣arrows␣of␣outrageous␣fortune,
Or␣to␣take␣arms␣against␣a␣sea␣of␣troubles,
And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/atom03'/>
␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
␣␣␣␣␣␣␣␣</entry>
␣␣␣␣␣␣</item>
␣␣␣␣</items>
␣␣</pubsub>
</iq>
```

6.5.4 Returning Some Items

A node may have a large number of items associated with it, in which case it may be problematic to return all of the items in response to an items request. In this case, the service SHOULD return some of the items and note that the list of items has been truncated by including a Result Set Management (XEP-0059) [11] notation.
A Pubsub entity supporting Result Set Management (XEP-0059) [12] SHOULD include a feature of "http://jabber.org/protocol/pubsub#rsm" in its disco#info response, to make it clear that the RSM feature is for PubSub, and not for e.g., Message Archive Management (XEP-0313) [13].

Listing 80: Service returns some items via result set management

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'>
```

[11] XEP-0059: Result Set Management <https://xmpp.org/extensions/xep-0059.html>.
[12] XEP-0059: Result Set Management <https://xmpp.org/extensions/xep-0059.html>.
[13] XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.

```
      <item id='368866411b877c30064a5f62b917cffe'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>The Uses of This World</title>
          <summary>
O, that this too too solid flesh would melt
Thaw and resolve itself into a dew!
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-12T17:47:23Z</published>
          <updated>2003-12-12T17:47:23Z</updated>
        </entry>
      </item>
      <item id='3300659945416e274474e469a1f0154c'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Ghostly Encounters</title>
          <summary>
O all you host of heaven! O earth! what else?
And shall I couple hell? O, fie! Hold, hold, my heart;
And you, my sinews, grow not instant old,
But bear me stiffly up. Remember thee!
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-12T23:21:34Z</published>
          <updated>2003-12-12T23:21:34Z</updated>
        </entry>
      </item>
      <item id='4e30f35051b7b8b42abe083742187228'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Alone</title>
          <summary>
Now I am alone.
O, what a rogue and peasant slave am I!
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-13T11:09:53Z</published>
          <updated>2003-12-13T11:09:53Z</updated>
        </entry>
      </item>
    </items>
    <set xmlns='http://jabber.org/protocol/rsm'>
      <first index='0'>368866411b877c30064a5f62b917cffe</first>
      <last>4e30f35051b7b8b42abe083742187228</last>
      <count>19</count>
```

```
    </set>
  </pubsub>
</iq>
```

6.5.5 Returning the Last Published Item

Even if the service or node does not support persistent items, it MAY return the last published item.

Listing 81: Service returns last published item

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
Whether 'tis␣nobler␣in␣the␣mind␣to␣suffer
The␣slings␣and␣arrows␣of␣outrageous␣fortune,
Or␣to␣take␣arms␣against␣a␣sea␣of␣troubles,
And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/atom03'/>
␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
␣␣␣␣␣␣␣␣</entry>
␣␣␣␣␣␣</item>
␣␣␣␣</items>
␣␣</pubsub>
</iq>
```

6.5.6 Returning Notifications Only

A service MAY return event notifications without payloads (e.g., to conserve bandwidth). If so, the client MAY request a specific item (using the ItemID) in order to retrieve the payload. When an entity requests items by ItemID, implementations MUST allow multiple items to be specified in the request.

Listing 82: Subscriber requests specific items by ItemID

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='items3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'>
      <item id='368866411b877c30064a5f62b917cffe'/>
      <item id='4e30f35051b7b8b42abe083742187228'/>
    </items>
  </pubsub>
</iq>
```

Listing 83: Service sends requested item(s)

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    id='items3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'>
      <item id='368866411b877c30064a5f62b917cffe'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>The Uses of This World</title>
          <summary>
O, that this too too solid flesh would melt
Thaw and resolve itself into a dew!
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-12T17:47:23Z</published>
          <updated>2003-12-12T17:47:23Z</updated>
        </entry>
      </item>
      <item id='4e30f35051b7b8b42abe083742187228'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Alone</title>
          <summary>
Now I am alone.
O, what a rogue and peasant slave am I!
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-13T11:09:53Z</published>
          <updated>2003-12-13T11:09:53Z</updated>
        </entry>
      </item>
    </items>
```

```
  </pubsub>
</iq>
```

6.5.7 Requesting the Most Recent Items

A service MAY allow entities to request the most recent N items by using the 'max_items' attribute. When max_items is used, implementations SHOULD return the N most recent (as opposed to the N oldest) items. (Note: A future version of this specification may recommend the use of XEP-0059 instead of the 'max_items' attribute.)

Listing 84: Subscriber requests two most recent items

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='items2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings' max_items='2'/>
  </pubsub>
</iq>
```

Listing 85: Service returns two most recent items

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'>
      <item id='4e30f35051b7b8b42abe083742187228'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Alone</title>
          <summary>
Now I am alone.
O, what a rogue and peasant slave am I!
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-13T11:09:53Z</published>
          <updated>2003-12-13T11:09:53Z</updated>
        </entry>
      </item>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
```

```
Whether 'tis␣nobler␣in␣the␣mind␣to␣suffer
The␣slings␣and␣arrows␣of␣outrageous␣fortune,
Or␣to␣take␣arms␣against␣a␣sea␣of␣troubles,
And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/atom03'/>
␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
␣␣␣␣␣␣␣␣␣␣</entry>
␣␣␣␣␣␣␣␣</item>
␣␣␣␣␣␣␣␣</items>
␣␣␣␣</pubsub>
</iq>
```

If there are fewer items in the node than requested, the service should return all of them.

Listing 86: Service returns no items if it has none

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'/>
  </pubsub>
</iq>
```

6.5.8 Requesting a Particular Item

The subscriber can request a particular item by specifying the Node ID and the appropriate ItemID.

Listing 87: Subscriber requests a particular item

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='items3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </items>
  </pubsub>
</iq>
```

The service would then return that specific item, if available.

6.5.9 Error Cases

There are several reasons why the items retrieval request might fail:

1. The requesting entity has multiple subscriptions to the node but does not specify a subscription ID.
2. The requesting entity is subscribed but specifies an invalid subscription ID.
3. The node does not return items to unsubscribed entities and the requesting entity is not subscribed.
4. The service or node does not support persistent items and does not return the last published item.
5. The service or node does not support item retrieval.
6. The node has an access model of "presence" and the requesting entity is not subscribed to the owner's presence.
7. The node has an access model of "roster" and the requesting entity is not in one of the authorized roster groups.
8. The node has an access model of "whitelist" and the requesting entity is not on the whitelist.
9. The service or node requires payment for item retrieval.
10. The requesting entity is blocked from retrieving items from the node (e.g., because having an affiliation of outcast).
11. The node does not exist.

These error cases are described more fully in the following sections.
If the requesting entity has multiple subscriptions to the node but does not specify a subscription ID, the service MUST return a <bad-request/> error to the subscriber, which SHOULD also include a pubsub-specific error condition of <subid-required/>.

Listing 88: Entity did not specify SubID

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <subid-required xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

If the requesting entity is subscribed but specifies an invalid subscription ID, the service MUST return a <not-acceptable/> error to the subscriber, which SHOULD also include a pubsub-specific error condition of <invalid-subid/>.

Listing 89: Entity specified invalid SubID

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <invalid-subid xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

If the node does not return items to unsubscribed entities and the requesting entity is not subscribed (which includes having a pending subscription), the service MUST return a <not-authorized/> error to the subscriber, which SHOULD also include a pubsub-specific error condition of <not-subscribed/>.

Listing 90: Entity is not subscribed

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='auth'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <not-subscribed xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

If the service or node does not support persistent items and does not return the last published item, the service MUST return a <feature-not-implemented/> error to the subscriber, specifying a pubsub-specific error condition of <unsupported/> and a feature of "persistent-items".

Listing 91: Persistent items not supported

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='persistent-items'/>
```

```
  </error>
</iq>
```

If the service or node does not support item retrieval (e.g., because the node is a collection node as described in XEP-0248), the service MUST return a <feature-not-implemented/> error to the subscriber, specifying a pubsub-specific error condition of <unsupported/> and a feature of "retrieve-items".

Listing 92: Item retrieval not supported

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='retrieve-items'/>
  </error>
</iq>
```

For nodes with an access model of "presence", if the requesting entity is not subscribed to the owner's presence then the pubsub service MUST respond with a <not-authorized/> error, which SHOULD also include a pubsub-specific error condition of <presence-subscription-required/>.

Listing 93: Entity is not authorized to retrieve items (presence subscription required)

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='auth'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <presence-subscription-required xmlns='http://jabber.org/protocol/
        pubsub#errors'/>
  </error>
</iq>
```

For nodes with an access model of "roster", if the requesting entity is not in one of the authorized roster groups then the pubsub service MUST respond with a <not-authorized/> error, which SHOULD also include a pubsub-specific error condition of <not-in-roster-group/>.

Listing 94: Entity is not authorized to retrieve items (not in roster group)

```
<iq type='error'
    from='pubsub.shakespeare.lit'
```

```
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='auth'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <not-in-roster-group xmlns='http://jabber.org/protocol/pubsub#
        errors'/>
  </error>
</iq>
```

For nodes with a node access model of "whitelist", if the requesting entity is not on the whitelist then the service MUST return a <not-allowed/> error, specifying a pubsub-specific error condition of <closed-node/>.

Listing 95: Node has whitelist access model

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <closed-node xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

Commercial deployments may wish to link subscribers to a database of paying customers. If the subscriber needs to provide payment in order to retrieve items from the node (e.g., if the subscriber is not in the customer database or the customer's account is not paid up), the service SHOULD return a <payment-required/> error to the subscriber.

Listing 96: Payment is required to retrieve items

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='auth'>
    <payment-required xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requesting entity is blocked from retrieving items (e.g., because having an affiliation of outcast), the service MUST return a <forbidden/> error to the subscriber.

Listing 97: Requesting entity is blocked

```
<iq type='error'
    from='pubsub.shakespeare.lit'
```

```
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node does not exist, the service SHOULD return an <item-not-found/> error to the subscriber.

Listing 98: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If there are no items at the node or the requested items do not exist, the service SHOULD return an IQ stanza of type "result" with an empty <items/> element.

Listing 99: No such item(s)

```
<iq from='pubsub.shakespeare.lit'
    id='items1'
    to='francisco@denmark.lit/barracks'
    type='result'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'/>
  </pubsub>
</iq>
```

7 Publisher Use Cases

7.1 Publish an Item to a Node

7.1.1 Request

A pubsub service MAY support the ability to publish items (if so, it MUST advertise support for the "http://jabber.org/protocol/pubsub#publish" feature). Any entity that is allowed to publish items to a node (i.e., a publisher or an owner) may do so at any time by sending an IQ-set to the service containing a pubsub element with a <publish/> child.
The syntax is as follows:

- The <publish/> element MUST possess a 'node' attribute, specifying the NodeID of the node.
- Depending on the node configuration, the <publish/> element MAY contain no <item/> elements or one <item/> element. [14] [15]
- The <item/> element provided by the publisher MAY possess an 'id' attribute, specifying a unique ItemID for the item. If an ItemID is not provided in the publish request, the pubsub service MUST generate one and MUST ensure that it is unique for that node.

Listing 100: Publisher publishes an item with an ItemID

```
<iq type='set'
    from='hamlet@denmark.lit/blogbot'
    to='pubsub.shakespeare.lit'
    id='publish1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='princely_musings'>
      <item id='bnd81g37d61f49fgn581'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them?
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32397</id>
          <published>2003-12-13T18:30:02Z</published>
          <updated>2003-12-13T18:30:02Z</updated>
        </entry>
      </item>
    </publish>
  </pubsub>
</iq>
```

Listing 101: Publisher publishes an item without an ItemID

```
<iq type='set'
```

[14] The inclusion of more than one <item/> element is no longer allowed, given the removal of batch publishing from version 1.13 of this specification.

[15] It is not necessary for a publication request to include a payload or even an <item/> element in order to trigger an event notification. For example, the result of publishing to a transient, notification-only node will be an event notification that does not include even an <item/> element. However, for the sake of convenience we refer to the act of publication as "publishing an item" (rather than, say, "triggering an event notification") even though a publication request will not always contain an <item/> element.

```
    from='hamlet@denmark.lit/blogbot'
    to='pubsub.shakespeare.lit'
    id='publish2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='princely_musings'>
      <item>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
Whether 'tis␣nobler␣in␣the␣mind␣to␣suffer
The␣slings␣and␣arrows␣of␣outrageous␣fortune,
Or␣to␣take␣arms␣against␣a␣sea␣of␣troubles,
And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/atom03'/>
␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
␣␣␣␣␣␣␣␣␣␣</entry>
␣␣␣␣␣␣␣␣␣␣</item>
␣␣␣␣␣␣␣␣␣␣</publish>
␣␣␣␣␣␣␣␣␣␣</pubsub>
</iq>
```

7.1.2 Success Case

If the pubsub service can successfully process the request, it MUST inform the publisher of success. If the publish request did not include an ItemID, the IQ-result SHOULD include an empty <item/> element that specifies the ItemID of the published item.

Listing 102: Service replies with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/blogbot'
    id='publish1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='princely_musings'/>
  </pubsub>
</iq>
```

Listing 103: Service replies with success and the ID of the new item

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/blogbot'
```

```
    id='publish2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </publish>
  </pubsub>
</iq>
```

Note: If the publisher previously published an item with the same ItemID, successfully processing the request means that the service MUST overwrite the old item with the new item and then proceed as follows.
The pubsub service MUST then send one event notification to each entity that meets the criteria for receiving an event notification (typically to each approved subscriber, although there are other contexts in which an entity may receive an event notification as summarized under Notification Triggers). Each <message/> stanza generated by a pubsub service SHOULD possess an 'id' attribute with a unique value so that the service can properly track any notification-related errors that may occur (see the Handling Notification-Related Errors section of this document). Depending on the node configuration, the event notification either will or will not contain the payload, as shown below.
Note: In order to facilitate authorization for item removal as described in the Delete an Item from a Node section of this document, implementations that support persistent items SHOULD store the item (if the node is so configured) and maintain a record of the publisher.
If the node is configured to include payloads, the subscribers will receive payloads with the event notifications.

Listing 104: Subscribers receive event notifications with payloads

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
Whether 'tis␣nobler␣in␣the␣mind␣to␣suffer
The␣slings␣and␣arrows␣of␣outrageous␣fortune,
Or␣to␣take␣arms␣against␣a␣sea␣of␣troubles,
And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/atom03'/>
␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
␣␣␣␣␣␣␣␣␣␣</entry>
```

```
␣␣␣␣␣␣</item>
␣␣␣␣</items>
␣␣</event>
</message>

<message␣from='pubsub.shakespeare.lit'␣to='bernardo@denmark.lit'␣id='
    bar'>
␣␣<event␣xmlns='http://jabber.org/protocol/pubsub#event'>
␣␣␣␣<items␣node='princely_musings'>
␣␣␣␣␣␣<item␣id='ae890ac52d0df67ed7cfdf51b644e901'>
␣␣␣␣␣␣␣␣<entry␣xmlns='http://www.w3.org/2005/Atom'>
␣␣␣␣␣␣␣␣␣␣<title>Soliloquy</title>
␣␣␣␣␣␣␣␣␣␣<summary>
To␣be,␣or␣not␣to␣be:␣that␣is␣the␣question:
Whether␣'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them?
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32397</id>
          <published>2003-12-13T18:30:02Z</published>
          <updated>2003-12-13T18:30:02Z</updated>
        </entry>
      </item>
    </items>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='horatio@denmark.lit' id='
    baz'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
Whether '␣tis␣nobler␣in␣the␣mind␣to␣suffer
The␣slings␣and␣arrows␣of␣outrageous␣fortune,
Or␣to␣take␣arms␣against␣a␣sea␣of␣troubles,
And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/atom03'/>
␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
```

```
␣␣␣␣␣␣␣␣</entry>
␣␣␣␣␣␣</item>
␣␣␣␣</items>
␣␣</event>
</message>

<message␣from='pubsub.shakespeare.lit'␣to='bard@shakespeare.lit'␣id='
    fez'>
␣␣<event␣xmlns='http://jabber.org/protocol/pubsub#event'>
␣␣␣␣<items␣node='princely_musings'>
␣␣␣␣␣␣<item␣id='ae890ac52d0df67ed7cfdf51b644e901'>
␣␣␣␣␣␣␣␣<entry␣xmlns='http://www.w3.org/2005/Atom'>
␣␣␣␣␣␣␣␣␣␣<title>Soliloquy</title>
␣␣␣␣␣␣␣␣␣␣<summary>
To␣be,␣or␣not␣to␣be:␣that␣is␣the␣question:
Whether␣'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them?
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32397</id>
          <published>2003-12-13T18:30:02Z</published>
          <updated>2003-12-13T18:30:02Z</updated>
        </entry>
      </item>
    </items>
  </event>
</message>
```

If the node is configured to not include payloads, the subscribers will receive event notifications only. (If payloads are not included, subscribers may request the published item via the protocol defined in the Retrieve Items from a Node section of this document.)

Listing 105: Subscribers receive event notifications only

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </items>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='bernardo@denmark.lit' id='
    bar'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
```

```
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </items>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='horatio@denmark.lit' id='
    baz'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </items>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='bard@shakespeare.lit' id='
    fez'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </items>
  </event>
</message>
```

If configured to do so (see Associating Events and Payloads with the Generating Entity), the service can include the publisher of the item when it generates event notifications.

Listing 106: Service Notifies Subscribers

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'
            publisher='hamlet@denmark.lit'>
        [ ... ENTRY ... ]
      </item>
    </items>
  </event>
</message>
```

If so, the service MUST also include the publisher with every other form of item retrieval.

Listing 107: Service returns items

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='items1'>
```

```
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'
            publisher='hamlet@denmark.lit'>
        [ ... ENTRY ... ]
      </item>
        [ ... MORE ITEMS ... ]
    </items>
  </pubsub>
</iq>
```

The value of the 'publisher' attribute MUST be generated by the service, not accepted by the service in the published item, since allowing the publisher to assert its JID would open the possibility of spoofing.
The JID stamped by the service can be either (1) the full JID <localpart@domain.tld/resource> of the publisher as taken the 'from' attribute of the IQ-set used to publish the item or (2) the bare JID <localpart@domain.tld> of the publisher as derived from a formal affiliation in the explicit list of whitelisted publishers.
If a single entity is subscribed to a node multiple times, the service SHOULD notate the event notification so that the entity can determine which subscription identifier(s) generated this event. If these notations are included, they MUST use the Stanza Headers and Internet Metadata (XEP-0131) [16] format and SHOULD be included after the event notification information (i.e., as the last child of the <message/> stanza).

Listing 108: Subscriber receives notated event notification

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </items>
  </event>
  <headers xmlns='http://jabber.org/protocol/shim'>
    <header name='SubID'>123-abc</header>
    <header name='SubID'>004-yyy</header>
  </headers>
</message>
```

If the service or node is configured so that there is a maximum number of items cached at the node and the maximum is reached when an item is published, the service MUST delete one of the existing items or reject the new item.
The behaviour of the service is determined by the "pubsub#publish_node_full" option which may take one of the following values: "retract-oldest", "discard-oldest", or "reject".
The "retract-oldest" option is the default one and SHOULD be implemented even if this

[16] XEP-0131: Stanza Headers and Internet Metadata <https://xmpp.org/extensions/xep-0131.html>.

configuration isn't supported.
It is RECOMMENDED for the service to follow the "first in, first out" rule and retract the oldest item. Depending on node configuration, deletion of an existing item MAY result in sending of a retract notification to the subscribers.
When configured as "discard-oldest", a service MUST discard the oldest item.
The "reject" option for this configuration is described in Node is full publishing errors.

7.1.3 Error Cases

There are several reasons why the publish request might fail:

1. The requesting entity does not have sufficient privileges to publish.
2. The node does not support item publication.
3. The node does not exist.
4. The node is full (max_items exceeded).
5. The payload size exceeds a service-defined limit.
6. The item contains more than one payload element or the namespace of the root payload element does not match the configured namespace for the node.
7. The request does not match the node configuration.

These error cases are described more fully in the following sections.
Note: If a publisher publishes an item with an Item ID and the ItemID matches that of an existing item, the pubsub service MUST NOT fail the publication but instead MUST overwrite the existing item and generate a new event notification (i.e., re-publication is equivalent to modification).
If the requesting entity does not have sufficient privileges to publish, the service MUST return a <forbidden/> error.

Listing 109: Entity does not have sufficient privileges to publish to node

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='publish1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node does not support item publication (e.g., because it is a collection node as described in XEP-0248), the service MUST return a <feature-not-implemented/> error, specifying a

pubsub-specific error condition of <unsupported/> and a feature of "publish".

Listing 110: Node does not support item publication

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='publish1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='publish'/>
  </error>
</iq>
```

If the requesting entity attempts to publish an item to a node that does not exist and the service does not support the "auto-create" feature (see Automatic Node Creation), the service MUST return an <item-not-found/> error.

Listing 111: Entity attempts to publish to a non-existent node

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='publish1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requesting entity attempts to publish an item to a node that has reached its maximum number of items, visible via pubsub#max_items, and the pubsub#publish_full_node configuration value is set to "reject", then the service MUST return a <conflict/> error, which SHOULD also include a pubsub-specific error condition of <node-full/>.

Listing 112: Entity attempts to publish to a node with max_items reached

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='publish1'>
  <error type='cancel'
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <node-full xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

If the payload size exceeds a service-defined limit, the service MUST return a <not-acceptable/> error, which SHOULD also include a pubsub-specific error condition of <payload-too-big/>.

Listing 113: Entity attempts to publish very large payload

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='publish1'>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <payload-too-big xmlns='http://jabber.org/protocol/pubsub#errors'/
        >
  </error>
</iq>
```

If the <item/> element does not contain exactly one payload element or the namespace of the root payload element does not match the configured namespace for the node, the service MUST bounce the request with a <bad-request/> error, which SHOULD also include a pubsub-specific error condition of <invalid-payload/>.

Listing 114: Entity attempts to publish item with multiple payload elements or namespace does not match

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='publish1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <invalid-payload xmlns='http://jabber.org/protocol/pubsub#errors'/
        >
  </error>
</iq>
```

If the request does not conform to the configured event type for the node, the service MAY bounce the request with a <bad-request/> error, which SHOULD also include a pubsub-specific error condition. The following rules apply:

- If the event type is persistent (either event notification or payload) and the publisher does not specify an ItemID, the service MUST generate the ItemID and MUST NOT bounce the publication request.
- If the event type is persistent (either event notification or payload) and the publisher does not include an item, the service MUST bounce the publication request with a <bad-request/> error and a pubsub-specific error condition of <item-required/>.

- If the event type is payload (either persistent or transient) and the publisher does not include a payload, the service SHOULD bounce the publication request with a <bad-request/> error and a pubsub-specific error condition of <payload-required/>.
- If the event type is notification + transient and the publisher provides an item, the service MUST bounce the publication request with a <bad-request/> error and a pubsub-specific error condition of <item-forbidden/>.

Examples of these errors are shown below.

Listing 115: Publisher attempts to publish to persistent node with no item

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='publish1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <item-required xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

Listing 116: Publisher attempts to publish to payload node with no payload

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='publish1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <payload-required xmlns='http://jabber.org/protocol/pubsub#errors'
       />
  </error>
</iq>
```

Listing 117: Publisher attempts to publish to transient notification node with item

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='publish1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <item-forbidden xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
</iq>
```

7.1.4 Automatic Node Creation

A pubsub service MAY automatically create a node when it receives a publish request sent to a node that does not exist (instead of returning an <item-not-found/> error). When doing so, the service SHOULD apply the default node configuration. If a service supports this functionality, it MUST advertise that fact by including a feature of "http://jabber.org/protocol/pubsub#auto-create" in its disco#info responses.

7.1.5 Publishing Options

A pubsub service MAY support the ability to specify options along with a publish request (if so, it MUST advertise support for the "http://jabber.org/protocol/pubsub#publish-options" feature). Here is an example:

Listing 118: Publishing with options

```
<iq type='set'
    from='hamlet@denmark.lit/blogbot'
    to='pubsub.shakespeare.lit'
    id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Soliloquy</title>
          <summary>
To be, or not to be: that is the question:
Whether 'tis␣nobler␣in␣the␣mind␣to␣suffer
The␣slings␣and␣arrows␣of␣outrageous␣fortune,
Or␣to␣take␣arms␣against␣a␣sea␣of␣troubles,
And␣by␣opposing␣end␣them?
␣␣␣␣␣␣␣␣␣␣</summary>
␣␣␣␣␣␣␣␣␣␣<link␣rel='alternate'␣type='text/html'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣href='http://denmark.lit/2003/12/13/atom03'/>
␣␣␣␣␣␣␣␣␣␣<id>tag:denmark.lit,2003:entry-32397</id>
␣␣␣␣␣␣␣␣␣␣<published>2003-12-13T18:30:02Z</published>
␣␣␣␣␣␣␣␣␣␣<updated>2003-12-13T18:30:02Z</updated>
␣␣␣␣␣␣␣␣</entry>
␣␣␣␣␣␣</item>
␣␣␣␣</publish>
␣␣␣␣<publish-options>
␣␣␣␣␣␣<x␣xmlns='jabber:x:data'␣type='submit'>
␣␣␣␣␣␣␣␣<field␣var='FORM_TYPE'␣type='hidden'>
␣␣␣␣␣␣␣␣␣␣<value>http://jabber.org/protocol/pubsub#publish-options</
    value>
␣␣␣␣␣␣␣␣</field>
␣␣␣␣␣␣␣␣<field␣var='pubsub#access_model'>
```

```
          <value>presence</value>
        </field>
      </x>
    </publish-options>
  </pubsub>
</iq>
```

The <publish-options/> element MUST contain a data form (see XEP-0004), whose FORM_TYPE MUST be "http://jabber.org/protocol/pubsub#publish-options" (see XEP-0068).
Each form field denotes a precondition to publishing the request. A pub-sub service advertising support for publishing options MUST check each precondition field against the node configuration of the same name, and it MUST reject the publication upon encountering unknown fields.
Preconditions MUST be processed as follows:

1. If the node exists and the precondition is not met, then the publish MUST fail with a <conflict/> error condition and a pubsub-specific condition of <precondition-not-met/>.
2. If the node exists and the precondition is met, then the publish succeeds.
3. If the node does not exist and the service supports the "auto-create" feature, then the service shall auto-create the node with default configuration in all respects except those specified in the preconditions, and the publish succeeds.
4. If the node does not exist and the service does not support the "auto-create" feature, then the publish shall fail.

7.2 Delete an Item from a Node

A publisher might want to delete an item once it has been published to a node that supports persistent items. Support for this feature ("delete-items") is RECOMMENDED.

7.2.1 Request

To delete an item, the publisher sends a retract request as shown in the following examples. The <retract/> element MUST possess a 'node' attribute, MAY possess a 'notify' attribute, and MUST contain one <item/> element; the <item/> element MUST be empty and MUST possess an 'id' attribute.

Listing 119: Entity deletes an item from a node

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='retract1'>
```

```
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <retract node='princely_musings'>
      <item id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </retract>
  </pubsub>
</iq>
```

7.2.2 Success Case

If no error occurs, the service MUST delete the item.

Listing 120: Service replies with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='retract1'/>
```

If no error occurs and the <retract/> element included a 'notify' attribute with a value of "true" or "1" [17], then the service MUST delete the item and MUST notify all subscribers as shown below. The syntax is identical to event notifications except that instead of an <item/> element, the <items/> element includes a <retract/> element.

Listing 121: Subscribers are notified of deletion

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <retract id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </items>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='bernardo@denmark.lit' id='
    bar'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <retract id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </items>
  </event>
</message>
```

[17] In accordance with Section 3.2.2.1 of XML Schema Part 2: Datatypes, the allowable lexical representations for the xs:boolean datatype are the strings "0" and "false" for the concept 'false' and the strings "1" and "true" for the concept 'true'; implementations MUST support both styles of lexical representation.

If a single entity is subscribed to the node multiple times, the service SHOULD notate the item deletion so that the entity can determine which subscription identifier(s) generated this event. As above, if these notations are included, they MUST use the Stanza Headers and Internet Metadata (SHIM) protocol and SHOULD be included after the notification data (i.e., as the last child of the <message/> stanza).

Listing 122: Subscriber receives notated event notification

```
<message from='pubsub.shakespeare.lit' to='bernardo@denmark.lit' id='
    bar'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <retract id='ae890ac52d0df67ed7cfdf51b644e901'/>
    </items>
  </event>
  <headers xmlns='http://jabber.org/protocol/shim'>
    <header name='SubID'>123-abc</header>
    <header name='SubID'>004-yyy</header>
  </headers>
</message>
```

7.2.3 Error Cases

There are several reasons why the item retraction request might fail:

1. The publisher does not have sufficient privileges to delete the requested item.
2. The node or item does not exist.
3. The request does not specify a node.
4. The request does not include an <item/> element or the <item/> element does not specify an ItemID.
5. The node does not support persistent items.
6. The service does not support the deletion of items.

These error cases are described more fully in the following sections.
If the requesting entity does not have sufficient privileges to delete the item, the service MUST return a <forbidden/> error.

Listing 123: Requesting entity does not have sufficient privileges

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
```

```
    id='retract1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node or item does not exist, the service MUST return an <item-not-found/> error.

Listing 124: Non-existent node or item

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='retract1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the request does not specify a node, the service MUST return a <bad-request/> error, which SHOULD also include a pubsub-specific error condition of <nodeid-required/>.

Listing 125: Request does not specify a node

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='retract1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <nodeid-required xmlns='http://jabber.org/protocol/pubsub#errors'/
        >
  </error>
</iq>
```

If the request does not include an <item/> element or the <item/> element does not specify an ItemID, the service MUST return a <bad-request/> error, which SHOULD also include a pubsub-specific error condition of <item-required/>.

Listing 126: Request does not specify an item

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='retract1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <item-required xmlns='http://jabber.org/protocol/pubsub#errors'/>
  </error>
```

```
</iq>
```

If the node does not support persistent items (e.g., because it is a collection node or a transient node that does not deliver payloads), the service MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "persistent-items".

Listing 127: Node does not support persistent items

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='retract1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='persistent-items'/>
  </error>
</iq>
```

If the service does not support item deletion, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "delete-items".

Listing 128: Service does not support item deletion

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='retract1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='delete-items'/>
  </error>
</iq>
```

8 Owner Use Cases

8.1 Create a Node

8.1.1 General Considerations

An entity may want to create a new node. Support for this feature ("create-nodes") is RECOMMENDED. However, a service MAY disallow creation of nodes based on the identity of

the requesting entity, or MAY disallow node creation altogether (e.g., reserving that privilege to a service-wide administrator).
There are two ways to create a node:

1. Create a node with default configuration for the specified node type.
2. Create and configure a node simultaneously.

Listing 129: Request to create a node

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='create1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <create node='princely_musings'/>
  </pubsub>
</iq>
```

These methods, along with method-specific error conditions, are explained more fully in the following sections.
In addition to method-specific error conditions, there are several general reasons why the node creation request might fail:

- The service does not support node creation.
- Only entities that are registered with the service are allowed to create nodes but the requesting entity is not registered.
- The requesting entity does not have sufficient privileges to create nodes.
- The requested NodeID already exists.
- The request did not include a NodeID and "instant nodes" are not supported.

These general error cases are described more fully below.
If the service does not support node creation, it MUST respond with a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "create-nodes".

Listing 130: Service does not support node creation

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='create1'>
  <error type='cancel'>
```

```
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='create-nodes'/>
  </error>
</iq>
```

If only entities that are registered with the service may create nodes but the requesting entity has not yet registered, the service MUST respond with a <registration-required/> error.

Listing 131: Service requires registration

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='create1'>
  <error type='auth'>
    <registration-required xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'
        />
  </error>
</iq>
```

If the requesting entity does not have sufficient privileges to create nodes, the service MUST respond with a <forbidden/> error.

Listing 132: Requesting entity is prohibited from creating nodes

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='create1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requested NodeID already exists, the service MUST respond with a <conflict/> error.

Listing 133: NodeID already exists

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='create1'>
  <error type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node creator does not specify a NodeID but the service does not support instant nodes, the service MUST return a <not-acceptable/> error, specifying a pubsub-specific error condition of <nodeid-required/>.

Listing 134: Service does not support instant nodes

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='create2'>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <nodeid-required xmlns='http://jabber.org/protocol/pubsub#errors'/
       >
  </error>
</iq>
```

If the node creator does not specify a NodeID but the service supports instant nodes, the service SHOULD generate a NodeID that is unique within the context of the service on behalf of the node creator.

Listing 135: Entity requests an instant node

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='create2'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <create/>
    </pubsub>
</iq>
```

If no error occurs, the pubsub service SHOULD create the node, generate a NodeID that is unique within the context of that service, and inform the user of success (including the NodeID in the response).

Listing 136: Service replies with success and generated NodeID

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='create2'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <create node='25e3d37dabbab9541f7523321421edc5bfeb2dae'/>
    </pubsub>
</iq>
```

Note: When a service successfully creates a node on behalf of the requesting entity, it MUST return an IQ result (in accordance with XMPP Core). If the node creation request did not specify a NodeID and the service supports creation of instant nodes, the service MUST specify the created NodeID in the IQ result. Similarly, if the node creation request specified a NodeID but the service modified the NodeID before creating the node, the service MUST also specify the modified node in the IQ result. In all other cases, the service MAY specify the NodeID in the IQ result but the node creator MUST NOT depend on receiving it from the service (since the node creator can determine which node was created by tracking the 'id' attribute that it specified for the IQ-set).

8.1.2 Create a Node With Default Configuration

As explained above, each node type has its own default configuration. By asking the service to create a node with default configuration, the node creator accepts the default configuration. If the service allows node configuration, the owner may reconfigure the node after creating the node (as described in the Configure a Node section of this document). In addition, a service MAY allow entities to determine the default configuration options for a given node type before creating a node (as described in the Request Default Node Configurations section of this document).
In order to create a node with default configuration, the node creator can simply include an empty <create/> child element.
In the following example, the node creator requests a leaf node (the default type) with an open access model (assumed to be the default type for this service).

Listing 137: Entity requests leaf node with (default) open access model

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='create1'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <create node='princely_musings'/>
    </pubsub>
</iq>
```

Note: The default setting for the 'pubsub#node_type' configuration field is "leaf".
In order to request an access model other than the default for the service, the node creator MUST include a Data Form in the node creation request that specifies a non-default value for the 'pubsub#access_model' field.

Listing 138: Entity requests leaf node with non-default access model

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
```

```
    id='create2'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <create node='princely_musings'/>
      <configure>
        <x xmlns='jabber:x:data' type='submit'>
          <field var='FORM_TYPE' type='hidden'>
            <value>http://jabber.org/protocol/pubsub#node_config</
                value>
          </field>
          <field var='pubsub#access_model'><value>whitelist</value></
              field>
        </x>
      </configure>
    </pubsub>
</iq>
```

If the access model is supported and none of the general or method-specific errors has occurred, the service SHOULD create the node and inform the requesting entity of success.

Listing 139: Service informs requesting entity of success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='create1'/>
```

If service does not support the specified access model, it MUST return a <not-acceptable/> error, specifying a pubsub-specific error condition of <unsupported-access-model/>.

Listing 140: Service does not support specified access model

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='create2'>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <unsupported-access-model xmlns='http://jabber.org/protocol/pubsub
        #errors'/>
  </error>
</iq>
```

(For error handling if the service does not support the specified node type, refer to XEP-0248.)

8.1.3 Create and Configure a Node

If an implementation allows node configuration (see the Configure a Node section of this document), it SHOULD allow node creation requests to contain the desired node configuration

in the node creation request.
Note: The <configure/> element MUST follow the <create/> element and MUST NOT possess a 'node' attribute, since the value of the <create/> element's 'node' attribute specifies the desired NodeID; if any of these rules are violated, the service MUST return a <bad-request/> error.

Listing 141: Entity requests a new node with non-default configuration.

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='create1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <create node='princely_musings'/>
    <configure>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='pubsub#title'><value>Princely Musings (Atom)</
            value></field>
        <field var='pubsub#deliver_notifications'><value>1</value></
            field>
        <field var='pubsub#deliver_payloads'><value>1</value></field>
        <field var='pubsub#persist_items'><value>1</value></field>
        <field var='pubsub#max_items'><value>10</value></field>
        <field var='pubsub#item_expire'><value>604800</value></field>
        <field var='pubsub#access_model'><value>open</value></field>
        <field var='pubsub#publish_model'><value>publishers</value></
            field>
        <field var='pubsub#purge_offline'><value>0</value></field>
        <field var='pubsub#send_last_published_item'><value>never</
            value></field>
        <field var='pubsub#presence_based_delivery'><value>false</
            value></field>
        <field var='pubsub#notification_type'><value>headline</value><
            /field>
        <field var='pubsub#notify_config'><value>0</value></field>
        <field var='pubsub#notify_delete'><value>0</value></field>
        <field var='pubsub#notify_retract'><value>0</value></field>
        <field var='pubsub#notify_sub'><value>0</value></field>
        <field var='pubsub#max_payload_size'><value>1028</value></
            field>
        <field var='pubsub#type'><value>urn:example:e2ee:bundle</value
            ></field>
        <field var='pubsub#body_xslt'>
          <value>http://jabxslt.jabberstudio.org/atom_body.xslt</value
              >
        </field>
      </x>
```

```
    </configure>
  </pubsub>
</iq>
```

Listing 142: Service replies with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='create1'/>
```

If a service supports this "create-and-configure" feature, it MUST advertise that fact by returning a feature of "http://jabber.org/protocol/pubsub#create-and-configure" in response to service discovery information requests. If the create-and-configure option is not supported but the requesting entity sends such a request anyway, the service SHOULD ignore the configuration part of the request and proceed as if it had not been included.

8.2 Configure a Node

After creating a new node, the node owner may want to modify the node configuration. Support for this feature is RECOMMENDED.

8.2.1 Request

Listing 143: Owner requests configuration form

```
<iq type='get'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='config1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <configure node='princely_musings'/>
  </pubsub>
</iq>
```

8.2.2 Success Case

If no error occurs, the server MUST return a configuration form to the node owner, which SHOULD contain the current node configuration as the default values.
Note: The following example shows some of the possible configuration options that MAY be provided. If an implementation implements these features using the **Data Forms** protocol, that implementation MUST use the fields that are registered with the XMPP Registrar in association with the 'http://jabber.org/protocol/pubsub' namespace (a preliminary representation of those field variables is shown below and in the pubsub#node_config FORM_TYPE

section of this document, but MUST NOT be construed as canonical, since the XMPP Registrar may standardize additional fields at a later date without changes to this document). An implementation MAY choose to specify different labels, values, and even field types, but MUST conform to the defined variable naming scheme.

Listing 144: Service responds with configuration form

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='config1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <configure node='princely_musings'>
      <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='pubsub#title' type='text-single'
               label='A␣friendly␣name␣for␣the␣node'/>
        <field var='pubsub#deliver_notifications' type='boolean'
               label='Whether␣to␣deliver␣event␣notifications'>
          <value>true</value>
        </field>
        <field var='pubsub#deliver_payloads' type='boolean'
               label='Whether␣to␣deliver␣payloads␣with␣event␣
                  notifications'>
          <value>true</value>
        </field>
        <field var='pubsub#notify_config' type='boolean'
               label='Notify␣subscribers␣when␣the␣node␣configuration␣
                  changes'>
          <value>0</value>
        </field>
        <field var='pubsub#notify_delete' type='boolean'
               label='Notify␣subscribers␣when␣the␣node␣is␣deleted'>
          <value>false</value>
        </field>
        <field var='pubsub#notify_retract' type='boolean'
               label='Notify␣subscribers␣when␣items␣are␣removed␣from␣
                  the␣node'>
          <value>false</value>
        </field>
        <field var='pubsub#notify_sub' type='boolean'
               label='Notify␣owners␣about␣new␣subscribers␣and␣
                  unsubscribes'>
          <value>0</value>
        </field>
        <field var='pubsub#persist_items' type='boolean'
               label='Persist␣items␣to␣storage'>
```

```
        <value>1</value>
      </field>
      <field var='pubsub#max_items' type='text-single'
             label='Max␣#␣of␣items␣to␣persist.␣‘max‘␣for␣no␣specific
                ␣limit␣other␣than␣a␣server␣imposed␣maximum.'>
        <value>10</value>
      </field>
      <field var='pubsub#item_expire' type='text-single'
             label='Time␣after␣which␣to␣automatically␣purge␣items.␣‘
                max‘␣for␣no␣specific␣limit␣other␣than␣a␣server␣
                imposed␣maximum.'>
        <value>604800</value>
      </field>
      <field var='pubsub#subscribe' type='boolean'
             label='Whether␣to␣allow␣subscriptions'>
        <value>1</value>
      </field>
      <field var='pubsub#access_model' type='list-single'
             label='Specify␣the␣subscriber␣model'>
        <option><value>authorize</value></option>
        <option><value>open</value></option>
        <option><value>presence</value></option>
        <option><value>roster</value></option>
        <option><value>whitelist</value></option>
        <value>open</value>
      </field>
      <field var='pubsub#roster_groups_allowed' type='list-multi'
             label='Roster␣groups␣allowed␣to␣subscribe'>
        <option><value>friends</value></option>
        <option><value>courtiers</value></option>
        <option><value>servants</value></option>
        <option><value>enemies</value></option>
      </field>
      <field var='pubsub#publish_model' type='list-single'
             label='Specify␣the␣publisher␣model'>
        <option><value>publishers</value></option>
        <option><value>subscribers</value></option>
        <option><value>open</value></option>
        <value>publishers</value>
      </field>
      <field var='pubsub#purge_offline' type='boolean'
             label='Purge␣all␣items␣when␣the␣relevant␣publisher␣goes
                ␣offline?'>
        <value>0</value>
      </field>
      <field var='pubsub#max_payload_size' type='text-single'
             label='Max␣Payload␣size␣in␣bytes'>
        <value>1028</value>
      </field>
```

```
        <field var='pubsub#send_last_published_item' type='list-single
            '
              label='When␣to␣send␣the␣last␣published␣item'>
          <option label='Never'><value>never</value></option>
          <option label='When␣a␣new␣subscription␣is␣processed'><value>
              on_sub</value></option>
          <option label='When␣a␣new␣subscription␣is␣processed␣and␣
              whenever␣a␣subscriber␣comes␣online'>
            <value>on_sub_and_presence</value>
          </option>
          <value>never</value>
        </field>
        <field var='pubsub#presence_based_delivery' type='boolean'
              label='Deliver␣event␣notifications␣only␣to␣available␣
                  users'>
          <value>0</value>
        </field>
        <field var='pubsub#notification_type' type='list-single'
              label='Specify␣the␣delivery␣style␣for␣event␣
                  notifications'>
          <option><value>normal</value></option>
          <option><value>headline</value></option>
          <value>headline</value>
        </field>
        <field var='pubsub#type' type='text-single'
              label='Specify␣the␣semantic␣type␣of␣payload␣data␣to␣be␣
                  provided␣at␣this␣node.'>
          <value>urn:example:e2ee:bundle</value>
        </field>
        <field var='pubsub#dataform_xslt' type='text-single'
              label='Payload␣XSLT'/>
      </x>
    </configure>
  </pubsub>
</iq>
```

8.2.3 Error Cases

There are several reasons why the node configuration request might fail:

1. The service does not support node configuration.
2. The requesting entity does not have sufficient privileges to configure the node.
3. The request did not specify a node.
4. The node has no configuration options.

5. The specified node does not exist.

These error cases are described more fully in the following sections.
If the service does not support node configuration, the service MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "config-node".

Listing 145: Service does not support node configuration

```
<iq type='error'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='config1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='config-node'/>
  </error>
</iq>
```

If the requesting entity does not have sufficient privileges to configure the node, the service MUST respond with a <forbidden/> error.

Listing 146: Requesting entity is prohibited from configuring this node

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='config1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the request did not specify a node, the service SHOULD return a <bad-request/> error. It is possible that by not including a NodeID, the requesting entity is asking to configure the root node; however, if the requesting entity is not a service-level admin, it makes sense to return <bad-request/> instead of <forbidden/>.

Listing 147: Request did not specify a node

```
<iq type='error'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='config1'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
```

```
    <nodeid-required xmlns='http://jabber.org/protocol/pubsub#errors'/
        >
  </error>
</iq>
```

If no configuration options are available (e.g., because node configuration is "locked down"), the service MUST return a <not-allowed/> error to the owner.

Listing 148: Node has no configuration options

```
<iq type='error'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='config1'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node does not exist, the service MUST return an <item-not-found/> error.

Listing 149: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='config1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.2.4 Form Submission

After receiving the configuration form, the owner SHOULD submit a completed configuration form. Note that the rules of XEP-0004 for incomplete submission form handling apply.

Listing 150: Owner submits node configuration form

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='config2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <configure node='princely_musings'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
```

```
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='pubsub#title'><value>Princely Musings (Atom)</
           value></field>
        <field var='pubsub#deliver_notifications'><value>1</value></
           field>
        <field var='pubsub#deliver_payloads'><value>1</value></field>
        <field var='pubsub#persist_items'><value>1</value></field>
        <field var='pubsub#max_items'><value>10</value></field>
        <field var='pubsub#item_expire'><value>604800</value></field>
        <field var='pubsub#access_model'><value>roster</value></field>
        <field var='pubsub#roster_groups_allowed'>
          <value>friends</value>
          <value>servants</value>
          <value>courtiers</value>
        </field>
        <field var='pubsub#publish_model'><value>publishers</value></
           field>
        <field var='pubsub#purge_offline'><value>0</value></field>
        <field var='pubsub#send_last_published_item'><value>never</
           value></field>
        <field var='pubsub#presence_based_delivery'><value>false</
           value></field>
        <field var='pubsub#notification_type'><value>headline</value><
           /field>
        <field var='pubsub#notify_config'><value>0</value></field>
        <field var='pubsub#notify_delete'><value>0</value></field>
        <field var='pubsub#notify_retract'><value>0</value></field>
        <field var='pubsub#notify_sub'><value>0</value></field>
        <field var='pubsub#max_payload_size'><value>1028</value></
           field>
        <field var='pubsub#type'><value>urn:example:e2ee:bundle</value
           ></field>
        <field var='pubsub#body_xslt'>
          <value>http://jabxslt.jabberstudio.org/atom_body.xslt</value
             >
        </field>
      </x>
    </configure>
  </pubsub>
</iq>
```

Alternatively, the owner MAY cancel the configuration process, in which case the existing configuration MUST be applied.

Listing 151: Owner cancels configuration process

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
```

```
    to='pubsub.shakespeare.lit'
    id='config2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <configure node='princely_musings'>
      <x xmlns='jabber:x:data' type='cancel'/>
    </configure>
  </pubsub>
</iq>
```

8.2.5 Form Processing

If the form can be successfully processed, the service MUST return an IQ-result.

Listing 152: Service replies with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='config2'/>
```

Note: If the node type was changed from leaf to collection and there are items associated with the node, the service MUST purge the node of all items (with or without notifying the subscribers).
If the requested node configuration change cannot be processed (e.g., because the node owner has attempted to change the configuration so that there are no node owners), the service MUST return a <not-acceptable/> error to the owner.

Listing 153: Configuration change cannot be processed

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='config2'>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the "pubsub#notify_config" option is set to true, the service MUST notify subscribers of the configuration change. (A service SHOULD support this option for leaf nodes and MUST support it for collection nodes as described in XEP-0248.) If the node configuration is set to notification-only, the notification MUST consist of an empty <configuration/> element whose 'node' attribute is set to the NodeID of the node; if the node configuration is set to full payloads, the <configuration/> element MUST in addition contain the node configuration as represented via the **Data Forms** protocol.

Listing 154: Service sends configuration change notification (event notification only)

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <configuration node='princely_musings'/>
  </event>
</message>
```

Listing 155: Service sends configuration change notification (full payload)

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <configuration node='princely_musings'>
      <x xmlns='jabber:x:data' type='result'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='pubsub#title'><value>Princely Musings (Atom)</
            value></field>
        <field var='pubsub#deliver_notifications'><value>1</value></
            field>
        <field var='pubsub#deliver_payloads'><value>1</value></field>
        <field var='pubsub#notify_config'><value>0</value></field>
        <field var='pubsub#notify_delete'><value>0</value></field>
        <field var='pubsub#notify_retract'><value>0</value></field>
        <field var='pubsub#notify_sub'><value>0</value></field>
        <field var='pubsub#persist_items'><value>1</value></field>
        <field var='pubsub#max_items'><value>10</value></field>
        <field var='pubsub#item_expire'><value>604800</value></field>
        <field var='pubsub#subscribe'><value>1</value></field>
        <field var='pubsub#access_model'><value>open</value></field>
        <field var='pubsub#publish_model'><value>publishers</value></
            field>
        <field var='pubsub#purge_offline'><value>0</value></field>
        <field var='pubsub#max_payload_size'><value>9216</value></
            field>
        <field var='pubsub#send_last_published_item'><value>never</
            value></field>
        <field var='pubsub#presence_based_delivery'><value>0</value></
            field>
        <field var='pubsub#notification_type'><value>headline</value><
            /field>
        <field var='pubsub#type'><value>urn:example:e2ee:bundle</value
            ></field>
        <field var='pubsub#body_xslt'>
          <value>http://jabxslt.jabberstudio.org/atom_body.xslt</value
              >
        </field>
```

```
      </x>
    </configuration>
  </event>
</message>
```

8.3 Request Default Node Configuration Options

An entity may want to request information about the default node configuration, e.g. in order to determine whether to perform create-and-configure as previously described. Support for this feature is OPTIONAL.

8.3.1 Request

To get the node options, the entity MUST send an empty <default/> element to the service with no NodeID; in response, the service SHOULD return the default node options.

Listing 156: Entity requests default node configuration options

```
<iq type='get'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='def1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <default/>
  </pubsub>
</iq>
```

8.3.2 Success Case

If no error occurs, the service MUST return the default node configuration options.

Listing 157: Service responds with default node configuration options

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='def1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <default>
      <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='pubsub#title' type='text-single'
               label='A␣friendly␣name␣for␣the␣node'/>
```

```
<field var='pubsub#deliver_notifications' type='boolean'
       label='Deliver␣event␣notifications'>
  <value>true</value>
</field>
<field var='pubsub#deliver_payloads' type='boolean'
     label='Deliver␣payloads␣with␣event␣notifications'>
  <value>1</value>
</field>
<field var='pubsub#description' type='text-single'
     label='A␣description␣of␣the␣node'/>
<field var='pubsub#notify_config' type='boolean'
       label='Notify␣subscribers␣when␣the␣node␣configuration␣
          changes'>
  <value>0</value>
</field>
<field var='pubsub#notify_delete' type='boolean'
       label='Notify␣subscribers␣when␣the␣node␣is␣deleted'>
  <value>0</value>
</field>
<field var='pubsub#notify_retract' type='boolean'
       label='Notify␣subscribers␣when␣items␣are␣removed␣from␣
          the␣node'>
  <value>0</value>
</field>
<field var='pubsub#notify_sub' type='boolean'
       label='Notify␣owners␣about␣new␣subscribers␣and␣
          unsubscribes'>
  <value>0</value>
</field>
<field var='pubsub#persist_items' type='boolean'
       label='Persist␣items␣to␣storage'>
  <value>1</value>
</field>
<field var='pubsub#max_items' type='text-single'
       label='Max␣#␣of␣items␣to␣persist.␣‘max‘␣for␣no␣specific
          ␣limit␣other␣than␣a␣server␣imposed␣maximum.'>
  <value>10</value>
</field>
<field var='pubsub#item_expire' type='text-single'
       label='Time␣after␣which␣to␣automatically␣purge␣items.␣‘
          max‘␣for␣no␣specific␣limit␣other␣than␣a␣server␣
          imposed␣maximum.'>
  <value>604800</value>
</field>
<field var='pubsub#subscribe' type='boolean'
       label='Whether␣to␣allow␣subscriptions'>
  <value>1</value>
</field>
<field var='pubsub#access_model' type='list-single'
```

```
        label='Specify␣the␣subscriber␣model'>
    <option><value>authorize</value></option>
    <option><value>open</value></option>
    <option><value>presence</value></option>
    <option><value>roster</value></option>
    <option><value>whitelist</value></option>
    <value>open</value>
  </field>
  <field var='pubsub#roster_groups_allowed' type='list-multi'
        label='Roster␣groups␣allowed␣to␣subscribe'>
    <option><value>friends</value></option>
    <option><value>courtiers</value></option>
    <option><value>servants</value></option>
    <option><value>enemies</value></option>
  </field>
  <field var='pubsub#publish_model' type='list-single'
        label='Specify␣the␣publisher␣model'>
    <option><value>publishers</value></option>
    <option><value>subscribers</value></option>
    <option><value>open</value></option>
    <value>publishers</value>
  </field>
  <field var='pubsub#purge_offline' type='boolean'
        label='Purge␣all␣items␣when␣the␣relevant␣publisher␣goes
           ␣offline?'>
    <value>0</value>
  </field>
  <field var='pubsub#max_payload_size' type='text-single'
        label='Max␣payload␣size␣in␣bytes'>
    <value>9216</value>
  </field>
  <field var='pubsub#send_last_published_item' type='list-single
    '
        label='When␣to␣send␣the␣last␣published␣item'>
    <option label='Never'><value>never</value></option>
    <option label='When␣a␣new␣subscription␣is␣processed'><value>
       on_sub</value></option>
    <option label='When␣a␣new␣subscription␣is␣processed␣and␣
       whenever␣a␣subscriber␣comes␣online'>
      <value>on_sub_and_presence</value>
    </option>
    <value>never</value>
  </field>
  <field var='pubsub#presence_based_delivery' type='boolean'
        label='Deliver␣notifications␣only␣to␣available␣users'>
    <value>0</value>
  </field>
  <field var='pubsub#notification_type' type='list-single'
        label='Specify␣the␣delivery␣style␣for␣notifications'>
```

```
          <option><value>normal</value></option>
          <option><value>headline</value></option>
          <value>headline</value>
        </field>
      </x>
    </default>
  </pubsub>
</iq>
```

8.3.3 Error Cases

There are several reasons why the default node configuration options request might fail:

1. The service does not support node configuration.
2. The service does not support retrieval of default node configuration.

These error cases are described more fully in the following sections.
If the service does not support node configuration, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "config-node".

Listing 158: Service does not support node configuration

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='def1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='config-node'/>
  </error>
</iq>
```

If the service does not support retrieval of default node configuration options, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "retrieve-default".

Listing 159: Service does not support retrieval of default node configuration options

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='def1'>
  <error type='cancel'>
```

```
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='retrieve-default'/>
  </error>
</iq>
```

8.4 Delete a Node

If a service supports node creation, it MUST support node deletion. If an implementation persists items, it MUST remove all items from persistent storage before the node itself is deleted.

8.4.1 Request

In order to delete a node, a node owner MUST send a node deletion request, consisting of a <delete/> element whose 'node' attribute specifies the NodeID of the node to be deleted.

Listing 160: Owner deletes a node

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='delete1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <delete node='princely_musings'/>
  </pubsub>
</iq>
```

The deletion request MAY include the URI of a replacement node to which requests might be redirected. Typically this is an XMPP URI or IRI as described under PubSub URIs, but it can be an HTTP URI or any other scheme.

Listing 161: Owner deletes a node with redirection

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='delete1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <delete node='princely_musings'>
      <redirect uri='xmpp:hamlet@denmark.lit?;node=blog'/>
    </delete>
  </pubsub>
</iq>
```

Support for redirection is OPTIONAL on the part of pubsub services.

8.4.2 Success Case

If no error occurs, the service MUST inform the owner of success.

Listing 162: Service replies with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    id='delete1'/>
```

In addition, the service MUST also send notification of node deletion to all subscribers (which SHOULD include pending and unconfigured subscriptions).

Listing 163: Subscribers are notified of node deletion

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <delete node='princely_musings'>
      <redirect uri='xmpp:hamlet@denmark.lit?;node=blog'/>
    </delete>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='bernardo@denmark.lit' id='
    bar'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <delete node='princely_musings'>
      <redirect uri='xmpp:hamlet@denmark.lit?;node=blog'/>
    </delete>
  </event>
</message>
```

8.4.3 Error Cases

There are several reasons why the node deletion request might fail:

1. The requesting entity does not have sufficient privileges to delete the node.
2. The node is the root collection node, which cannot be deleted (see XEP-0248).
3. The specified node does not exist.

These error cases are described more fully in the following sections.
If the requesting entity does not have sufficient privileges to delete the node (e.g., is not an owner), the service MUST return a <forbidden/> error.

Listing 164: Entity is not an owner

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='delete1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requesting entity attempts to delete a node that does not exist, the service MUST return an <item-not-found/> error.

Listing 165: Owner attempts to delete a non-existent node

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='delete1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.5 Purge All Node Items

If a service persists published items, a node owner may want to purge the node of all published items (thus removing all items from the persistent store). It is OPTIONAL for a service to implement this feature.

8.5.1 Request

In order to purge a node of all items, a node owner sends a node purge request consisting of a <purge/> element whose 'node' attribute specifies the NodeID of the node to be purged.

Listing 166: Owner purges all items from a node

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
```

```
    id='purge1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <purge node='princely_musings'/>
  </pubsub>
</iq>
```

8.5.2 Success Case

If no error occurs, the service MUST purge the node and inform the owner of success.

Listing 167: Service replies with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    id='purge1'/>
```

If the node or service has been configured to notify subscribers on deletion of items, a purge request MUST NOT result in sending the same notifications as are sent when deleting items (since purging a node with many persisted items could result in a large number of notifications); instead, the node MUST send a single notification to each subscriber, containing an empty <purge/> child element.

Listing 168: Subscribers are notified of node purge

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit' id='
    foo'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <purge node='princely_musings'/>
  </event>
</message>

<message from='pubsub.shakespeare.lit' to='bernardo@denmark.lit' id='
    bar'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <purge node='princely_musings'/>
  </event>
</message>
```

8.5.3 Error Cases

There are several reasons why the node purge request might fail:

1. The node or service does not support node purging.
2. The requesting entity does not have sufficient privileges to purge the node.

3. The node is not configured to persist items.
4. The specified node does not exist.

These error cases are described more fully in the following sections.
If the node or service does not support node purging, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "purge-nodes".

Listing 169: Service does not support node purging

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='purge1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='purge-nodes'/>
  </error>
</iq>
```

If the requesting entity does not have sufficient privileges to purge the node (e.g., because it is not a node owner), the service MUST return a <forbidden/> error.

Listing 170: Entity is not an owner

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='purge1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the service or node does not persist items (e.g., because the node is a collection node as described in XEP-0248), it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "persistent-items".

Listing 171: Node is not configured for persistent items

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='purge1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
```

```
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='persistent-items'/>
  </error>
</iq>
```

If the node does not exist, the service MUST return an <item-not-found/> error.

Listing 172: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='purge1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.6 Manage Subscription Requests

A service MAY send subscription approval requests to the node owner(s) at any time. An approval request consists of a message stanza containing a Data Form scoped by the "http://jabber.org/protocol/pubsub#subscribe_authorization" FORM_TYPE. The form MUST contain a boolean field that has a 'var' attribute of "pubsub#allow", which is the field that designates whether or not to allow the subscription request. The form SHOULD include fields that specify the node identifier and the JID of the pending subscriber. The message MAY include a <body/> element that contains natural-language text explaining that the message contains a pending subscription form.

Listing 173: Service sends authorization request to node owner

```
<message to='hamlet@denmark.lit' from='pubsub.shakespeare.lit' id='
    approve1'>
  <x xmlns='jabber:x:data' type='form'>
    <title>PubSub subscriber request</title>
    <instructions>
      To approve this entity&apos;s subscription request,
      click the OK button. To deny the request, click the
      cancel button.
    </instructions>
    <field var='FORM_TYPE' type='hidden'>
      <value>http://jabber.org/protocol/pubsub#subscribe_authorization
          </value>
    </field>
    <field var='pubsub#subid' type='hidden'><value>123-abc</value></
        field>
    <field var='pubsub#node' type='text-single' label='Node␣ID'>
      <value>princely_musings</value>
```

```
    </field>
    <field var='pubsub#subscriber_jid' type='jid-single' label='
        Subscriber␣Address'>
      <value>horatio@denmark.lit</value>
    </field>
    <field var='pubsub#allow' type='boolean'
           label='Allow␣this␣JID␣to␣subscribe␣to␣this␣pubsub␣node?'>
      <value>false</value>
    </field>
  </x>
</message>
```

In order to approve the request, the owner shall submit the form and set the "pubsub#allow" field to a value of "1" or "true"; for tracking purposes the message MUST reflect the 'id' attribute originally provided.

Listing 174: Owner approves subscription request

```
<message from='hamlet@denmark.lit/elsinore' to='pubsub.shakespeare.lit
    ' id='approve1'>
  <x xmlns='jabber:x:data' type='submit'>
    <field var='FORM_TYPE' type='hidden'>
      <value>http://jabber.org/protocol/pubsub#subscribe_authorization
          </value>
    </field>
    <field var='pubsub#subid'>
      <value>123-abc</value>
    </field>
    <field var='pubsub#node'>
      <value>princely_musings</value>
    </field>
    <field var='pubsub#subscriber_jid'>
      <value>horatio@denmark.lit</value>
    </field>
    <field var='pubsub#allow'>
      <value>true</value>
    </field>
  </x>
</message>
```

The service then SHOULD notify the approved subscriber (see the Notification of Subscription State Changes section of this document).

Listing 175: Subscription approval notification

```
<message
    from='pubsub.shakespeare.lit'
    to='horatio@denmark.lit'
    id='approvalnotify1'>
```

```
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <subscription node='princely_musings' jid='horatio@denmark.lit'
        subscription='subscribed'/>
  </event>
</message>
```

In order to deny the request, the owner shall submit the form and set the "pubsub#allow" field to a value of "0" or "false"; as above, the message MUST reflect the 'id' attribute originally provided.

Listing 176: Owner denies subscription request

```
<message from='hamlet@denmark.lit/elsinore' to='pubsub.shakespeare.lit
    ' id='approve1'>
  <x xmlns='jabber:x:data' type='submit'>
    <field var='FORM_TYPE' type='hidden'>
      <value>http://jabber.org/protocol/pubsub#subscribe_authorization
          </value>
    </field>
    <field var='pubsub#subid'>
      <value>123-abc</value>
    </field>
    <field var='pubsub#node'>
      <value>princely_musings</value>
    </field>
    <field var='pubsub#subscriber_jid'>
      <value>horatio@denmark.lit</value>
    </field>
    <field var='pubsub#allow'>
        <value>false</value>
    </field>
  </x>
</message>
```

The service then SHOULD notify the denied subscriber (see the Notification of Subscription State Changes section of this document).

Listing 177: Subscription cancellation / denial notification

```
<message
    from='pubsub.shakespeare.lit'
    to='horatio@denmark.lit'
    id='unsubnotify1'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <subscription node='princely_musings' jid='horatio@denmark.lit'
        subscription='none'/>
  </event>
</message>
```

In order to cancel the form submission, the owner shall reply with the form's 'type' attribute set to "cancel".

Listing 178: Owner cancels form submission

```
<message from='hamlet@denmark.lit/elsinore' to='pubsub.shakespeare.lit
    ' id='approve1'>
  <x xmlns='jabber:x:data' type='cancel'>
    <field var='FORM_TYPE' type='hidden'>
      <value>http://jabber.org/protocol/pubsub#subscribe_authorization
          </value>
    </field>
  </x>
</message>
```

The service MUST check the "pubsub#allow" field to see if the subscription should be allowed or denied. If the owner cancels the Data Form, then the subscription request MUST remain in the pending state.

8.7 Process Pending Subscription Requests

A node owner may want to request all of the pending subscription requests for all of their nodes at a service. It is OPTIONAL for a service to implement this feature.
This feature MUST be implemented using the Ad-Hoc Commands (XEP-0050) [18] protocol, where the command name ('node' attribute of the command element) MUST have a value of "http://jabber.org/protocol/pubsub#get-pending".

8.7.1 Request

Listing 179: Owner requests pending subscription requests

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='pending1'>
  <command xmlns='http://jabber.org/protocol/commands'
           node='http://jabber.org/protocol/pubsub#get-pending'
           action='execute'/>
</iq>
```

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

8.7.2 Success Case

If no error occurs, the service SHOULD return a data form for managing subscription requests, which MUST contain a single field with a 'var' attribute value of "pubsub#node" whose <option/> elements specify the nodes for which the requesting entity has subscription approval privileges (as an optimization, the service MAY specify only the nodes that have subscription requests pending).

Listing 180: Service responds with data form to be populated

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='pending1'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='pubsub-get-pending:20031021T150901Z-600'
           node='http://jabber.org/protocol/pubsub#get-pending'
           status='executing'
           action='execute'>
    <x xmlns='jabber:x:data' type='form'>
      <field var='FORM_TYPE' type='hidden'>
        <value>http://jabber.org/protocol/pubsub#
            subscribe_authorization</value>
      </field>
      <field type='list-single' var='pubsub#node'>
        <option><value>princely_musings</value></option>
        <option><value>news_from_elsinore</value></option>
      </field>
    </x>
  </command>
</iq>
```

8.7.3 Error Cases

There are several reasons why the pending subscription approval request might fail:

1. The service does not support the ad-hoc commands protocol.
2. The service supports ad-hoc commands but does not support the "get-pending" feature.
3. The requesting entity does not have sufficient privileges to approve subscription requests.
4. The specified node does not exist.

These error cases are described more fully in the following sections.
If the service does not support the ad-hoc commands protocol, it MUST respond with a

<service-unavailable/> error.

Listing 181: Service responds with node not found

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='pending1'>
  <error type='cancel'>
    <service-unavailable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the service does not support the "get-pending" feature, it MUST respond with a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "get-pending".

Listing 182: Service responds with node not found

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='pending1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='get-pending'/>
  </error>
</iq>
```

If the requesting entity does not have sufficient privileges to approve subscription requests, the service MUST respond with a <forbidden/> error.

Listing 183: Entity does not have sufficient privileges to approve subscription requests

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='pending1'>
  <error type='cancel'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requested node does not exist, the service MUST respond with an <item-not-found/> error.

Listing 184: Service responds with node not found

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='pending1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.7.4 Per-Node Request

Upon receiving the data form for managing subscription requests, the owner then MAY request pending subscription approval requests for a given node.

Listing 185: Owner requests all pending subscription requests for a node

```
<iq type='set' to='pubsub.shakespeare.lit' id='pending2'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='pubsub-get-pending:20031021T150901Z-600'
           node='http://jabber.org/protocol/pubsub#get-pending'
           action='execute'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='pubsub#node'>
        <value>princely_musings</value>
      </field>
    </x>
  </command>
</iq>
```

If no error occurs, the service shall respond with success.

Listing 186: Service responds with success

```
<iq from='pubsub.shakespeare.lit'
    id='pending2'
    to='hamlet@denmark.lit/elsinore'
    type='result'>
  <command xmlns='http://jabber.org/protocol/commands'
           sessionid='pubsub-get-pending:20031021T150901Z-600'
           node='http://jabber.org/protocol/pubsub#get-pending'
           action='completed'/>
</iq>
```

The service shall then send one subscription approval message for each pending subscription request, as shown above for a single pending subscription request.
Note: A service SHOULD conform to its affiliation policies in maintaining the list of pending

subscriptions. In particular, if the affiliation of an entity with a pending subscription is modified to owner or publisher, the service SHOULD automatically approve the subscription request and remove the entity's previous request from the pending list. Similarly, if the affiliation of an entity with a pending subscription is modified to outcast, the service SHOULD automatically reject the subscription request and remove the entity's previous request from the pending list. (If an entity's subscription request is denied, the service SHOULD send a <message/> to the entity, where the message conforms to the format described in the Notification of Subscription State Changes section of this document.)

8.8 Manage Subscriptions

A node owner may want to edit the list of subscriptions associated with a given node. Support for this feature ("pubsub#manage-subscriptions") is OPTIONAL.

8.8.1 Retrieve Subscriptions List

First the owner retrieves the subscriptions list.
In order to request a list of all subscriptions, a node owner MUST send a subscriptions request, consisting of a <subscriptions/> element whose 'node' attribute specifies the NodeID of the relevant node.

Listing 187: Owner requests all subscriptions

```
<iq type='get'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='subman1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <subscriptions node='princely_musings'/>
  </pubsub>
</iq>
```

If no error occurs, the service MUST return the list of subscriptions for entities whose subscription state is "subscribed" or "unconfigured" (it MUST NOT return entities whose subscription state is "none" and SHOULD NOT return entities whose subscription state is "pending"). The result MAY specify multiple <subscription/> elements for the same entity (JID), but each element MUST possess a distinct value of the 'subid' attribute (as shown below).

Listing 188: Service returns list of subscriptions

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='subman1'>
```

```
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <subscriptions node='princely_musings'>
      <subscription jid='hamlet@denmark.lit' subscription='subscribed'
          />
      <subscription jid='polonius@denmark.lit' subscription='
          unconfigured'/>
      <subscription jid='bernardo@denmark.lit' subscription='
          subscribed' subid='123-abc'/>
      <subscription jid='bernardo@denmark.lit' subscription='
          subscribed' subid='004-yyy'/>
    </subscriptions>
  </pubsub>
</iq>
```

There are several reasons why the manage subscriptions request might fail:

1. The service does not support subscription management.
2. The requesting entity does not have sufficient privileges to manage subscriptions.
3. The specified node does not exist.

These error cases are described more fully in the following sections.
If an implementation does not support subscription management, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "manage-subscriptions".

Listing 189: Node or service does not support subscription management

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='subman1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='manage-subscriptions'/>
  </error>
</iq>
```

If the requesting entity is not a node owner, the service MUST return a <forbidden/> error.

Listing 190: Entity is not an owner

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='subman1'>
  <error type='auth'>
```

```
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node does not exist, the service MUST return an <item-not-found/> error.

Listing 191: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='subman1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.8.2 Modify Subscriptions

Upon receiving the subscriptions list, the node owner MAY modify subscription states. The owner MUST send only modified subscription states (i.e., a "delta"), not the complete list. (Note: If the 'subscription' attribute is not specified in a modification request, then the value MUST NOT be changed.)

Listing 192: Owner modifies subscriptions

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='subman2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <subscriptions node='princely_musings'>
      <subscription jid='bard@shakespeare.lit' subscription='
          subscribed'/>
    </subscriptions>
  </pubsub>
</iq>
```

Listing 193: Service responds with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    id='subman2'/>
```

There are several reasons why the modify subscriptions request might fail:

1. The service does not support subscription management.

2. The requesting entity does not have sufficient privileges to manage subscriptions.
3. The specified node does not exist.

These error cases are described more fully in the following sections.
If an implementation does not support subscription management, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "manage-subscriptions".

Listing 194: Node or service does not support subscription management

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='subman2'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='manage-subscriptions'/>
  </error>
</iq>
```

If the requesting entity is not a node owner, the service MUST return a <forbidden/> error.

Listing 195: Entity is not an owner

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='subman1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node does not exist, the service MUST return an <item-not-found/> error.

Listing 196: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='subman2'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

The owner MAY change multiple subscriptions in a single request. If one of the <subscription/> elements specified is invalid, the service MUST return an IQ error (which SHOULD be <not-acceptable/>) with the invalid entries, where the subscription returned is the original,

un-altered subscription.

Listing 197: Owner sets subscription for multiple entities

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='subman3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <subscriptions node='princely_musings'>
      <subscription jid='polonius@denmark.lit' subscription='none'/>
      <subscription jid='bard@shakespeare.lit' subscription='
          subscribed'/>
    </subscriptions>
  </pubsub>
</iq>
```

Listing 198: Service responds with an error

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='subman3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <subscriptions node='princely_musings'>
      <subscription jid='polonius@denmark.lit' subscription='
          subscribed'/>
    </subscriptions>
  </pubsub>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If errors occur during a modification request for multiple entities, the pubsub service MUST return any <subscription/> element(s) which caused the error. Returned entities which failed to be modified MUST include the existing 'subscription' attribute. Any <subscription/> elements which are not returned in an IQ error case MUST be treated as successful modifications. The owner MAY specify multiple <subscription/> elements for the same entity, but each element MUST possess a distinct value of the 'subid' attribute.

8.8.3 Delete a Subscriber

In order to remove an entity from the subscriptions list, the owner MUST set the value of the 'subscription' attribute to "none" and the service MUST remove that entity from the subscriptions list and not return it in response to future list requests.

8.8.4 Notifying Subscribers

An implementation SHOULD notify an entity whose subscription has changed (see the Notification of Subscription State Changes section of this document).

Listing 199: Service sends notification of subscription change

```
<message from='pubsub.shakespeare.lit' to='polonius@denmark.lit'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <subscription node='princely_musings' jid='polonius@denmark.lit'
        subscription='none'/>
  </event>
</message>
```

8.9 Manage Affiliations

A node owner may want to manage the affiliations of entities associated with a given node and to set affiliations for new entities. Support for this feature ("pubsub#modify-affiliations") is OPTIONAL.

8.9.1 Retrieve Affiliations List

First the owner retrieves the affiliation list.
In order to request a list of all affiliated entities, a node owner MUST send an affiliations request, consisting of an <affiliations/> element whose 'node' attribute specifies the NodeID of the relevant node.

Listing 200: Owner requests all affiliated entities

```
<iq type='get'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='ent1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <affiliations node='princely_musings'/>
  </pubsub>
</iq>
```

If no error occurs, the service MUST return the list of entities whose affiliation is "owner", "member", "publisher", "publish-only", or "outcast" (it MUST NOT return entities whose affiliation is "none").

Listing 201: Service returns list of affiliated entities

```
<iq type='result'
```

```
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='ent1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <affiliations node='princely_musings'>
      <affiliation jid='hamlet@denmark.lit' affiliation='owner'/>
      <affiliation jid='polonius@denmark.lit' affiliation='outcast'/>
    </affiliations>
  </pubsub>
</iq>
```

There are several reasons why the affiliated entities request might fail:

1. The service does not support modification of affiliations.
2. The requesting entity does not have sufficient privileges to modify affiliations.
3. The specified node does not exist.

These error cases are described more fully in the following sections.
If an implementation does not support modification of affiliations, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "modify-affiliations".

Listing 202: Node or service does not support affiliation management

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='ent1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='modify-affiliations'/>
  </error>
</iq>
```

If the requesting entity is not a node owner, the service MUST return a <forbidden/> error.

Listing 203: Entity is not an owner

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='ent1'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node does not exist, the service MUST return an <item-not-found/> error.

Listing 204: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='ent1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.9.2 Modify Affiliation

A node owner may want to edit the affiliation of an entity associated with a given node or to set the affiliation for a new entity.
In order to modify an affiliation, a node owner MUST send an IQ set containing the modified affiliation or affiliations. The owner MUST send only modified affiliations (i.e., a "delta"), not the complete list. (Note: If the 'affiliation' attribute is not specified in a modification request, then the value MUST NOT be changed.)

Listing 205: Owner modifies affiliation

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='ent2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <affiliations node='princely_musings'>
      <affiliation jid='bard@shakespeare.lit' affiliation='publisher'/
        >
    </affiliations>
  </pubsub>
</iq>
```

Listing 206: Service responds with success

```
<iq type='result'
    from='pubsub.shakespeare.lit'
    id='ent2'/>
```

There are several reasons why the modify-affiliations request might fail:

1. The requested affiliation is not supported by the node or service.
2. The service does not support modification of affiliations.

3. The requesting entity does not have sufficient privileges to modify affiliations.

4. The specified node does not exist.

These error cases are described more fully in the following sections.
If an implementation does not support modification of affiliations, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "modify-affiliations".

Listing 207: Node or service does not support affiliation management

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='ent1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='modify-affiliations'/>
  </error>
</iq>
```

If the node or service does not support the requested affiliation, it MUST return a <feature-not-implemented/> error, specifying a pubsub-specific error condition of <unsupported/> and a feature of "member-affiliation", "outcast-affiliation", "publisher-affiliation", or "publish-only-affiliation" as appropriate.

Listing 208: Node or service does not support the requested affiliation

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='ent1'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <unsupported xmlns='http://jabber.org/protocol/pubsub#errors'
                 feature='member-affiliation'/>
  </error>
</iq>
```

If the requesting entity is not a node owner, the service MUST return a <forbidden/> error.

Listing 209: Entity is not an owner

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='ent1'>
  <error type='auth'>
```

```
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the node does not exist, the service MUST return an <item-not-found/> error.

Listing 210: Node does not exist

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    id='ent1'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

The owner MAY change multiple affiliations in a single request. If one of the <subscription/> elements specified is invalid, the service MUST return an IQ error (which SHOULD be <not-acceptable/>) with the invalid entries, where the affiliation returned is the original, un-altered affiliation.
The following example shows an entity attempting to make the owner something other than an affiliation of "owner", an action which MUST NOT be allowed if there is only one owner.

Listing 211: Owner sets affiliation for multiple entities

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='ent3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <affiliations node='princely_musings'>
      <affiliation jid='hamlet@denmark.lit' affiliation='none'/>
      <affiliation jid='polonius@denmark.lit' affiliation='none'/>
      <affiliation jid='bard@shakespeare.lit' affiliation='publisher'/
        >
    </affiliations>
  </pubsub>
</iq>
```

Listing 212: Service responds with an error

```
<iq type='error'
    from='pubsub.shakespeare.lit'
    to='hamlet@denmark.lit/elsinore'
    id='ent3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub#owner'>
    <affiliations node='princely_musings'>
      <affiliation jid='hamlet@denmark.lit' affiliation='owner'/>
    </affiliations>
```

```
  </pubsub>
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

The state chart at the beginning of this document is a MUST-IMPLEMENT set of rules for checking possible state transitions. Implementations MAY enforce other (more strict) rules. If errors occur during a modification request for multiple entities, the pubsub service MUST return any <affiliation/> element(s) which caused the error. Returned entities which failed to be modified MUST include the existing 'affiliation' attribute. Any <subscription/> elements which are not returned in an IQ error case MUST be treated as successful modifications. The owner MUST NOT specify multiple <affiliation/> elements for the same entity; otherwise the service MUST return a <bad-request/> error.

8.9.3 Delete an Entity

In order to remove an entity from the affiliations list, the owner MUST set the value of the 'affiliation' attribute to "none" and the service MUST remove that entity from the affiliations list and not return it in response to future list requests.

8.9.4 Notifying Entities

An implementation MAY send an event notification to an entity whose affiliation has changed, which MAY contain a <body/> element specifying natural-language text regarding the affiliation change and which SHOULD contain the modified affiliation data.

Listing 213: Service sends notification of affiliation change

```
<message from='pubsub.shakespeare.lit' to='polonius@denmark.lit'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <affiliations node='princely_musings'>
      <affilation jid='polonius@denmark.lit' affiliation='none'/>
    </affiliations>
  </pubsub>
</message>
```

9 IM Account Integration

Publish-subscribe functionality can be integrated into existing instant messaging and presence services (see RFC 3921), such that each registered account functions as a virtual pubsub service (sometimes called "pubsub-on-a-JID"). In such deployments, the root pubsub node for each virtual pubsub service has the same address as the bare JID (<localpart@domain.tld>

or <domain.tld>) of the account, which is typically associated with an IM user (e.g., <hamlet@denmark.lit>). Since an IM user typically has a roster of "buddies" and shares presence information with those buddies, the virtual pubsub service can use roster and presence information to provide some helpful shortcuts for subscribers, in particular the auto-subscribe and filtered-notifications features described in this section.
Note: PEP ties the receipt of PEP notifications to the subscriber's presence, but does not tie the generation of PEP notifications to the publisher's presence. If the publisher wishes to stop generating PEP events (or to generate an "empty" event as can be done for some PEP payloads) before ending its presence session, the publisher MUST direct its client to do so and MUST NOT depend on the PEP service to automatically "zero out" its PEP information when the PEP service receives unavailable presence from the publisher.
If an instant messaging and presence account is also a virtual pubsub service, service discovery information ("disco#info") responses from the bare JID of the account MUST include a feature of "http:/jabber.org/protocol/pubsub#pubsub-on-a-jid":

Listing 214: IM server returns supported features on behalf of IM account

```
<iq from='hamlet@denmark.lit'
    id='bvg194j7'
    to='francisco@denmark.lit/barracks'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='account' type='registered'/>
    <feature var='http://jabber.org/protocol/pubsub#pubsub-on-a-jid'/>
  </query>
</iq>
```

Note: Because the account owner's bare JID is the default destination address for any stanzas a client generates, clients often omit the "to" attribute on such stanzas; on this point, see RFC 6120 and (with regard to rosters) RFC 6121.

9.1 Auto-Subscribe

When a contact is affiliated with the account owner through sharing of XMPP presence, the "auto-subscribe" feature greatly simplifies the subscription process. In particular, support for the "auto-subscribe" has the following implications:

9.1.1 Account Owner

Because the account owner itself is implicitly subscribed to its own XMPP presence (e.g., each XMPP resource receives presence information from all of the account owner's resources), a service MUST consider the account owner to have a pubsub subscription to the account owner's root collection node with a subscription_type of "items" and a subscription_depth of

”all”. This is true for all access models.

9.1.2 Presence Subscriber

If an entity (i.e., an IM contact) has an XMPP presence subscription to the account owner’s bare JID (<localpart@domain.tld> or <domain.tld>), a service MUST consider the contact to have a pubsub subscription to the account owner’s root collection node with a subscription_type of ”items” and a subscription_depth of ”all” if:

1. The node has an access model of ”open”.
2. The node has an access model of ”presence”.
3. The node has an access model of ”roster” and the contact is in the specified roster group.

If the contact does not have permission to receive information from any of the account owner’s particular nodes below the level of the root collection node (e.g., because a particular node has an access model of ”roster” but the contact is not in the specified roster group), the service MUST NOT send notifications regarding that node to the contact and also MUST NOT return any errors to the contact regarding a potential implicit subscription to that node (e.g., the service MUST NOT return a pubsub subscription error to the contact when the contact sends presence to the account owner).
Note: When an IM contact has a subscription to the account owner’s presence, the automated pubsub subscription MUST be based on the JID contained in the ’from’ address of the presence subscription request, which for an IM contact will be a bare JID (<localpart@domain.tld> or <domain.tld>).

9.1.3 Presence Sharer

If the node has an open access model, the pubsub service SHOULD also consider an entity to be temporarily and implicitly subscribed to the node if the entity has sent presence to the account owner in the absence of a presence subscription. In this case, the subscription SHOULD be based on the ’from’ address of the presence stanza, which will be a full JID (<localpart@domain.tld/resource> or <domain.tld/resource>). When the service receives unavailable presence from the full JID, it MUST cancel the temporary subscription.

9.2 Filtered Notifications

A contact might not want to receive notifications for all the nodes hosted at a user’s virtual pubsub service. A contact SHOULD signal its preferences to the account owner’s server by including XEP-0115 information that specifies the NodeIDs for which the contact wishes to receive notifications (if any). This information is used by a pubsub service that supports the

”filtered-notifications” feature to send notifications only from those NodeIDs that match the subscriber’s preferences.
In order to make this possible, all possible NodeIDs can be appended with the string ”+notify” to indicate that the contact wishes to receive notifications for the specified NodeID. Thus if Romeo wants to receive notifications for location data (User Geolocation (XEP-0080) [19]) and tune data (User Tune (XEP-0118) [20]) but not activity data (User Activity (XEP-0108) [21]), his client would advertise support for the following strings in the disco#info results it sends: [22]

- http://jabber.org/protocol/geoloc+notify
- http://jabber.org/protocol/tune+notify

This set of strings would then be advertised by including them in the identity+features hash encapsulated via the ’ver’ attribute as described in XEP-0115.

Listing 215: Contact sends presence with caps

```
<presence from='romeo@montague.lit/orchard'>
  <c xmlns='http://jabber.org/protocol/caps'
     node='http://www.chatopus.com/#2.2'
     ver='AFBT0mPr29zQE5aGtCJp97CIS6E='/>
</presence>
```

It is the responsibility of the account owner’s server to cache XEP-0115 information. When the server receives presence from a contact, it MUST check that presence information for entity capabilities data and correlate that data with the desired NodeIDs for the contact’s client. The server MUST NOT send notifications related to any NodeIDs that the contact’s client has not asked for via the relevant ”NodeID+notify” disco#info feature. This enables a client to turn off all notifications (e.g., because of bandwidth restrictions) and to easily receive all desired data formats simply by adding support for the appropriate ”NodeID+notify” combination in its disco#info results and client capabilities. However, it also implies that a client can request notifications only on a global basis and cannot request, say, mood information only from certain contacts in the user’s roster. Community consensus is that this is an acceptable tradeoff. Also, note that this works only if the account owner has a presence subscription to the contact and the contact has a presence subscription to the account owner.
Some examples may help to illustrate the concept of notification filtering. Here we show presence generated by two of the contacts listed above (benvolio@montague.lit does not have any presence subscriptions to or from juliet@capulet.lit and therefore is not involved in these

[19] XEP-0080: User Geolocation <https://xmpp.org/extensions/xep-0080.html>.
[20] XEP-0118: User Tune <https://xmpp.org/extensions/xep-0118.html>.
[21] XEP-0108: User Activity <https://xmpp.org/extensions/xep-0108.html>.
[22] Including, say, the ’http://jabber.org/protocol/geoloc’ NodeID indicates that the client understands the geolocation namespace described in XEP-0080, whereas including the ’http://jabber.org/protocol/geoloc+notify’ namespace indicates that the client wishes to receive notifications related to geolocation, where the NodeID is the same as the geolocation namespace ’http://jabber.org/protocol/geoloc’ (in this case there is a one-to-one correspondence between the namespace name and the NodeID).

protocol flows).

Listing 216: Presence with caps

```
<presence from='nurse@capulet.lit/chamber'>
  <c xmlns='http://jabber.org/protocol/caps'
     node='http://exodus.jabberstudio.org/#0.9.1'
     ver='wXj6c5xhT9frdqhvTSjkdejUUP8='/>
</presence>

<presence from='romeo@montague.lit/orchard'>
  <c xmlns='http://jabber.org/protocol/caps'
     node='http://www.chatopus.com/#2.2'
     ver='1FDrLLbYMpzvcI95jgSHABSWDRY='/>
</presence>
```

We assume that Juliet's server doesn't know anything about these capabilities, so it sends service discovery information requests to each of the clients on Juliet's behalf (realistically, the capulet.lit server will quickly build up a cache of client capabilities, with the result that it will not need to send these service discovery requests):

Listing 217: Account server queries contact

```
<iq from='juliet@capulet.lit'
    to='nurse@capulet.lit/chamber'
    type='get'
    id='disco123'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq from='nurse@capulet.lit/chamber'
    to='juliet@capulet.lit'
    type='result'
    id='disco123'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='client' type='pc'/>
    <feature var='http://jabber.org/protocol/activity'/>
    <feature var='http://jabber.org/protocol/activity+notify'/>
    <feature var='http://jabber.org/protocol/geoloc'/>
    <feature var='http://jabber.org/protocol/geoloc+notify'/>
    <feature var='http://jabber.org/protocol/muc'/>
    <feature var='http://jabber.org/protocol/tune'/>
    <feature var='http://jabber.org/protocol/tune+notify'/>
  </query>
</iq>
```

The server shall also query the identity+features for <romeo@montague.lit>:

Listing 218: Account server queries contact

```
<iq from='juliet@capulet.lit'
    to='romeo@montague.lit/orchard'
    type='get'
    id='disco234'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq from='romeo@montague.lit/orchard'
    to='juliet@capulet.lit'
    type='result'
    id='disco234'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='client' type='pda'/>
    <feature var='http://jabber.org/protocol/geoloc'/>
    <feature var='http://jabber.org/protocol/geoloc+notify'/>
    <feature var='http://jabber.org/protocol/tune'/>
    <feature var='http://jabber.org/protocol/tune+notify'/>
  </query>
</iq>
```

(As noted in XEP-0115, the server MUST check the hash provided in the 'ver' attribute against the generation method to ensure that no poisoning has occurred.)
Now we revisit account owner publication and server generation of notifications, with filtering enabled because the server has caps information:

- If Juliet publishes a tune item to the presence-access "http://jabber.org/protocol/tune" node, her server will send notifications to <nurse@capulet.lit/chamber> and <romeo@montague.lit/orchard> (full JIDs).

- If Juliet publishes an activity item to the presence-access "http://jabber.org/protocol/activity" node, her server will send notifications only to <nurse@capulet.lit/chamber>.

- If Juliet publishes a geolocation item to the roster-access "http://jabber.org/protocol/geoloc" node with the "pubsub#roster_groups_allowed" variable set to a value of "Friends", her server will send notifications only to <romeo@montague.lit/orchard> because the nurse is not in that roster group.

10 Feature Summary

This section summarizes the features described herein, specifies the appropriate requirements level for each feature (REQUIRED, RECOMMENDED, or OPTIONAL), and provides

cross-references to the section of this document in which each feature is described.
Note: The feature names are all of the form ”http://jabber.org/protocol/pubsub#name”, where ”name” is the text specified in the first column below.

Name	Description	Support	Section
access-authorize	The default access model is ”authorize”.	OPTIONAL	Nodes Access Models
access-open	The default access model is ”open”.	OPTIONAL	Nodes Access Models
access-presence	The default access model is ”presence”.	OPTIONAL	Nodes Access Models
access-roster	The default access model is ”roster”.	OPTIONAL	Nodes Access Models
access-whitelist	The default access model is ”whitelist”.	OPTIONAL	Nodes Access Models
auto-create	The service supports auto-creation of nodes on publish to a non-existent node.	OPTIONAL	Automatic Node Creation
auto-subscribe	The service supports auto-subscription to a nodes based on presence subscription.	RECOMMENDED	Auto-Subscribe
collections	Collection nodes are supported.	OPTIONAL	Refer to XEP-0248
config-node	Configuration of node options is supported.	RECOMMENDED	Configure a Node
create-and-configure	Simultaneous creation and configuration of nodes is supported.	RECOMMENDED	Create and Configure a Node
create-nodes	Creation of nodes is supported.	RECOMMENDED	Create a Node
delete-items	Deletion of items is supported.	RECOMMENDED	Delete an Item from a Node
delete-nodes	Deletion of nodes is supported.	RECOMMENDED	Delete a Node

Name	Description	Support	Section
filtered-notifications	Notifications are filtered based on Entity Capabilities data.	RECOMMENDED	Filtered Notifications
get-pending	Retrieval of pending subscription approvals is supported.	OPTIONAL	Manage Subscription Requests
instant-nodes	Creation of instant nodes is supported.	RECOMMENDED	Create a Node
item-ids	Publishers may specify item identifiers.	RECOMMENDED	
last-published	By default the last published item is sent to new subscribers and on receipt of available presence from existing subscribers.	RECOMMENDED	Event Types
leased-subscription	Time-based subscriptions are supported.	OPTIONAL	Time-Based Subscriptions (Leases)
manage-subscriptions	Node owners may manage subscriptions.	OPTIONAL	Manage Subscriptions
member-affiliation	The member affiliation is supported.	RECOMMENDED	Affiliations
meta-data	Node metadata is supported.	RECOMMENDED	
modify-affiliations	Node owners may modify affiliations.	OPTIONAL	Manage Affiliations
multi-collection	A single leaf node can be associated with multiple collections.	OPTIONAL	Refer to XEP-0248

Name	Description	Support	Section
multi-items	The service supports the storage of multiple items per node. It requires the pubsub#max_items configuration item to be exposed to the user and allows sensible values (higher than one) to be set in Configure a Node.	OPTIONAL	
multi-subscribe	A single entity may subscribe to a node multiple times.	OPTIONAL	Multiple Subscriptions
outcast-affiliation	The outcast affiliation is supported.	RECOMMENDED	Affiliations
persistent-items	Persistent items are supported.	RECOMMENDED	
presence-notifications	Presence-based delivery of event notifications is supported.	OPTIONAL	
presence-subscribe	Authorized contacts are automatically subscribed to a user's virtual pubsub service.	RECOMMENDED	Auto-Subscribe
publish	Publishing items is supported.	REQUIRED	Publish an Item to a Node
publish-options	Publishing an item with options is supported.	OPTIONAL	Publishing Options
publish-only-affiliation	The publish-only affiliation is supported.	OPTIONAL	Affiliations
publisher-affiliation	The publisher affiliation is supported.	RECOMMENDED	Affiliations

Name	Description	Support	Section
publish-node-full	Specified behaviour may happen when publishing on a full node.	OPTIONAL	Retracting an item and Rejecting an item.
purge-nodes	Purging of nodes is supported.	OPTIONAL	Purge All Node Items
retract-items	Item retraction is supported.	OPTIONAL	Delete an Item from a Node
retrieve-affiliations	Retrieval of current affiliations is supported.	RECOMMENDED	Retrieve Affiliations
retrieve-default	Retrieval of default node configuration is supported.	RECOMMENDED	Request Default Node Configuration Options
retrieve-default-sub	Retrieval of default subscription configuration is supported.	OPTIONAL	Request Default Subscription Configuration Options
retrieve-items	Item retrieval is supported.	RECOMMENDED	Retrieve Items from a Node
retrieve-subscriptions	Retrieval of current subscriptions is supported.	RECOMMENDED	Retrieve Subscriptions
subscribe	Subscribing and unsubscribing are supported.	REQUIRED	Subscribe to a Node and Unsubscribe from a Node
subscription-options	Configuration of subscription options is supported.	OPTIONAL	Configure Subscription Options
subscription-notifications	Notification of subscription state changes is supported.	OPTIONAL	Notification of Subscription State Changes

11 Error Conditions

Condition	Description
<conflict/>	The node already exists, or is full. See pubsub#publish_node_full.

Condition	Description
<feature-not-implemented/>	The operation being attempted on a node (or the system) has failed because the service or node does not support the operation; the error SHOULD also specify which feature is unsupported.
<forbidden/>	An entity does not have sufficient privileges to perform the action, is requesting an operation for another Jabber ID (e.g., francisco@denmark.lit attempts to subscribe bernardo@denmark.lit to a node), or the requesting entity has an affiliation of "outcast".
<item-not-found/>	The node or item specified for some operation does not exist.
<not-allowed/>	An entity has attempted to perform an action which the service implements; however the service-wide admin or the node owner has disabled the action for that service or node.
<not-authorized/>	An entity has attempted to subscribe to or retrieve items from a node but is not authorized to see the account owner's presence, is not in the appropriate roster group, or is not on the whitelist for subscriptions.
<payment-required/>	Subscriptions and item retrieval are based on some kind payment service. Payments would be done out-of-band using some agreed-upon method (not defined herein).
<registration-required/>	Entities are required to register before node creation is allowed.

Note: Refer to Error Condition Mappings (XEP-0086) [23] for more information regarding error syntax.

12 Implementation Notes

12.1 Data Model

When implementing this protocol, it is often helpful to consider the data model behind it. Note that this section describes such a conceptual model for informational purposes only and does not define any requirements about how, where or in what format an implementation actually stores data.
A persistent pubsub node may be viewed as an ordered key-value store, where the ItemIDs are the keys, and the item payloads are the values.
The order of items is publication order, such that the most recently published is the "newest". This holds true even if publishing with an ItemID that already exists on the node: the older item is dropped to preserve uniqueness of keys. This is sometimes referred to as "overwriting"

[23] XEP-0086: Error Condition Mappings <https://xmpp.org/extensions/xep-0086.html>.

in this document.
The order is of most importance when the max_items option is used to limit the number of items on the node. When the limit is reached, the oldest item may be dropped from the node according to the configured policy (see the publish_node_full option).
If published ItemIDs are always unique, a persistent node may also be viewed as a kind of "first in, first out" (FIFO) queue.

12.2 Notification Triggers

There are many possible triggers for sending an event notification to an entity for the currently published item or the last published item, as summarized below:

1. The entity explicitly requests one or more items from the node and is authorized to retrieve items; when the service receives such a request, it sends the items to the entity.
2. The entity is an authorized subscriber to the node (explicitly via subscription or implicitly based on a role of owner or publisher); when the publisher sends a publish request, the service sends the currently published item to the entity (subject to presence checks and notification filtering if appropriate).
3. The entity is not subscribed but is eligible to do so and has sent presence containing appropriate entity capabilities data to a service that supports filtered notifications (effectively establishing a "temporary subscription" based on an expressed notification interest); when the service first receives such presence, it sends the last published item to the entity (sending it only once upon first receiving such presence, not on subsequent presence updates that contain the same notification interest).
4. The entity is not subscribed but is eligible to do so and has sent presence containing appropriate entity capabilities data to a service that supports filtered notifications (effectively establishing a "temporary subscription"); when the publisher sends a publish request that matches the entity's expressed notification interest, the service sends the currently published item to the entity.
5. The entity gains access to the node because of a change to the node access model; as a result, the service sends the last published item to the entity.
6. The entity is added to the roster group associated with a node access model of "roster"; as a result, the service sends the last published item to the entity.

12.3 Intended Recipients for Notifications

When a pubsub service generates notifications, it MUST adhere to the delivery rules implicit in the subscription option configuration for each subscriber. In particular, the 'to' address SHOULD be that of the subscribed JID only. The service SHOULD NOT attempt to guess at

the most available resource associated with the subscribed JID (e.g., in the context of instant messaging systems).

12.4 Handling Notification-Related Errors

As noted above, a pubsub service SHOULD ensure that the <message/> stanza for each event notification it generates possesses an 'id' attribute with a value. (This notification ID is not to be confused with either the node ID or the item ID.) This ID MUST be unique within the context of the pubsub service in order to ensure proper tracking of any delivery-related errors.
Exactly how a service shall handle delivery-related errors is a matter of implementation. In general, such handling is effectively similar to the bounce processing performed by other message delivery systems, such as mail transfer agents and mailing list software. The following are some suggested guidelines regarding the handling of XMPP-specific error conditions in relation to pubsub event notifications (see RFC 3920 and XEP-0086 regarding XMPP error condition semantics):

- If the XMPP error is of type "cancel" (e.g., <item-not-found/>), or the error condition is <gone/>, the pubsub service SHOULD terminate the subscription of the entity to that node and MAY terminate the subscription of that entity to all nodes hosted at the service.
- If the XMPP error is of type "auth" (e.g., <registration-required/>) or "wait" (e.g., <remote-server-timeout/>), or the error condition is <bad-request/>, <redirect/>, or <not-acceptable/>, the pubsub service SHOULD increment a bounce counter for that entity and MAY attempt to resend the event notification after some configurable amount of time. The service MAY terminate the subscription of the entity to that node if the bounce counter has reached some configurable limit.

12.5 Temporary Subscriptions

An implementation MAY enable an entity to subscribe to a node temporarily, i.e., only for as long as the subscriber is online in its current presence session. To subscribe temporarily, the subscriber MUST set the "pubsub#expire" subscription configuration option to a literal value of "presence".

Listing 219: A Temporary Subscription

```
<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='lease3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'>
      <x xmlns='jabber:x:data' type='submit'>
```

```
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        ...
        <field var='pubsub#expire'><value>presence</value></field>
        ...
      </x>
    </options>
  </pubsub>
</iq>
```

The service will then automatically cancel the subscription when it receives presence of type "unavailable" from the subscriber.
An implementation MAY enable the node owner to force all subscriptions to be temporary, which is useful for nodes that are also configured to use presence-based delivery. This setting uses the "pubsub#tempsub" node configuration option set to a value of true.

Listing 220: Owner sets all subscriptions to temporary

```
<iq type='set'
    from='hamlet@denmark.lit/elsinore'
    to='pubsub.shakespeare.lit'
    id='configtemp'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <configure node='princely_musings'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var='pubsub#tempsub'><value>true</value></field>
      </x>
    </configure>
  </pubsub>
</iq>
```

12.6 Presence-Based Delivery of Events

Implementations of pubsub MAY deliver event notifications only when the subscriber is online. In these cases, the option may be a node configuration option as shown in the examples above. To facilitate this, the pubsub service needs to subscribe to the subscriber's presence and check the subscriber's current presence information before sending any event notifications (as described in RFC 3921). Presence subscriptions MUST be based on the subscribed JID.

12.7 Not Routing Events to Offline Storage

Sending events to users of existing XMPP servers may force event notifications to be routed to offline storage for later delivery (as described in Best Practices for Handling Offline Messages (XEP-0160) [24]). This may not always be desirable. The possible ways of preventing this behavior include:

- Use presence-based subscription options as described above.
- Use delivery semantics as defined by Advanced Message Processing (XEP-0079) [25].
- Specify a message type of "headline", which in most existing server implementations will prevent offline storage.

12.8 Including a Message Body

If a service understands the semantics for a particular payload type and an entity's subscription is so configured (by the "pubsub#include_body" subscription option to true), the service SHOULD include an appropriate XMPP <body/> child element along with the payloads it sends in event notifications for a given node, where the body's XML character data summarizes or represents the information contained in the payload (this enables clients that do not understand the payload format to present the appropriate information to an end user). For example, the Atom <summary/> element (see RFC 4287) could be mapped to the XMPP <body/> element. A service MUST NOT provide the "pubsub#include_body" subscription option for a node if it does not have a defined way to transform part or all of the payload format into a sensible message body. A node owner MAY define an XSLT for transforming the payload format into a message body, via the "pubsub#body_xslt" node configuration option. This XSLT is applied by the pubsub service after receiving a publish request and before sending the appropriate notifications, not by the client before sending a publish request.
If the service does not understand the semantics for a particular payload type and therefore cannot transform the payload into a human-readable message body, it SHOULD NOT include a <body/> child.
If a subscriber has multiple subscriptions to the same node, where some of the SubIDs have include_body set to true and others have include_body set to false, the service SHOULD include a body with all notifications.

12.9 Node ID and Item ID Uniqueness

NodeIDs MUST be treated as unique identifiers within the context of a particular pubsub service.
If item identifiers are used, they MUST be treated as unique within the scope of the node.

[24] XEP-0160: Best Practices for Handling Offline Messages <https://xmpp.org/extensions/xep-0160.html>.
[25] XEP-0079: Advanced Message Processing <https://xmpp.org/extensions/xep-0079.html>.

The combination of the NodeID + ItemID MUST be unique within a given service, and MUST specify a single published item at a single node.
If a publisher publishes an item and the ItemID matches that of an existing item, the pubsub service MUST overwrite the existing item and generate a new event notification.
Because it is possible for a node's configuration to change such that ItemIDs are required (e.g., a change from transient to persistent), a service SHOULD use ItemIDs for internal tracking purposes even if it does not include them with the notifications it generates prior to the configuration change.

12.10 Item Caching

A service MAY cache the last item published to a node, even if the node is configured for transient publication (i.e., configured to not persist items). The last published item SHOULD be sent to new subscribers upon successful processing of a subscription request or approval by a node owner.
Note: Particular profiles of the generic publish-subscribe protocol MAY define more stringent requirements regarding the "cache-last-item" feature.

12.11 Batch Processing

A publisher MAY include multiple <item/> elements in a publish request and MAY include multiple <item/> elements in a retract request. This results in "batch processing" of publications or retractions.
If the service cannot process any one of the items to be published or retracted, the entire batch MUST fail and the service MUST NOT publish or retract any of the items.
If a batch publish contains so many items that publication of all the items would exceed the maximum number of items for the node, the service MUST return a <not-allowed/> error, which SHOULD also include a pubsub-specific error condition of <max-items-exceeded/>.
Note: Batch publication renders the concept of "last published item" problematic; therefore, if information coherence is needed, a publisher SHOULD publish items in separate requests rather than in batch mode.

12.12 Auto-Subscribing Owners and Publishers

A service MUST allow owners and publishers to subscribe to a node, and to retrieve items from a node even if they are not subscribed. A service MAY auto-subscribe owners and publishers if they are not already subscribed, in which case it SHOULD generate a subscription ID if necessary for the subscription and SHOULD send a notification of successful subscription as described in the Notification of Subscription State Changes section of this document.

12.13 Authorizing Subscription Requests (Pending Subscribers)

How subscription requests are sent to node owners is a matter of implementation. Possibilities include:

- Send requests to all owners (these may be placed in offline storage as described in XEP-0160) and first approval wins.
- The service could subscribe to owner presence, and send only to the owners that are online.
- All owners vote on the new subscriber.
- Any owner is allowed to veto the subscriber.

An implementation MAY use any of these methods, or some other method not defined herein.

12.14 Notification of Subscription State Changes

Various actions and events may result in changes to a subscription state:

- Approval or denial of a subscription request as described in the Manage Subscription Requests use case

- Cancellation of an existing subscription, for which many ”triggers” are possible:

  - The entity simply unsubscribes from the node
  - The node is of type ”presence” and the underlying presence subscription is cancelled
  - The node is of type ”roster” and the entity is moved to an unauthorized roster group

When a subscription state change occurs, a service SHOULD send a message to the (new, former, or denied) subscriber informing it of the change, where the message contains an <event/> element with a single <subscription/> child that specifies the node, JID, and subscription state. The notification MAY contain a <body/> element specifying natural-language text regarding the subscription change. The JID to which the service sends the notification is the address that was set in the ’jid’ attribute of the subscription request. Examples are shown below.

Listing 221: Subscription approval notification

```
<message
    from='pubsub.shakespeare.lit'
    to='horatio@denmark.lit'
    id='approvalnotify1'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <subscription node='princely_musings' jid='horatio@denmark.lit'
        subscription='subscribed'/>
  </event>
</message>
```

Listing 222: Subscription cancellation / denial notification

```
<message
    from='pubsub.shakespeare.lit'
    to='horatio@denmark.lit'
    id='unsubnotify1'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <subscription node='princely_musings' jid='horatio@denmark.lit'
        subscription='none'/>
  </event>
</message>
```

If the service has knowledge of the (former or denied) subscriber's presence, it SHOULD send the message to all of the subscriber's resources; if not, it MUST send the message to the subscriber's affiliated JID.
If a service or node supports this feature, it MUST return a feature of "subscription-notifications" in its response to service discovery information requests.

12.15 NodeID Semantics

NodeIDs MAY have semantic meaning in particular profiles, implementations, or deployments of pubsub. However, it is STRONGLY RECOMMENDED that such semantic meaning not be used to encapsulate the hierarchical structure of nodes; instead, node hierarchy SHOULD be encapsulated using collections and their associated child nodes as described in XEP-0248.

12.16 Inclusion of SHIM Headers

When SubIDs are used, Stanza Headers and Internet Metadata (SHIM) headers are to be included in order to differentiate notifications sent regarding a particular subscription. The relevant use cases and scenarios are:

- Sending notifications regarding newly-published items as described in the Publish an Item to a Node use case.

- Sending notifications regarding deleted items as described in the Delete an Item from a Node use case.

The SHIM headers are generated by the node to which the subscriber has a subscription, which may be either a leaf node or a collection node (refer to XEP-0248).
SHIM headers are not to be included when the content does not differ based on subscription ID, e.g., when a node sends notification of a configuration change to the node itself, notification that the node has been purged, or notification that the node has been deleted.

12.17 Associating Events and Payloads with the Generating Entity

An implementation MAY enable the node configuration to specify an association between the event notification and the entity to which the published information pertains, but such a feature is OPTIONAL. Here are some possible examples:

- In the context of a geolocation notification service using User Geolocation (XEP-0080) [26], the user may generate the geolocation information or the information may be generated by an automated service (e.g., a service offered by a mobile telephony provider), but in either case the information is *about* the user's geolocation and therefore all replies should go to the user (who is probably the node owner).
- In the context of a group weblog, different users might publish to the weblog and replies might go to the publisher of an entry rather than to the weblog owner.
- In the context of an integrated pubsub and multi-user chat system, the node owner might be the room owner but all replies need to be sent to the room rather than to the owner.

Therefore we define the "itemreply" node configuration option, with three possible values:

- "owner" (i.e., the node owner or an alias for the node owners)
- "publisher" (i.e., the item publisher)
- "none"

A node owner MUST NOT define more than one of these options.
Interoperability note: As of March 2023, the value of "owner" is not widely supported in generic implementations, and clients may choose to avoid it. In contrast, the value of "none" was not specified in previous versions of this document but has been added retrospectively due to implementation support. Therefore, the primary use of the 'pubsub#itemreply' option is to allow node owners to determine whether the server will include a 'publisher' attribute in items or not.

[26] XEP-0080: User Geolocation <https://xmpp.org/extensions/xep-0080.html>.

The JID associated with an item is conveyed via the 'publisher' attribute on the <item/> element, as demonstrated in the below example:

Listing 223: Event notification with publisher JID

```
<message from='pubsub.shakespeare.lit'
         to='bassanio@merchantofvenice.lit'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='n48ad4fj78zn38st734'>
      <item id='i1s2d3f4g5h6bjeh936'
            publisher='portia@merchantofvenice.lit'>
        <geoloc xmlns='http://jabber.org/protocol/geoloc'>
          <description>Venice</description>
          <lat>45.44</lat>
          <lon>12.33</lon>
        </geoloc>
      </item>
    </items>
  </event>
</message>
```

Alternatively, if a service implements the personal eventing subset of this protocol, the virtual pubsub service is the account owner's bare JID and notifications are sent from that JID; for details, refer to XEP-0163.

12.18 Chaining

The word "chaining" refers to the practice of subscribing one node to another node. For instance, consider a scenario in which the node <pubsub@example.net/NewsBroadcaster> wants to distribute information received from the node "NewsFeed" at <pubsub.example.com>. While it is theoretically possible for <pubsub@example.net/NewsBroadcaster> to directly subscribe to the NewsFeed node (since the former node is directly addressable as a JID), implementations MUST NOT chain nodes in this fashion. Instead, implementations MUST subscribe from the address of the pubsub service rather than the node (in the example shown here, the subscription would be sent from <pubsub@example.net> rather than <pubsub@example.net/NewsBroadcaster>).

12.19 Time-Based Subscriptions (Leases)

In some systems it may be desirable to provide a subscription "leasing" feature in order to expire old or stale subscriptions. Leases can be implemented using configurable subscription options; specifically, when an entity subscribes, the service would require configuration of subscription options and the configuration form would contain a field of "pubsub#expire". This field MUST contain a dateTime (as specified in XMPP Date and Time Profiles (XEP-0082)

[27]).
The leasing process is shown below.

Listing 224: Leasing process

```
<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='lease1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe
        node='princely_musings'
        jid='francisco@denmark.lit'/>
  </pubsub>
</iq>

<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='lease1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='princely_musings'
        jid='francisco@denmark.lit'
        subscription='unconfigured'>
      <subscribe-options>
        <required/>
      </subscribe-options>
    </subscription>
  </pubsub>
</iq>

<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='lease2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'/>
  </pubsub>
</iq>

<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='lease2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'>
```

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

```
      <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        ...
        <field var='pubsub#expire' type='text-single'
               label='Requested␣lease␣period'/>
        ...
      </x>
    </options>
  </pubsub>
</iq>

<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='lease3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        ...
        <field var='pubsub#expire'><value>2006-02-28T11:59:59Z</value>
            </field>
        ...
      </x>
    </options>
  </pubsub>
</iq>

<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='lease3'/>
```

The service MAY send a message to the subscriber when the lease is almost over (e.g., 24 hours before the end of the lease term). This MUST be done by sending a <message/> containing a <subscription/> element qualified by the 'http://jabber.org/protocol/pubsub#event' namespace and including an 'expiry' attribute.

Listing 225: Service notifies subscriber of impending lease end

```
<message from='pubsub.shakespeare.lit' to='francisco@denmark.lit/
    barracks'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
```

```
    <subscription
        expiry='2006-02-28T23:59:59Z'
        jid='francisco@denmark.lit'
        node='princely_musings'
        subid='ba49252aaa4f5d320c24d3766f0bdcade78c78d3'
        subscription='subscribed'/>
  </event>
</message>
```

When the subscriber wants to renew the lease, it would get the current subscription options, change the value of the "pubsub#expire" field, and submit the new subscription options back to the service. If the new expire value exceeds the maximum value allowed for subscription leases, the service MUST change the value of the field to be the current date/time plus the maximum allowed lease period.

Listing 226: Renewing a lease

```
<iq type='get'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
    id='renew1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'/>
  </pubsub>
</iq>

<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='renew1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'>
      <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        ...
        <field var='pubsub#expire' type='text-single'
               label='Requested␣lease␣period'/>
        ...
      </x>
    </options>
  </pubsub>
</iq>

<iq type='set'
    from='francisco@denmark.lit/barracks'
    to='pubsub.shakespeare.lit'
```

```
    id='renew2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings' jid='francisco@denmark.lit'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
             value>
        </field>
        ...
        <field var='pubsub#expire'><value>2006-03-31T23:59:59Z</value>
           </field>
        ...
      </x>
    </options>
  </pubsub>
</iq>

<iq type='result'
    from='pubsub.shakespeare.lit'
    to='francisco@denmark.lit/barracks'
    id='renew2'/>
```

12.20 Content-Based Pubsub Systems

A service MAY enable entities to subscribe to nodes and apply a filter to notifications (e.g., keyword matching such as "send me all news entries from Slashdot that match the term 'XMPP'"). Such a content-based service SHOULD allow an entity to subscribe more than once to the same node and, if so, MUST use subscription identifiers (SubIDs) to distinguish between multiple subscriptions. In order to prevent collisions, a service that supports content-based subscriptions using SubIDs SHOULD generate SubIDs on behalf of subscribers rather than allowing subscribers to set their own SubIDs. [28]

Content-based services SHOULD use subscription options to specify the filter(s) to be applied. Because there many possible filtering mechanisms (many of which may be application-specific), this document does not define any such method. However, filtering mechanisms may be defined in separate specifications.

A fictional example of the subscription options configuration process for content-based pubsub is shown below.

Listing 227: A content-based subscription

```
<iq type='set'
    from='bard@shakespeare.lit/globe'
    to='pubsub.shakespeare.lit'
    id='filter1'>
```

[28] Another way to implement content-based subscriptions is to host one node per keyword or other filter; however, this is likely to require an extremely large number of nodes.

```
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe
        node='princely_musings'
        jid='francisco@denmark.lit'/>
  </pubsub>
</iq>

<iq type='result'
    from='pubsub.shakespeare.lit'
    to='bard@shakespeare.lit/globe'
    id='filter1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscription
        node='princely_musings'
        jid='bard@shakespeare.lit'
        subid='991d7fd1616fd041015064133cd097a10030819e'
        subscription='unconfigured'>
      <subscribe-options>
        <required/>
      </subscribe-options>
    </subscription>
  </pubsub>
</iq>

<iq type='get'
    from='bard@shakespeare.lit/globe'
    to='pubsub.shakespeare.lit'
    id='filter2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings'
             jid='bard@shakespeare.lit'
             subid='991d7fd1616fd041015064133cd097a10030819e'/>
  </pubsub>
</iq>

<iq type='result'
    from='pubsub.shakespeare.lit'
    to='bard@shakespeare.lit/globe'
    id='filter2'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings'
             jid='bard@shakespeare.lit'
             subid='991d7fd1616fd041015064133cd097a10030819e'>
      <x xmlns='jabber:x:data' type='form'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        ...
```

```
        <field var='x-http://shakespeare.lit/search#keyword'
               type='text-single'
               label='Keyword␣to␣match'/>
        ...
      </x>
    </options>
  </pubsub>
</iq>

<iq type='set'
    from='bard@shakespeare.lit/globe'
    to='pubsub.shakespeare.lit'
    id='filter3'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <options node='princely_musings'
             jid='bard@shakespeare.lit'
             subid='991d7fd1616fd041015064133cd097a10030819e'>
      <x xmlns='jabber:x:data' type='submit'>
        <field var='FORM_TYPE' type='hidden'>
          <value>http://jabber.org/protocol/pubsub#subscribe_options</
              value>
        </field>
        ...
        <field var='x-http://shakespeare.lit/search#keyword'><value>
            peasant</value></field>
        ...
      </x>
    </options>
  </pubsub>
</iq>

<iq type='result'
    from='pubsub.shakespeare.lit'
    to='bard@shakespeare.lit/globe'
    id='filter3'/>
```

The subscriber will then be notified about events that match the keyword.

Listing 228: Event notification for matched keyword

```
<message from='pubsub.shakespeare.lit' to='bard@shakespeare.lit'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='princely_musings'>
      <item id='4e30f35051b7b8b42abe083742187228'>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>Alone</title>
          <summary>
Now I am alone.
O, what a rogue and peasant slave am I!
```

```
          </summary>
          <link rel='alternate' type='text/html'
                href='http://denmark.lit/2003/12/13/atom03'/>
          <id>tag:denmark.lit,2003:entry-32396</id>
          <published>2003-12-13T11:09:53Z</published>
          <updated>2003-12-13T11:09:53Z</updated>
        </entry>
      </item>
    </items>
  </event>
  <headers xmlns='http://jabber.org/protocol/shim'>
    <header name='SubID'>991d7fd1616fd041015064133cd097a10030819e</
        header>
  </headers>
</message>
```

12.21 Singleton Nodes

For some nodes, it is desirable to have at most one item associated with the node at any one time (for example, a client may want to store its preferences using a node name that is a namespace controlled by that client). When this pattern is desired, it is RECOMMENDED for the publisher to specify an ItemID of "current" to ensure that the publication of a new item will overwrite the existing item.

Listing 229: Publishing to a Singleton Node

```
<iq type='set'
    from='horatio@denmark.lit/mobile'
    to='pubsub.shakespeare.lit'
    id='single1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='elsinore-doorbell'>
      <item id='current'/>
    </publish>
  </pubsub>
</iq>
```

Naturally, the node owner can enforce the singleton node pattern by setting the max_items configuration option to "1".

12.22 PubSub URIs

An XMPP URI (see RFC 5122 [29]) can be used for the purpose of identification or interaction. Some examples are provided below.

[29] RFC 5122: Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP) <http://tools.ietf.org/html/rfc5122>.

The following URI merely identifies a pubsub node.

Listing 230: XMPP URI for a node

```
xmpp:pubsub.shakespeare.lit?;node=princely_musings
```

The following URI identifies a specific item at a node.

Listing 231: XMPP URI for a pubsub item

```
xmpp:pubsub.shakespeare.lit?;node=princely_musings;item=
    ae890ac52d0df67ed7cfdf51b644e901
```

The following URI defines how to subscribe to a node (for details, see the URI Query Types section of this document).

Listing 232: URI for subscribing to a pubsub node

```
xmpp:pubsub.shakespeare.lit?pubsub;action=subscribe;node=
    princely_musings
```

The following URI defines how to retrieve a specific item from a node.

Listing 233: URI for retrieving a pubsub item

```
xmpp:pubsub.shakespeare.lit?pubsub;action=retrieve;node=
    princely_musings;item=ae890ac52d0df67ed7cfdf51b644e901
```

12.23 Integration with Data Forms

This section defines a datatype "integer-or-max" to be used with Data Forms Validation (XEP-0122) [30]. An implementation may use this type to validate fields such as "pubsub#children_max", "pubsub#item_expire" or "pubsub#max_items".

```
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema' targetNamespace
    ="http://jabber.org/protocol/pubsub#integer-or-max">
  <xs:simpleType name='integer-or-max'>
    <xs:union>
      <xs:simpleType>
        <xs:restriction base='integer'/>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base='string'>
          <xs:enumeration value='max'/>
        </xs:restriction>
```

[30] XEP-0122: Data Forms Validation <https://xmpp.org/extensions/xep-0122.html>.

```
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>
</xs:schema>
```

The meaning of a value of this type is defined in each field named above.
When the <range/> element is used in combination with this datatype, it applies to the integer value.

13 Internationalization Considerations

13.1 Field Labels

The Data Forms shown in this specification include English-language labels for various fields; implementations that will display such forms to human users SHOULD provide localized label text for fields that are defined for the registered FORM_TYPEs.

14 Security Considerations

14.1 Private Information

The data published to a pubsub node might contain sensitive information (e.g., a user's geolocation). Therefore, node owners SHOULD exercise care in approving subscription requests. Security considerations regarding particular kinds of information are the responsibility of the "using protocol".

14.2 Authorization

XMPP PubSub contains a hierarchy of affiliations for the purpose of authorization and access control. A service MUST NOT allow non-owners or other unauthorized entities to complete any actions defined under the Owner Use Cases section of this document.

14.3 Access Models

A service MUST adhere to the defined access model in determining whether to send event notifications or payloads to an entity, or allow an entity to retrieve items from a node. A service MAY enforce additional privacy and security policies when determining whether an entity is allowed to subscribe to a node or retrieve items from a node; however, any such policies shall be considered specific to an implementation or deployment and are out of scope for this document.

14.4 Presence Leaks

In the context of instant messaging systems it is possible for the act of publishing an item to reveal the node owner or item publisher's network availability. However, this risk is mitigated by the following factors:

1. A node does not necessarily reveal the existence of the publishing entity.
2. XMPP PubSub systems are not necessarily tied to instant messaging systems.
3. Even in the context of IM systems, a node provides information distinct from network availability (e.g., user tunes).
4. Even then, the actual publisher might not be an IM user (e.g., an automated calendaring or geolocation system).

15 IANA Considerations

This document does not require interaction with the Internet Assigned Numbers Authority (IANA) [31].

16 XMPP Registrar Considerations

16.1 Protocol Namespaces

The XMPP Registrar [32] includes the following namespaces in its registry of protocol namespaces (see <https://xmpp.org/registrar/namespaces.html>):

- http://jabber.org/protocol/pubsub
- http://jabber.org/protocol/pubsub#errors
- http://jabber.org/protocol/pubsub#event
- http://jabber.org/protocol/pubsub#owner

[31] 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/>.

[32] 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/>.

16.2 Service Discovery Category/Type

The XMPP Registrar includes a category of "pubsub" in its registry of Service Discovery identities (see <https://xmpp.org/registrar/disco-categories.html>), as well as three specific types within that category:

collection	A pubsub node of the "collection" type as described in XEP-0248.
leaf	A pubsub node of the "leaf" type as described in XEP-0060.
service	A pubsub service that supports the functionality defined in XEP-0060. Prior to version 1.5 of XEP-0060, this type was called "generic".

The registry submission is as follows:

```
<category>
  <name>pubsub</name>
  <desc>Services and nodes that adhere to XEP-0060.</desc>
  <type>
    <name>collection</name>
    <desc>A pubsub node of the "collection" type.</desc>
    <doc>XEP-0248</doc>
  </type>
  <type>
    <name>leaf</name>
    <desc>A pubsub node of the "leaf" type.</desc>
    <doc>XEP-0060</doc>
  </type>
  <type>
    <name>service</name>
    <desc>A pubsub service that supports the functionality defined in
       XEP-0060.</desc>
    <doc>XEP-0060</doc>
  </type>
</category>
```

Future submissions to the XMPP Registrar may register additional types.

16.3 Service Discovery Features

The XMPP Registrar maintains a registry of service discovery features (see <https://xmpp.org/registrar/disco-features.html>), which includes a number of features that may be returned by pubsub services. The following registry submission has been provided to the XMPP Registrar for that purpose.

```
<var>
  <name>http://jabber.org/protocol/pubsub#access-authorize</name>
  <desc>The default node access model is authorize.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#access-open</name>
  <desc>The default node access model is open.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#access-presence</name>
  <desc>The default node access model is presence.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#access-roster</name>
  <desc>The default node access model is roster.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#access-whitelist</name>
  <desc>The default node access model is whitelist.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#auto-create</name>
  <desc>The service supports automatic creation of nodes on first
      publish.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#auto-subscribe</name>
  <desc>The service supports automatic subscription to a nodes based
      on presence subscription.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#collections</name>
  <desc>Collection nodes are supported.</desc>
  <doc>XEP-0248</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#config-node</name>
  <desc>Configuration of node options is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#config-node-max</name>
```

```
  <desc>Server supports ‘max‘ being set as a value for node
      configuration options: pubsub#max_items, pubsub#item_expire and
      pubsub#children_max</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#create-and-configure</name>
  <desc>Simultaneous creation and configuration of nodes is supported.
      </desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#create-nodes</name>
  <desc>Creation of nodes is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#delete-items</name>
  <desc>Deletion of items is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#delete-nodes</name>
  <desc>Deletion of nodes is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#filtered-notifications</name
      >
  <desc>The service supports filtering of notifications based on
      Entity Capabilities.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#get-pending</name>
  <desc>Retrieval of pending subscription approvals is supported.</
      desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#instant-nodes</name>
  <desc>Creation of instant nodes is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#item-ids</name>
  <desc>Publishers may specify item identifiers.</desc>
  <doc>XEP-0060</doc>
</var>
```

```
<var>
  <name>http://jabber.org/protocol/pubsub#last-published</name>
  <desc>
    The service supports sending of the last published item to new
    subscribers and to newly available resources.
  </desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#leased-subscription</name>
  <desc>Time-based subscriptions are supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#manage-subscriptions</name>
  <desc>Node owners may manage subscriptions.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#member-affiliation</name>
  <desc>The member affiliation is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#meta-data</name>
  <desc>Node metadata is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#modify-affiliations</name>
  <desc>Node owners may modify affiliations.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#multi-collection</name>
  <desc>A single leaf node can be associated with multiple collections
      .</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#multi-items</name>
  <desc>The service supports the storage of multiple items per node.</
      desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#multi-subscribe</name>
  <desc>A single entity may subscribe to a node multiple times.</desc>
  <doc>XEP-0060</doc>
```

```
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#outcast-affiliation</name>
  <desc>The outcast affiliation is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#persistent-items</name>
  <desc>Persistent items are supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#presence-notifications</name
     >
  <desc>Presence-based delivery of event notifications is supported.</
     desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#presence-subscribe</name>
  <desc>Implicit presence-based subscriptions are supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#publish</name>
  <desc>Publishing items is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#publish-options</name>
  <desc>Publication with publish options is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#publish-only-affiliation</
     name>
  <desc>The publish-only affiliation is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#publisher-affiliation</name>
  <desc>The publisher affiliation is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#publish-node-full</name>
  <desc>Specified behaviour may happen when publishing on a full node.
     </desc>
  <doc>XEP-0060</doc>
```

```
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#purge-nodes</name>
  <desc>Purging of nodes is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#retract-items</name>
  <desc>Item retraction is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#retrieve-affiliations</name>
  <desc>Retrieval of current affiliations is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#retrieve-default</name>
  <desc>Retrieval of default node configuration is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#retrieve-default-sub</name>
  <desc>Retrieval of default subscription configuration is supported.<
      /desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#retrieve-items</name>
  <desc>Item retrieval is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#retrieve-subscriptions</name
      >
  <desc>Retrieval of current subscriptions is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#subscribe</name>
  <desc>Subscribing and unsubscribing are supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
  <name>http://jabber.org/protocol/pubsub#subscription-options</name>
  <desc>Configuration of subscription options is supported.</desc>
  <doc>XEP-0060</doc>
</var>
<var>
```

```
  <name>http://jabber.org/protocol/pubsub#subscription-notifications</
      name>
  <desc>Notification of subscription state changes is supported.</desc
      >
  <doc>XEP-0060</doc>
</var>
```

16.4 Field Standardization

XEP-0068 defines a process for standardizing the fields used within Data Forms scoped by a particular namespace, and the XMPP Registrar maintains a registry of such FORM_TYPES (see <https://xmpp.org/registrar/formtypes.html>). Within pubsub, there are four uses of such forms:

1. Authorization of subscriptions using the 'http://jabber.org/protocol/pubsub#subscribe_-authorization' namespace.
2. Configuration of subscription options using the 'http://jabber.org/protocol/pubsub#subscribe_-options' namespace.
3. Configuration of a node using the 'http://jabber.org/protocol/pubsub#node_config' namespace.
4. Setting of metadata information using the 'http://jabber.org/protocol/pubsub#meta-data' namespace.

The registry submissions associated with these namespaces are defined below.
Note: There is no requirement that configuration fields need to be registered with the XMPP Registrar. However, as specified in Section 3.4 of XEP-0068, names of custom (unregistered) fields MUST begin with the characters "x-" if the form itself is scoped by a registered FORM_-TYPE.

16.4.1 pubsub#subscribe_authorization FORM_TYPE

```
<form_type>
  <name>http://jabber.org/protocol/pubsub#subscribe_authorization</
      name>
  <doc>XEP-0060</doc>
  <desc>Forms enabling authorization of subscriptions to pubsub nodes<
      /desc>
  <field
      var='pubsub#allow'
      type='boolean'
      label='Whether␣to␣allow␣the␣subscription'/>
  <field
```

```
      var='pubsub#node'
      type='text-single'
      label='The␣NodeID␣of␣the␣relevant␣node'/>
  <field
      var='pubsub#subscriber_jid'
      type='jid-single'
      label='The␣address␣(JID)␣of␣the␣subscriber'/>
  <field
      var='pubsub#subid'
      type='text-single'
      label='The␣subscription␣identifier␣associated␣with␣the␣
         subscription␣request'/>
</form_type>
```

16.4.2 pubsub#subscribe_options FORM_TYPE

```
<form_type>
  <name>http://jabber.org/protocol/pubsub#subscribe_options</name>
  <doc>XEP-0060</doc>
  <desc>Forms enabling configuration of subscription options for
     pubsub nodes</desc>
  <field
      var='pubsub#deliver'
      type='boolean'
      label='Whether␣an␣entity␣wants␣to␣receive
␣␣␣␣␣␣␣␣␣␣␣␣␣or␣disable␣notifications'/>
  <field
      var='pubsub#digest'
      type='boolean'
      label='Whether␣an␣entity␣wants␣to␣receive␣digests
␣␣␣␣␣␣␣␣␣␣␣␣␣(aggregations)␣of␣notifications␣or␣all
␣␣␣␣␣␣␣␣␣␣␣␣␣notifications␣individually'/>
  <field var='pubsub#digest_frequency'
         type='text-single'
         label='The␣minimum␣number␣of␣milliseconds␣between
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣sending␣any␣two␣notification␣digests'/>
  <field
      var='pubsub#expire'
      type='text-single'
      label='The␣DateTime␣at␣which␣a␣leased␣subscription
␣␣␣␣␣␣␣␣␣␣␣␣␣will␣end␣or␣has␣ended'/>
  <field
      var='pubsub#include_body'
      type='boolean'
      label='Whether␣an␣entity␣wants␣to␣receive␣an␣XMPP
␣␣␣␣␣␣␣␣␣␣␣␣␣message␣body␣in␣addition␣to␣the␣payload
␣␣␣␣␣␣␣␣␣␣␣␣␣format'/>
  <field
```

```
      var='pubsub#show-values'
      type='list-multi'
      label='The presence states for which an entity
            wants to receive notifications'>
    <option label='XMPP Show Value of Away'>
      <value>away</value>
    </option>
    <option label='XMPP Show Value of Chat'>
      <value>chat</value>
    </option>
    <option label='XMPP Show Value of DND (Do Not Disturb)'>
      <value>dnd</value>
    </option>
    <option label='Mere Availability in XMPP (No Show Value)'>
      <value>online</value>
    </option>
    <option label='XMPP Show Value of XA (Extended Away)'>
      <value>xa</value>
    </option>
  </field>
  <field var='pubsub#subscription_type'
         type='list-single'>
    <option label='Receive notification of new items only'>
      <value>items</value>
    </option>
    <option label='Receive notification of new nodes only'>
      <value>nodes</value>
    </option>
  </field>
  <field var='pubsub#subscription_depth'
         type='list-single'>
    <option label='Receive notification from direct child nodes only'>
      <value>1</value>
    </option>
    <option label='Receive notification from all descendent nodes'>
      <value>all</value>
    </option>
  </field>
</form_type>
```

16.4.3 pubsub#meta-data FORM_TYPE

```
<form_type>
  <name>http://jabber.org/protocol/pubsub#meta-data</name>
  <doc>XEP-0060</doc>
  <desc>Forms enabling setting of metadata information about pubsub
      nodes</desc>
  <field var='pubsub#contact'
```

```
        type='jid-multi'
        label='The JIDs of those to contact with questions'/>
  <field var='pubsub#creation_date'
        type='text-single'
        label='The datetime when the node was created'/>
  <field var='pubsub#creator'
        type='jid-single'
        label='The JID of the node creator'/>
  <field var='pubsub#description'
        type='text-single'
        label='A description of the node'/>
  <field var='pubsub#language'
        type='list-single'
        label='The default language of the node'/>
  <field var='pubsub#access_model'
        type='list-single'
        label='Access model'/>
  <field var='pubsub#publish_model'
        type='list-single'
        label='Publish model'/>
  <field var='pubsub#num_subscribers'
        type='text-single'
        label='The number of subscribers to the node'/>
  <field var='pubsub#owner'
        type='jid-multi'
        label='The JIDs of those with an affiliation of owner'/>
  <field var='pubsub#publisher'
        type='jid-multi'
        label='The JIDs of those with an affiliation of publisher'/>
  <field var='pubsub#title'
        type='text-single'
        label='The name of the node'/>
  <field var='pubsub#type'
        type='text-single'
        label='Payload semantic type information'/>
  <field var='pubsub#max_items'
        type='text-single'
        label='Max # of items to persist'/>
</form_type>
```

16.4.4 pubsub#node_config FORM_TYPE

```
<form_type>
  <name>http://jabber.org/protocol/pubsub#node_config</name>
  <doc>XEP-0060</doc>
  <desc>Forms enabling configuration of pubsub nodes</desc>
  <field var='pubsub#access_model'
        type='list-single'
```

```
          label='Who may subscribe and retrieve items'>
    <option label='Subscription requests must be approved and only
        subscribers may retrieve items'>
      <value>authorize</value>
    </option>
    <option label='Anyone may subscribe and retrieve items'>
      <value>open</value>
    </option>
    <option label='Anyone with a presence subscription of both or from
        may subscribe and retrieve items'>
      <value>presence</value>
    </option>
    <option label='Anyone in the specified roster group(s) may
        subscribe and retrieve items'>
      <value>roster</value>
    </option>
    <option label='Only those on a whitelist may subscribe and
        retrieve items'>
      <value>whitelist</value>
    </option>
  </field>
  <field var='pubsub#body_xslt'
         type='text-single'
         label='The URL of an XSL transformation which can be
                applied to payloads in order to generate an
                appropriate message body element.'/>
  <field var='pubsub#children_association_policy'
         type='list-single'
         label='Who may associate leaf nodes with a collection'>
    <option label='Anyone may associate leaf nodes with the collection
        '>
      <value>all</value>
    </option>
    <option label='Only collection node owners may associate leaf
        nodes with the collection'>
      <value>owners</value>
    </option>
    <option label='Only those on a whitelist may associate leaf nodes
        with the collection'>
      <value>whitelist</value>
    </option>
  </field>
  <field var='pubsub#children_association_whitelist'
         type='jid-multi'
         label='The list of JIDs that may associate leaf nodes with a
             collection'/>
  <field var='pubsub#children'
         type='text-multi'
```

```
        label='The child nodes (leaf or collection) associated with a
            collection.'/>
  <field var='pubsub#children_max'
        type='text-single'
        label='The maximum number of child nodes that can be
            associated with a collection.   `max` for no specific
            limit other than a server imposed maximum.'/>
  <field var='pubsub#collection'
        type='text-multi'
        label='The collection(s) with which a node is affiliated'/>
  <field var='pubsub#contact'
        type='jid-multi'
        label='The JIDs of those to contact with questions'/>
  <field var='pubsub#dataform_xslt'
        type='text-single'
        label='The URL of an XSL transformation which can be
                applied to the payload format in order to generate
                a valid Data Forms result that the client could
                display using a generic Data Forms rendering
                engine'/>
  <field var='pubsub#deliver_notifications' type='boolean'
        label='Whether to deliver event notifications'>
    <value>true</value>
  </field>
  <field var='pubsub#deliver_payloads'
        type='boolean'
        label='Whether to deliver payloads with event notifications;
            applies only to leaf nodes'/>
  <field var='pubsub#description'
        type='text-single'
        label='A description of the node'/>
  <field var='pubsub#item_expire'
        type='text-single'
        label='Number of seconds after which to automatically purge
            items. `max` for no specific limit other than a server
            imposed maximum.'/>
  <field var='pubsub#itemreply'
        type='list-single'
        label='Whether owners or publisher should receive replies to
            items'>
    <option label='Statically specify a replyto of the node owner(s)'>
      <value>owner</value>
    </option>
    <option label='Dynamically specify a replyto of the item publisher
        '>
      <value>publisher</value>
    </option>
  </field>
  <field var='pubsub#language'
```

```
        type='list-single'
        label='The default language of the node'/>
<field var='pubsub#max_items'
        type='text-single'
        label='The maximum number of items to persist. 'max' for no
           specific limit other than a server imposed maximum.'/>
<field var='pubsub#max_payload_size'
        type='text-single'
        label='The maximum payload size in bytes'/>
<field var='pubsub#node_type'
        type='list-single'
        label='Whether the node is a leaf (default) or a collection'>
  <option label='The node is a leaf node (default)'>
    <value>leaf</value>
  </option>
  <option label='The node is a collection node'>
    <value>collection</value>
  </option>
</field>
<field var='pubsub#notification_type' type='list-single'
        label='Specify the delivery style for notifications'>
  <option label='Messages of type normal'>
    <value>normal</value>
  </option>
  <option label='Messages of type headline'>
    <value>headline</value>
  </option>
</field>
<field var='pubsub#notify_config'
        type='boolean'
        label='Whether to notify subscribers when the node
           configuration changes'/>
<field var='pubsub#notify_delete'
        type='boolean'
        label='Whether to notify subscribers when the node is deleted
           '/>
<field var='pubsub#notify_retract'
        type='boolean'
        label='Whether to notify subscribers when items are removed
           from the node'/>
<field var='pubsub#notify_sub'
        type='boolean'
        label='Whether to notify owners about new subscribers and
           unsubscribes'/>
<field var='pubsub#persist_items'
        type='boolean'
        label='Whether to persist items to storage'/>
<field var='pubsub#presence_based_delivery'
        type='boolean'
```

```
        label='Whether␣to␣deliver␣notifications␣to␣available␣users␣
           only'/>
<field var='pubsub#publish_model'
       type='list-single'
       label='The␣publisher␣model'>
  <option label='Only␣publishers␣may␣publish'>
    <value>publishers</value>
  </option>
  <option label='Subscribers␣may␣publish'>
    <value>subscribers</value>
  </option>
  <option label='Anyone␣may␣publish'>
    <value>open</value>
  </option>
</field>
<field var='pubsub#publish_node_full'
       type='list-single'
       label='Action␣to␣be␣performed␣by␣the␣service␣when␣publishing␣
           on␣a␣full␣node.'>
  <option label='Retract␣the␣oldest␣item␣alongside␣publishing␣the␣
      new␣item.'>
    <value>retract-oldest</value>
  </option>
  <option label='Discard␣the␣oldest␣item␣alongside␣publishing␣the␣
      new␣item.'>
    <value>discard-oldest</value>
  </option>
  <option label='Reject␣publication␣of␣the␣new␣item.'>
    <value>reject</value>
  </option>
</field>
<field var='pubsub#purge_offline'
       type='boolean'
       label='Whether␣to␣purge␣all␣items␣when␣the␣relevant␣publisher
           ␣goes␣offline'/>
<field var='pubsub#roster_groups_allowed'
       type='list-multi'
       label='The␣roster␣group(s)␣allowed␣to␣subscribe␣and␣retrieve␣
           items'/>
<field var='pubsub#send_last_published_item'
       type='list-single'
       label='When␣to␣send␣the␣last␣published␣item'>
  <option label='Never'>
    <value>never</value>
  </option>
  <option label='When␣a␣new␣subscription␣is␣processed'>
    <value>on_sub</value>
  </option>
```

```
    <option label='When␣a␣new␣subscription␣is␣processed␣and␣whenever␣a
       ␣subscriber␣comes␣online'>
      <value>on_sub_and_presence</value>
    </option>
  </field>
  <field var='pubsub#tempsub'
         type='boolean'
         label='Whether␣to␣make␣all␣subscriptions␣temporary,␣based␣on␣
            subscriber␣presence'/>
  <field var='pubsub#subscribe' type='boolean'
         label='Whether␣to␣allow␣subscriptions'>
    <value>1</value>
  </field>
  <field var='pubsub#title'
         type='text-single'
         label='A␣friendly␣name␣for␣the␣node'/>
  <field var='pubsub#type'
         type='text-single'
        label='The␣semantic␣type␣information␣of␣data␣in␣the␣node,␣
           usually␣specified
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣by␣the␣namespace␣of␣the␣payload␣(if␣any)'/>
</form_type>
```

16.5 SHIM Headers

The XMPP Registrar includes "Collection" and "SubID" in its registry of SHIM headers (see <https://xmpp.org/registrar/shim.html>). The registry submission is as follows:

```
<header>
  <name>Collection</name>
  <desc>The collection via which an event notification was received
     from the originating node.</desc>
  <doc>XEP-0248</doc>
</header>
<header>
  <name>SubID</name>
  <desc>A subscription identifer within the pubsub protocol.</desc>
  <doc>XEP-0060</doc>
</header>
```

Future submissions to the XMPP Registrar may register additional SHIM headers that can be used in relation to the pubsub protocol, and such submission may occur without updating this specification.

16.6 URI Query Types

As authorized by XMPP URI Query Components (XEP-0147) [33], the XMPP Registrar maintains a registry of queries and key-value pairs for use in XMPP URIs (see <https://xmpp.org/registrar/querytypes.html>).
The "pubsub" querytype is defined herein for interaction with pubsub services, with three keys: (1) "action" (whose defined values are "subscribe", "unsubscribe", and "retrieve"), (2) "node" (to specify a pubsub node), and optionally "item" (to specify a particular item at a node).

Listing 234: Pubsub Subscribe Action: IRI/URI

```
xmpp:pubsub.shakespeare.lit?pubsub;action=subscribe;node=
    princely_musings
```

Listing 235: Pubsub Subscribe Action: Resulting Stanza

```
<iq to='pubsub.shakespeare.lit' type='set'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe node='princely_musings'/>
  </pubsub>
</iq>
```

Listing 236: Pubsub Unsubscribe Action: IRI/URI

```
xmpp:pubsub.shakespeare.lit?pubsub;action=unsubscribe;node=
    princely_musings
```

Listing 237: Pubsub Unsubscribe Action: Resulting Stanza

```
<iq to='pubsub.shakespeare.lit' type='set'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <unsubscribe node='princely_musings'/>
  </pubsub>
</iq>
```

Listing 238: Pubsub Retrieve Action: IRI/URI

```
xmpp:pubsub.shakespeare.lit?pubsub;action=retrieve;node=
    princely_musings
```

Listing 239: Pubsub Retrieve Action: Resulting Stanza

```
<iq to='pubsub.shakespeare.lit' type='get'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='princely_musings'/>
  </pubsub>
</iq>
```

[33] XEP-0147: XMPP URI Query Components <https://xmpp.org/extensions/xep-0147.html>.

The following submission registers the "pubsub" querytype.

```
<querytype>
  <name>pubsub</name>
  <proto>http://jabber.org/protocol/pubsub</proto>
  <desc>enables interaction with a publish-subscribe service</desc>
  <doc>XEP-0060</doc>
  <keys>
    <key>
      <name>action</name>
      <desc>the pubsub action</desc>
      <values>
        <value>
          <name>subscribe</name>
          <desc>enables subscribing to a pubsub node</desc>
        </value>
        <value>
          <name>unsubscribe</name>
          <desc>enables unsubscribing from a pubsub node</desc>
        </value>
        <value>
          <name>retrieve</name>
          <desc>enables retrieving a specific item from a pubsub node<
             /desc>
        </value>
      </values>
    </key>
    <key>
      <name>node</name>
      <desc>the pubsub node</desc>
    </key>
    <key>
      <name>item</name>
      <desc>the pubsub item</desc>
    </key>
  </keys>
</querytype>
```

16.7 Data Forms Validation Datatypes

The following datatype shall be registered for use with Data Forms Validation:

```
<datatype>
  <name>integer-or-max</name>
    <desc>
      Datatype used for text-single fields where the accepted value is
      either an integer or the string "max".
```

```
    </desc>
    <doc>XEP-0060</doc>
</datatype>
```

17 XML Schemas

17.1 http://jabber.org/protocol/pubsub

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

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

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0060: http://xmpp.org/extensions/xep-0060.html
    </xs:documentation>
  </xs:annotation>

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

  <xs:element name='pubsub'>
    <xs:complexType>
      <xs:choice>
        <xs:sequence>
          <xs:element ref='create'/>
          <xs:element ref='configure' minOccurs='0'/>
        </xs:sequence>
        <xs:sequence>
          <xs:element ref='subscribe' minOccurs='0'/>
          <xs:element ref='options' minOccurs='0'/>
        </xs:sequence>
        <xs:sequence>
          <xs:element ref='publish'/>
          <xs:element ref='publish-options' minOccurs='0'/>
        </xs:sequence>
        <xs:choice minOccurs='0'>
          <xs:element ref='affiliations'/>
          <xs:element ref='default'/>
          <xs:element ref='items'/>
```

```
        <xs:element ref='retract'/>
        <xs:element ref='subscription'/>
        <xs:element ref='subscriptions'/>
        <xs:element ref='unsubscribe'/>
      </xs:choice>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name='affiliations'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='affiliation' minOccurs='0' maxOccurs='
          unbounded'/>
    </xs:sequence>
    <xs:attribute name='node' type='xs:string' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='affiliation'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='affiliation' use='required'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='member'/>
              <xs:enumeration value='none'/>
              <xs:enumeration value='outcast'/>
              <xs:enumeration value='owner'/>
              <xs:enumeration value='publisher'/>
              <xs:enumeration value='publish-only'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name='node' type='xs:string' use='required'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='configure'>
  <xs:complexType>
    <xs:choice minOccurs='0' xmlns:xdata='jabber:x:data'>
      <xs:element ref='xdata:x'/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

```
<xs:element name='create'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='node' type='xs:string' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='default'>
  <xs:complexType>
    <xs:choice minOccurs='0'>
      <xs:element ref='xdata:x'/>
    </xs:choice>
    <xs:attribute name='node' type='xs:string' use='optional'/>
    <xs:attribute name='type'
                  use='optional'
                  default='leaf'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='collection'/>
          <xs:enumeration value='leaf'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name='items'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='item' minOccurs='0' maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='max_items' type='xs:positiveInteger' use='
       optional'/>
    <xs:attribute name='node' type='xs:string' use='required'/>
    <xs:attribute name='subid' type='xs:string' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='item'>
  <xs:complexType>
    <xs:sequence minOccurs='0'>
      <xs:any namespace='##other'/>
    </xs:sequence>
    <xs:attribute name='id' type='xs:string' use='optional'/>
    <xs:attribute name='publisher' type='xs:string' use='optional'/>
  </xs:complexType>
```

```
</xs:element>

<xs:element name='options'>
  <xs:complexType>
    <xs:sequence minOccurs='0'>
      <xs:any namespace='jabber:x:data'/>
    </xs:sequence>
    <xs:attribute name='jid' type='xs:string' use='required'/>
    <xs:attribute name='node' type='xs:string' use='optional'/>
    <xs:attribute name='subid' type='xs:string' use='optional'/>
  </xs:complexType>
</xs:element>

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

<xs:element name='publish-options'>
  <xs:complexType>
    <xs:choice minOccurs='0' xmlns:xdata='jabber:x:data'>
      <xs:element ref='xdata:x'/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name='retract'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='item' minOccurs='1' maxOccurs='unbounded'/>
    </xs:sequence>
    <xs:attribute name='node' type='xs:string' use='required'/>
    <xs:attribute name='notify' type='xs:boolean' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='subscribe-options'>
  <xs:complexType>
    <xs:sequence>
      <xs:element name='required' type='empty' minOccurs='0'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name='subscribe'>
```

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

  <xs:element name='subscriptions'>
    <xs:complexType>
      <xs:sequence minOccurs='0' maxOccurs='unbounded'>
        <xs:element ref='subscription'/>
      </xs:sequence>
      <xs:attribute name='node' type='xs:string' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='subscription'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='subscribe-options' minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='jid' type='xs:string' use='required'/>
      <xs:attribute name='node' type='xs:string' use='optional'/>
      <xs:attribute name='subid' type='xs:string' use='optional'/>
      <xs:attribute name='subscription' use='optional'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='none'/>
            <xs:enumeration value='pending'/>
            <xs:enumeration value='subscribed'/>
            <xs:enumeration value='unconfigured'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

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

17.2 http://jabber.org/protocol/pubsub#errors

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

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

  <xs:annotation>
    <xs:documentation>
      This namespace is used for error reporting only, as
      defined in XEP-0060:

      http://xmpp.org/extensions/xep-0060.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='closed-node' type='empty'/>
  <xs:element name='configuration-required' type='empty'/>
  <xs:element name='invalid-jid' type='empty'/>
  <xs:element name='invalid-options' type='empty'/>
  <xs:element name='invalid-payload' type='empty'/>
  <xs:element name='invalid-subid' type='empty'/>
  <xs:element name='item-forbidden' type='empty'/>
  <xs:element name='item-required' type='empty'/>
  <xs:element name='jid-required' type='empty'/>
  <xs:element name='max-items-exceeded' type='empty'/>
  <xs:element name='max-nodes-exceeded' type='empty'/>
  <xs:element name='nodeid-required' type='empty'/>
  <xs:element name='not-in-roster-group' type='empty'/>
  <xs:element name='not-subscribed' type='empty'/>
  <xs:element name='payload-too-big' type='empty'/>
  <xs:element name='payload-required' type='empty'/>
  <xs:element name='pending-subscription' type='empty'/>
  <xs:element name='precondition-not-met' type='empty'/>
```

```
<xs:element name='presence-subscription-required' type='empty'/>
<xs:element name='subid-required' type='empty'/>
<xs:element name='too-many-subscriptions' type='empty'/>
<xs:element name='unsupported'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='feature' use='required'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='access-authorize'/>
              <xs:enumeration value='access-open'/>
              <xs:enumeration value='access-presence'/>
              <xs:enumeration value='access-roster'/>
              <xs:enumeration value='access-whitelist'/>
              <xs:enumeration value='auto-create'/>
              <xs:enumeration value='auto-subscribe'/>
              <xs:enumeration value='collections'/>
              <xs:enumeration value='config-node'/>
              <xs:enumeration value='create-and-configure'/>
              <xs:enumeration value='create-nodes'/>
              <xs:enumeration value='delete-items'/>
              <xs:enumeration value='delete-nodes'/>
              <xs:enumeration value='filtered-notifications'/>
              <xs:enumeration value='get-pending'/>
              <xs:enumeration value='instant-nodes'/>
              <xs:enumeration value='item-ids'/>
              <xs:enumeration value='last-published'/>
              <xs:enumeration value='leased-subscription'/>
              <xs:enumeration value='manage-subscriptions'/>
              <xs:enumeration value='member-affiliation'/>
              <xs:enumeration value='meta-data'/>
              <xs:enumeration value='modify-affiliations'/>
              <xs:enumeration value='multi-collection'/>
              <xs:enumeration value='multi-items'/>
              <xs:enumeration value='multi-subscribe'/>
              <xs:enumeration value='outcast-affiliation'/>
              <xs:enumeration value='persistent-items'/>
              <xs:enumeration value='presence-notifications'/>
              <xs:enumeration value='presence-subscribe'/>
              <xs:enumeration value='publish'/>
              <xs:enumeration value='publish-options'/>
              <xs:enumeration value='publish-only-affiliation'/>
              <xs:enumeration value='publisher-affiliation'/>
              <xs:enumeration value='purge-nodes'/>
              <xs:enumeration value='retract-items'/>
              <xs:enumeration value='retrieve-affiliations'/>
              <xs:enumeration value='retrieve-default'/>
              <xs:enumeration value='retrieve-default-sub'/>
```

```
                <xs:enumeration value='retrieve-items'/>
                <xs:enumeration value='retrieve-subscriptions'/>
                <xs:enumeration value='subscribe'/>
                <xs:enumeration value='subscription-options'/>
                <xs:enumeration value='subscription-notifications'/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='unsupported-access-model' type='empty'/>

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

</xs:schema>
```

17.3 http://jabber.org/protocol/pubsub#event

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

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

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0060: http://xmpp.org/extensions/xep-0060.html
    </xs:documentation>
  </xs:annotation>

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

  <xs:element name='event'>
    <xs:complexType>
      <xs:choice>
        <xs:element ref='collection'/>
```

```
      <xs:element ref='configuration'/>
      <xs:element ref='delete'/>
      <xs:element ref='items'/>
      <xs:element ref='purge'/>
      <xs:element ref='subscription'/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name='collection'>
  <xs:complexType>
    <xs:choice>
      <xs:element ref='associate'/>
      <xs:element ref='disassociate'/>
    </xs:choice>
    <xs:attribute name='node' type='xs:string' use='optional'/>
  </xs:complexType>
</xs:element>

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

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

<xs:element name='configuration'>
  <xs:complexType>
    <xs:sequence minOccurs='0' xmlns:xdata='jabber:x:data'>
      <xs:element ref='xdata:x'/>
    </xs:sequence>
    <xs:attribute name='node' type='xs:string' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='delete'>
```

```
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='redirect' minOccurs='0' maxOccurs='1'/>
      </xs:sequence>
      <xs:attribute name='node' type='xs:string' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='items'>
    <xs:complexType>
      <xs:choice>
        <xs:element ref='item' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element ref='retract' minOccurs='0' maxOccurs='unbounded'/
            >
      </xs:choice>
      <xs:attribute name='node' type='xs:string' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='item'>
    <xs:complexType>
      <xs:choice minOccurs='0'>
        <xs:any namespace='##other'/>
      </xs:choice>
      <xs:attribute name='id' type='xs:string' use='optional'/>
      <xs:attribute name='publisher' type='xs:string' use='optional'/>
    </xs:complexType>
  </xs:element>

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

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

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

  <xs:element name='subscription'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='expiry' type='xs:dateTime' use='optional
              '/>
          <xs:attribute name='jid' type='xs:string' use='required'/>
          <xs:attribute name='node' type='xs:string' use='optional'/>
          <xs:attribute name='subid' type='xs:string' use='optional'/>
          <xs:attribute name='subscription' use='optional'>
            <xs:simpleType>
              <xs:restriction base='xs:NCName'>
                <xs:enumeration value='none'/>
                <xs:enumeration value='pending'/>
                <xs:enumeration value='subscribed'/>
                <xs:enumeration value='unconfigured'/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </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>
```

17.4 http://jabber.org/protocol/pubsub#owner

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
```

```
  targetNamespace='http://jabber.org/protocol/pubsub#owner'
  xmlns='http://jabber.org/protocol/pubsub#owner'
  elementFormDefault='qualified'>

<xs:annotation>
  <xs:documentation>
    The protocol documented by this schema is defined in
    XEP-0060: http://xmpp.org/extensions/xep-0060.html
  </xs:documentation>
</xs:annotation>

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

<xs:element name='pubsub'>
  <xs:complexType>
    <xs:choice>
      <xs:element ref='affiliations'/>
      <xs:element ref='configure'/>
      <xs:element ref='default'/>
      <xs:element ref='delete'/>
      <xs:element ref='purge'/>
      <xs:element ref='subscriptions'/>
    </xs:choice>
  </xs:complexType>
</xs:element>

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

<xs:element name='affiliation'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='empty'>
        <xs:attribute name='affiliation' use='required'>
          <xs:simpleType>
            <xs:restriction base='xs:NCName'>
              <xs:enumeration value='member'/>
              <xs:enumeration value='none'/>
              <xs:enumeration value='outcast'/>
              <xs:enumeration value='owner'/>
```

```
              <xs:enumeration value='publisher'/>
              <xs:enumeration value='publish-only'/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name='jid' type='xs:string' use='required'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='configure'>
  <xs:complexType>
    <xs:choice minOccurs='0' xmlns:xdata='jabber:x:data'>
      <xs:element ref='xdata:x'/>
    </xs:choice>
    <xs:attribute name='node' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='default'>
  <xs:complexType>
    <xs:choice minOccurs='0' xmlns:xdata='jabber:x:data'>
      <xs:element ref='xdata:x'/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name='delete'>
  <xs:complexType>
    <xs:sequence>
      <xs:element ref='redirect' minOccurs='0' maxOccurs='1'/>
    </xs:sequence>
    <xs:attribute name='node' type='xs:string' use='required'/>
  </xs:complexType>
</xs:element>

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

<xs:element name='redirect'>
  <xs:complexType>
```

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

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

  <xs:element name='subscription'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='subscription' use='required'>
            <xs:simpleType>
              <xs:restriction base='xs:NCName'>
                <xs:enumeration value='none'/>
                <xs:enumeration value='pending'/>
                <xs:enumeration value='subscribed'/>
                <xs:enumeration value='unconfigured'/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name='jid' type='xs:string' use='required'/>
        </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>
```

18 Acknowledgements

Thanks to Kirk Bateman, Robin Collier, Blaine Cook, Ovidiu Craciun, Brian Cully, Dave Cridland, Guillaume Desmottes, Gaston Dombiak, William Edney, Seth Fitzsimmons, Fabio Forno, Nathan Fritz, Julien Genestoux, Anastasia Gornostaeva, Joe Hildebrand, Curtis King, Tuomas Koski, Petri Liimatta, Tobias Markmann, Pedro Melo, Dirk Meyer, Tory Patnoe, Peter Petrov, Sonny Piers, Christophe Romain, Pavel Šimerda, Andy Skelton, Kevin Smith, Chris Teegarden, Simon Tennant, Matt Tucker, Melvin Vermeeren, Matthew Wild, Bob Wyman, Matus Zamborsky, and Brett Zamir for their feedback.

19 Author Note

Peter Millard, primary author of this specification from version 0.1 through version 1.7, died on April 26, 2006. The remaining co-authors are indebted to him for his many years of work on publish-subscribe technologies.