XEP-0030: Service Discovery

Joe Hildebrand
mailto:jhildebr@cisco.com
xmpp:hildjj@jabber.org

Peter Millard

Ryan Eatmon
mailto:reatmon@jabber.org
xmpp:reatmon@jabber.org

Peter Saint-Andre
mailto:peter@andyet.net
xmpp:stpeter@stpeter.im
https://stpeter.im/

2016-10-13
Version 2.5rc2

Status	Type	Short Name
Final	Standards Track	disco

This specification defines an XMPP protocol extension for discovering information about other XMPP entities. Two kinds of information can be discovered: (1) the identity and capabilities of an entity, including the protocols and features it supports; and (2) the items associated with an entity, such as the list of rooms hosted at a multi-user chat service.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

The ability to discover information about entities on the Jabber network is extremely valuable. Such information might include features offered or protocols supported by the entity, the entity's type or identity, and additional entities that are associated with the original entity in some way (often thought of as "children" of the "parent" entity). While mechanisms for doing so are not defined in XMPP Core [1], several protocols have been used in the past within the Jabber community for service discovery, specifically Jabber Browsing (XEP-0011) [2] and Agent Information (XEP-0094) [3]. However, those protocols are perceived to be inadequate for several reasons:

1. Neither Jabber Browsing nor Agent Information is easily extensible. For example, the categories and subcategories listed for JID-Types in XEP-0011 are explicitly defined as the only official categories, and any additions to the list of JID-Types would require a modification to XEP-0011. While the Jabber Browsing specification does allow for the use of unofficial categories and types prefixed with the string 'x-', this introduces migration issues. This lack of flexibility violates one of the Jabber community's core XMPP Design Guidelines (XEP-0134) [4].

2. In Agent Information, there is no way to advertise supported features. While Jabber Browsing includes such a mechanism, the only way to express the availability of a feature is to advertise a supported protocol namespace. Yet some features may not be uniquely associated with a protocol namespace, which are one implementation of features but not the only one.

3. A Jabber Browsing result returns a combination of (1) namespaces supported by a Jabber Entity, (2) items associated with a Jabber Entity, and (3) namespaces supported by the associated items. This approach mixes information levels and requires parents to know everything about child nodes, thereby introducing significant confusion.

4. In both Jabber Browsing and Agent Information, items must be addressable as JIDs; however, this may not be possible in some applications.

This document addresses the perceived weaknesses of both the Jabber Browsing and Agent Information protocols. The result is a standards-track protocol for service discovery (often abbreviated to "disco", as is familiar in protocols such as SOAP [5]).

[1] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[2] XEP-0011: Jabber Browsing <https://xmpp.org/extensions/xep-0011.html>.
[3] XEP-0094: Agent Information <https://xmpp.org/extensions/xep-0094.html>.
[4] XEP-0134: XMPP Design Guidelines <https://xmpp.org/extensions/xep-0134.html>.
[5] SOAP <http://www.w3.org/TR/SOAP/>.

2 Requirements

The authors have designed the service discovery protocol with the following requirements in mind:

- The protocol MUST support all functionality supported by the protocols it supersedes (Jabber Browsing and Agent Information).

- There are three kinds of information that need to be discovered about an entity:

  1. its basic identity (type and/or category)
  2. the features it offers and protocols it supports
  3. any additional items associated with the entity, whether or not they are addressable as JIDs

  All three MUST be supported, but the first two kinds of information relate to the entity itself whereas the third kind of information relates to items associated with the entity itself; therefore two different query types are needed.

- Discovering information about a child item MUST be accomplished by sending a separate discovery request to that item, not to the parent entity. (One result of this is that discovering complete information about an entire tree will require multiple request/response pairs in order to ”walk the tree”.)

- The lists of identities and features MUST be flexible.

- The protocol itself MUST be extensible.

3 Discovering Information About a Jabber Entity

3.1 Basic Protocol

A requesting entity may want to discover information about another entity on the network. The information desired generally is of two kinds:

1. **The target entity’s identity.** In disco, an entity’s identity is broken down into its category (server, client, gateway, directory, etc.) and its particular type within that category (IM server, phone vs. handheld client, MSN gateway vs. AIM gateway, user directory vs. chatroom directory, etc.). This information helps requesting entities to

determine the group or "bucket" of services into which the entity is most appropriately placed (e.g., perhaps the entity is shown in a GUI with an appropriate icon). An entity MAY have multiple identities. When multiple identity elements are provided, the name attributes for each identity element SHOULD have the same value.

2. **The features offered and protocols supported by the target entity.** This information helps requesting entities determine what actions are possible with regard to this entity (registration, search, join, etc.), what protocols the entity supports, and specific feature types of interest, if any (e.g., for the purpose of feature negotiation).

In order to discover such information, the requesting entity MUST send an IQ stanza of type "get", containing an empty <query/> element qualified by the 'http://jabber.org/protocol/disco#info' namespace, to the JID of the target entity (a 'node' attribute on the <query/> element is OPTIONAL as described in the Info Nodes and Items Nodes section of this document):

Listing 1: Querying for information

```
<iq type='get'
    from='romeo@montague.net/orchard'
    to='plays.shakespeare.lit'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

The target entity then MUST either return an IQ result, or return an error (see the Error Conditions section of this document). The result MUST contain a <query/> element qualified by the 'http://jabber.org/protocol/disco#info' namespace, which in turn contains one or more <identity/> elements and one or more <feature/> elements. (Note: Every entity MUST have at least one identity, and every entity MUST support at least the 'http://jabber.org/protocol/disco#info' feature; however, an entity is not required to return a result and MAY return an error, most likely <feature-not-implemented/> or <service-unavailable/>, although other error conditions may be appropriate.)
Each <identity/> element MUST possess the 'category' and 'type' attributes specifying the category and type for the entity, and MAY possess a 'name' attribute specifying a natural-language name for the entity; the <identity/> element MAY also possess a standard 'xml:lang' attribute, which enables the entity to return localized results if desired (i.e., the <query/> element MAY include multiple <identity/> elements with the same category+type but with different 'xml:lang' values, however the <query/> element MUST NOT include multiple <identity/> elements with the same category+type+xml:lang but with different 'name' values).
Each <feature/> element MUST possess a 'var' attribute whose value is a protocol namespace or other feature offered by the entity, and MUST NOT have any children.
Preferably, both the category/type values and the feature values will be registered in a public

registry, as described in the XMPP Registrar Considerations section of this document.

Listing 2: Result-set for information request

```
<iq type='result'
    from='plays.shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        type='text'
        name='Play-Specific Chatrooms'/>
    <identity
        category='directory'
        type='chatroom'
        name='Play-Specific Chatrooms'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='http://jabber.org/protocol/muc'/>
    <feature var='jabber:iq:register'/>
    <feature var='jabber:iq:search'/>
    <feature var='jabber:iq:time'/>
    <feature var='jabber:iq:version'/>
  </query>
</iq>
```

If the JID of the specified target entity does not exist, the server or other authoritative entity SHOULD return an <item-not-found/> error, unless doing so would violate the privacy and security considerations specified in XMPP Core and XMPP IM [6] or local privacy and security policies (see also the Security Considerations of this document):

Listing 3: Target entity does not exist

```
<iq type='error'
    from='plays.shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If privacy and security considerations or policies prevent the server or other authoritative entity from returning an <item-not-found/> error, it SHOULD return a <service-unavailable/>

[6] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.

error instead:

Listing 4: Service unavailable

```
<iq type='error'
    from='plays.shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
  <error type='cancel'>
    <service-unavailable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

When an entity sends a disco#info request to a bare JID (<account@domain.tld>) hosted by a server, the server itself MUST reply on behalf of the hosted account, either with an IQ-error or an IQ-result. For important rules regarding access to this functionality, see the Security Considerations section of this document. In particular, in response to a disco#info request sent to a bare JID with no node, if access is not denied the server SHOULD return an IQ-result for the bare JID, in which the primary identity SHOULD have a category of "account" with an appropriate type as specified in the Service Discovery Identities registry (most likely, a type of "registered"). Note: This enables authorized or trusted entities to discover whether the account exists and its account type (e.g., in IM systems to determine the existence of an account before adding it to a contact's roster).

Listing 5: Requesting info from a bare JID

```
<iq type='get'
    from='shakespeare.lit'
    to='juliet@capulet.com'
    id='info2'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Here we assume that shakespeare.lit is trusted by capulet.com and that the account <juliet@capulet.com> is a registered account:

Listing 6: Server replies on behalf of bare JID

```
<iq type='result'
    from='juliet@capulet.com'
    to='shakespeare.lit'
    id='info2'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='account' type='registered'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
  </query>
</iq>
```

A query sent to an associated entity may result in different or more detailed information. One example is sending a query to a particular conference room rather than the parent conference service:

Listing 7: Querying a specific conference room

```
<iq type='get'
    from='juliet@capulet.com/balcony'
    to='balconyscene@plays.shakespeare.lit'
    id='info3'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq type='result'
    from='balconyscene@plays.shakespeare.lit'
    to='juliet@capulet.com/balcony'
    id='info3'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='conference'
        name='A Dark Cave'
        type='text'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/muc'/>
    <feature var='muc_passwordprotected'/>
    <feature var='muc_hidden'/>
    <feature var='muc_temporary'/>
    <feature var='muc_open'/>
    <feature var='muc_unmoderated'/>
    <feature var='muc_nonanonymous'/>
  </query>
</iq>
```

Another example of this is sending a query to a specific connected resource for an IM user:

Listing 8: Querying a connected resource for further information

```
<iq type='get'
    from='juliet@capulet.com/balcony'
    to='romeo@montague.net/orchard'
    id='info4'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq type='result'
    from='romeo@montague.net/orchard'
    to='juliet@capulet.com/balcony'
    id='info4'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
```

```
    <identity
        category='client'
        type='pc'
        name='Gabber'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='jabber:iq:time'/>
    <feature var='jabber:iq:version'/>
  </query>
</iq>
```

3.2 Info Nodes

A disco#info query MAY also be directed to a specific node identifier associated with a JID, although the primary use of nodes is as Items Nodes rather than as info nodes:

Listing 9: Querying a specific JID and node combination

```
<iq type='get'
    from='romeo@montague.net/orchard'
    to='mim.shakespeare.lit'
    id='info3'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://jabber.org/protocol/commands'/>
</iq>
```

If the request included a 'node' attribute, the response MUST mirror the specified 'node' attribute to ensure coherence between the request and the response.

Listing 10: JID+node result

```
<iq type='result'
    from='mim.shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='info3'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://jabber.org/protocol/commands'>
    <identity
        category='automation'
        type='command-list'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
  </query>
</iq>
```

4 Discovering the Items Associated with a Jabber Entity

4.1 Basic Protocol

In order for the requesting entity to discover the items associated with a Jabber Entity, it MUST send an IQ stanza of type "get" to the target entity, containing an empty <query/> element qualified by the 'http://jabber.org/protocol/disco#items' namespace:

Listing 11: Requesting all items

```
<iq type='get'
    from='romeo@montague.net/orchard'
    to='shakespeare.lit'
    id='items1'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

The target entity then MUST either return its list of publicly-available items, or return an error. The list of items MUST be provided in an IQ stanza of type "result", with each item specified by means of an <item/> child of a <query/> element qualified by the 'http://jabber.org/protocol/disco#items' namespace (the <item/> child MUST possess a 'jid' attribute specifying the JID of the item and MAY possess a 'name' attribute specifying a natural-language name for the item):

Listing 12: Result-set for all items

```
<iq type='result'
    from='shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='items1'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='people.shakespeare.lit'
          name='Directory of Characters'/>
    <item jid='plays.shakespeare.lit'
          name='Play-Specific Chatrooms'/>
    <item jid='mim.shakespeare.lit'
          name='Gateway to Marlowe IM'/>
    <item jid='words.shakespeare.lit'
          name='Shakespearean Lexicon'/>
    <item jid='globe.shakespeare.lit'
          name='Calendar of Performances'/>
    <item jid='headlines.shakespeare.lit'
          name='Latest Shakespearean News'/>
    <item jid='catalog.shakespeare.lit'
          name='Buy Shakespeare Stuff!'/>
    <item jid='en2fr.shakespeare.lit'
          name='French Translation Service'/>
  </query>
```

```
</iq>
```

The <item/> element MUST NOT contain XML character data and SHOULD be empty; while it MAY contain XML data in another namespace, such data MUST be ignored if an implementation does not understand it.
If there are no items associated with an entity (or if those items are not publicly available), the target entity MUST return an empty query element to the requesting entity:

Listing 13: Empty result set

```
<iq type='result'
    from='shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='items1'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

As with disco#info requests, when an entity sends a disco#items request to a bare JID (<account@domain.tld>) hosted by a server, the server itself MUST reply on behalf of the hosted account. For important rules regarding access to this functionality, see the Security Considerations section of this document. In particular, in response to a disco#items request sent to a bare JID with no node, if access is not denied the server SHOULD return the associated items including connected or available resources as appropriate:

Listing 14: Requesting items from a bare JID

```
<iq type='get'
    from='shakespeare.lit'
    to='juliet@capulet.com'
    id='items2'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

Here we assume that shakespeare.lit is trusted by capulet.com and that the account <juliet@capulet.com> has two available resources:

Listing 15: Server replies on behalf of bare JID

```
<iq type='result'
    from='juliet@capulet.com'
    to='shakespeare.lit'
    id='items2'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='juliet@capulet.com/balcony'/>
    <item jid='juliet@capulet.com/chamber'/>
  </query>
</iq>
```

4.2 Items Nodes

It is possible that an item associated with an entity will not be addressable as a JID; examples might include offline messages stored in an inbox (see Flexible Offline Message Retrieval (XEP-0013) [7]), entries in a Jabber-enabled weblog, XML-RPC services associated with a client or component, items available in an online trading system (e.g., a catalog or auction), news postings located at an NNTP gateway, and topics hosted by a Publish-Subscribe (XEP-0060) [8] component. In order to handle such items, the <item/> element MAY possess an OPTIONAL 'node' attribute that supplements the REQUIRED 'jid' attribute.
The value of the node attribute may or may not have semantic meaning; from the perspective of Service Discovery, a node is merely something that is associated with an entity. In order to discover more about the node, the requesting entity MUST query the entity's JID while specifying the node. If the value of the 'node' attribute has semantic meaning, that meaning is provided by the "using protocol" or application, not by the Service Discovery protocol. A node attribute SHOULD NOT be included unless it is necessary to provide or discover information about an entity that cannot be directly addressed as a JID (i.e., if the associated item can be addressed as a JID, do not include a node). The value of the 'node' attribute MUST NOT be null.
In the following example, a user requests all available items from an online catalog service:

Listing 16: Requesting nodes

```
<iq type='get'
    from='romeo@montague.net/orchard'
    to='catalog.shakespeare.lit'
    id='items2'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

If there are items associated with the target entity but they are not addressable as JIDs, the service SHOULD then return a list of nodes (where each <item/> element MUST possess a 'jid' attribute, SHOULD possess a 'node' attribute, and MAY possess a 'name' attribute):

Listing 17: Service returns nodes

```
<iq type='result'
    from='catalog.shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='items2'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='catalog.shakespeare.lit'
          node='books'
          name='Books by and about Shakespeare'/>
    <item jid='catalog.shakespeare.lit'
          node='clothing'
```

[7] XEP-0013: Flexible Offline Message Retrieval <https://xmpp.org/extensions/xep-0013.html>.
[8] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.

```
          name='Wear your literary taste with pride'/>
    <item jid='catalog.shakespeare.lit'
          node='music'
          name='Music from the time of Shakespeare'/>
  </query>
</iq>
```

There may be futher nodes associated with the "first-level" nodes returned in the above query (e.g., the nodes may be categories that have associated items). The requesting entity can query a node further by sending a request to the JID and specifying the node of interest in the query.

Listing 18: Requesting further nodes

```
<iq type='get'
    from='romeo@montague.net/orchard'
    to='catalog.shakespeare.lit'
    id='items3'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='music'/>
</iq>
```

The service then returns the further nodes associated with the "parent" node. In the following example, the service itself enforces an alphabetically-ordered hierarchical structure on the nodes that are returned, but such a structure is a matter of implementation rather than protocol.

Listing 19: Service returns further nodes

```
<iq type='result'
    from='catalog.shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='items3'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='music'>
    <item jid='catalog.shakespeare.lit'
          node='music/A'/>
    <item jid='catalog.shakespeare.lit'
          node='music/B'/>
    <item jid='catalog.shakespeare.lit'
          node='music/C'/>
    <item jid='catalog.shakespeare.lit'
          node='music/D'/>
    .
    .
    .
  </query>
</iq>
```

The requesting entity can then query further if desired:

Listing 20: Requesting even more nodes

```
<iq type='get'
    from='romeo@montague.net/orchard'
    to='catalog.shakespeare.lit'
    id='items4'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='music/D'/>
</iq>
```

Listing 21: Service returns even more nodes

```
<iq type='result'
    from='catalog.shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='items4'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='music/D'>
    <item jid='catalog.shakespeare.lit'
          node='music/D/dowland-firstbooke'
          name='John Dowland - First Booke of Songes or Ayres'/>
    <item jid='catalog.shakespeare.lit'
          node='music/D/dowland-solace'
          name='John Dowland - A Pilgrimes Solace'/>
  </query>
</iq>
```

4.3 Node Hierarchies

The foregoing examples show a hierarchy of nodes, in which some nodes are branches (i.e., contain further nodes) and some nodes are leaves (i.e., do not contain further nodes). The "hierarchy" category SHOULD be used to identify such nodes, where the "branch" and "leaf" types are exhaustive of the types within this category.
If the hierarchy category is used, every node in the hierarchy MUST be identified as either a branch or a leaf; however, since a node MAY have multiple identities, any given node MAY also possess an identity other than "hierarchy/branch" or "hierarchy/leaf".
Therefore, a disco#info request to the "music/D" node shown above would yield <identity category='hierarchy' type='branch'/> while a disco#info request to the "music/D/dowland-firstbooke" node would yield <identity category='hierarchy' type='leaf'/> (and each node could yield additional identities as appropriate).

4.4 Relationship Between an Entity and its Items

This section explains in greater detail the relationship between an entity and its associated items.
In general, the items returned by an entity in a disco#items result MUST be items over which the entity has some relationship of ownership -- either direct control over the item itself (e.g., Publish-Subscribe nodes owned by the entity) or at least the ability to provide or vouch for the item in a canonical way on the Jabber network (e.g., groupchat rooms directly hosted by a multi-user chat service or IRC channels to which a gateway provides access).
Such a relationship does not constrain the relationship between the owning entity's address and the address of the associated entity. In particular, any of the following scenarios is perfectly acceptable:

1. Upon querying an entity (JID1) for items, one receives a list of items that can be addressed as JIDs; each associated item has its own JID, but no such JID equals JID1.

2. Upon querying an entity (JID1) for items, one receives a list of items that cannot be addressed as JIDs; each associated item has its own JID+node, where each JID equals JID1 and each NodeID is unique.

3. Upon querying an entity (JID1+NodeID1) for items, one receives a list of items that can be addressed as JIDs; each associated item has its own JID, but no such JID equals JID1.

4. Upon querying an entity (JID1+NodeID1) for items, one receives a list of items that cannot be addressed as JIDs; each associated item has its own JID+node, but no such JID+node equals JID1+NodeID1 and each NodeID is unique in the context of the associated JID.

In addition, the results MAY also be mixed, so that a query to a JID or a JID+node could yield both (1) items that are addressed as JIDs and (2) items that are addressed as JID+node combinations.
Consider the case of an entity that owns multiple publish-subscribe nodes -- for example, a person who owns one such node for each of his music players. The following examples show what the disco#items query and result might look like (using the protocol defined in User Tune (XEP-0118) [9]):

Listing 22: User queries entity regarding tunes

```
<iq from='juliet@capulet.com/chamber'
    id='items4'
```

[9] XEP-0118: User Tune <https://xmpp.org/extensions/xep-0118.html>.

```
    to='romeo@montague.net'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://jabber.org/protocol/tune'/>
</iq>
```

The queried entity now returns a list of publish-subscribe nodes over which it has control, each of which is hosted on a different pubsub service:

Listing 23: Entity returns multiple items

```
<iq from='romeo@montague.net'
    id='items4'
    to='juliet@capulet.com/chamber'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'
         node='http://jabber.org/protocol/tune'>
    <item jid='pubsub.shakespeare.lit'
          name='Romeo&apos;s CD player'
          node='s623nms9s3bfh8js'/>
    <item jid='pubsub.montague.net'
          node='music/R/Romeo/iPod'/>
    <item jid='tunes.characters.lit'
          node='g8k4kds9sd89djf3'/>
  </query>
</iq>
```

5 Publishing Available Items

This feature has been removed in favor of the XMPP publish-subscribe technology defined in XEP-0060.

6 Implementation Notes

6.1 Number of Info Requests

When the requesting application is a client, it may want to retrieve service discovery information about all of a user's contacts after retrieving the user's roster and receiving presence from contacts in the user's roster (e.g., to show capabilities). Unfortunately, a user's roster can be quite large, resulting in sending a large number of outbound disco#info requests and receiving a large number of inbound disco#info responses upon login. Because this "disco flood" is undesirable for reasons of scalability and bandwidth usage, client applications SHOULD use Entity Capabilities (XEP-0115) [10] to determine the capabilities of entities from

[10] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

which they receive presence information and SHOULD NOT send disco#info requests to such entities.

6.2 Number of Items Requests

In order to retrieve full information about an entity and its associated items, the requesting application needs to "walk the tree" of items. Naturally, this can result in a large number of requests and responses. The requesting application SHOULD NOT send follow-up requests to all items associated with an entity if the list of such items is long (e.g., more than twenty items). Entities that will routinely host a large number of items (e.g., IRC gateways or NNTP services) SHOULD structure nodes into hierarchies and/or provide more robust searching capabilities, for example via Jabber Search (XEP-0055) [11]; they SHOULD NOT return extremely large result sets via Service Discovery.

6.3 Response Consistency

This document recommends but does not require that a responding entity must return the same results in response to the same request from different requesting entities (e.g., an entity could return a different list of items or features based on the degree to which it trusts the requesting entity, or based on the known capabilities of the requesting entity). However, the responding entity SHOULD return the same <identity/> element (category+type) to all disco#info requests sent to the same JID+node combination.

7 Error Conditions

If a specific entity (JID or JID+node) does not support the disco namespace, refuses to return disco results to the specific requesting entity, or refuses to return disco results to any requesting entity, it SHOULD return an appropriate error message (such as <service-unavailable/>, <forbidden/>, or <not-allowed/>, respectively). One example is shown below.

Listing 24: JID+node error

```
<iq type='error'
    from='mim.shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='info3'>
  <query xmlns='http://jabber.org/protocol/disco#info'
         node='http://jabber.org/protocol/commands'/>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
```

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

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

Other error conditions may be appropriate depending on the application.
The following table summarizes the common error conditions that can have special meaning in the context of Service Discovery (for information regarding error condition syntax and semantics, see Error Condition Mappings (XEP-0086) [12]).

Condition	Cause
<item-not-found/>	The JID or JID+NodeID of the specified target entity does not exist and that fact can be divulged in accordance with privacy and security considerations and policies.
<service-unavailable/>	The target entity does not support this protocol, or the specified target entity does not exist but that fact cannot be divulged because of privacy and security considerations.

The other error conditions specified in XMPP Core MAY be returned as well (<forbidden/>, <not-allowed/>, <not-authorized/>, etc.), including application-specific conditions.
As noted above, if an entity has no associated items, it MUST return an empty <query/> element (rather than an error) in response to a disco#items request.

8 Security Considerations

Certain attacks may be made easier when an entity discloses (via disco#info responses) that it supports particular protocols or features; however, in general, service discovery introduces no new vulnerabilities, since a malicious entity could discover that the responding entity supports such protocols and features by sending requests specific to those protocols rather than by sending service discovery requests.
A responding entity is under no obligation to return the identical service discovery response when replying to service discovery requests received from different requesting entities, and MAY perform authorization checks before responding in order to determine how (or whether) to respond.
A server MUST carefully control access to any functionality that would enable directory harvesting attacks or that would leak information about connected or available resources; this functionality consists of the server's replies to disco#info and disco#items requests sent to bare JIDs (addresses of the form account@domain.tld) hosted on the server, since the server responds to such requests on behalf of the account. The following rules apply to the handling of service discovery requests sent to bare JIDs:

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

1. In response to a disco#info request, the server MUST return a <service-unavailable/> error if one of the following is true:

   a) The target entity does not exist (no matter if the request specifies a node or not).

   b) The requesting entity is not authorized to receive presence from the target entity (i.e., via the target having a presence subscription to the requesting entity of type "both" or "from") or is not otherwise trusted (e.g., another server in a trusted network).

2. In response to a disco#items request, the server MUST return an empty result set if:

   a) The target entity does not exist (no matter if the request specifies a node or not).

   b) The request did not specify a node, the only items are available resources (as defined in RFC 3921), and the requesting entity is not authorized to receive presence from the target entity (i.e., via the target having a presence subscription to the requesting entity of type "both" or "from") or is not otherwise trusted (e.g., another server in a trusted network). [13]

9 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [14].

10 XMPP Registrar Considerations

10.1 Protocol Namespaces

The XMPP Registrar [15] includes the 'http://jabber.org/protocol/disco#info' and 'http://jabber.org/protocol/disco#items' namespaces in its registry of protocol namespaces.

[13] However, the server MAY return items other than available resources (if any).

[14] 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/>.

[15] 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/>.

10.2 Registries

10.2.1 Identity Categories and Types Registry

The XMPP Registrar maintains a registry of values for the 'category' and 'type' attributes of the <identity/> element in the 'http://jabber.org/protocol/disco#info' namespace; see <https://xmpp.org/registrar/disco-categories.html>.
In order to submit new values to this registry, the registrant shall define an XML fragment of the following form and either include it in the relevant XMPP Extension Protocol or send it to the email address <registrar@xmpp.org>:

```
<category>
  <name>the name of the category (all lower-case)</name>
  <desc>a natural-language description of the category</desc>
  <type>
    <name>the name of the specific type (all lower-case)</name>
    <desc>a natural-language description of the type</desc>
    <doc>the document (e.g., XEP) in which this type is specified</doc
        >
  </type>
</category>
```

The registrant may register more than one category at a time, each contained in a separate <category/> element. The registrant may also register more than one type at a time, each contained in a separate <type/> child element. Registrations of new types within an existing category must include the full XML snippet but should not include the category description (only the name).
This document defines a "hierarchy" category that contains two and only two types: "branch" and "leaf"; the associated registry submission is as follows:

```
<category>
  <name>hierarchy</name>
  <desc>
    An entity that exists in the context of a
    service discovery node hierarchy.
  </desc>
  <type>
    <name>branch</name>
    <desc>
      A "container node" for other entities in a
      service discovery node hierarchy.
    </desc>
    <doc>XEP-0030</doc>
  </type>
  <type>
    <name>leaf</name>
    <desc>
```

```
      A "terminal node" in a service discovery
      node hierarchy.
    </desc>
    <doc>XEP-0030</doc>
  </type>
</category>
```

10.2.2 Features Registry

The XMPP Registrar maintains a registry of features for use as values of the 'var' attribute of the <feature/> element in the 'http://jabber.org/protocol/disco#info' namespace; see <https://xmpp.org/registrar/disco-features.html>.
In order to submit new values to this registry, the registrant shall define an XML fragment of the following form and either include it in the relevant XMPP Extension Protocol or send it to the email address <registrar@xmpp.org>:

```
<var>
  <name>name of feature or namespace</name>
  <desc>a natural-language description of the feature</desc>
  <doc>the document (e.g., XEP) in which this feature is specified</
     doc>
</var>
```

The registrant may register more than one feature at a time, each contained in a separate <feature/> element.

10.2.3 Well-Known Nodes

A "using protocol" may specify one or more service discovery nodes that have a special and well-defined meaning in the context of that protocol. For the purpose of reserving these node names globally across all Jabber protocols, the XMPP Registrar maintains a registry of well-known service discovery nodes at <https://xmpp.org/registrar/nodes.html>.
In order to submit new values to this registry, the registrant shall define an XML fragment of the following form and either include it in the relevant XMPP Extension Protocol or send it to the email address <registrar@xmpp.org>:

```
<node>
  <name>the name of the node</name>
  <desc>a natural-language description of the node</desc>
  <doc>the document (e.g., XEP) in which this node is specified</doc>
</node>
```

The registrant may register more than one node at a time, each contained in a separate <node/> element.

10.3 URI Query Types

As authorized by XMPP URI Query Components (XEP-0147) [16], 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 "disco" querytype is defined herein for service discovery interactions, with three keys: (1) "node" (the optional node to query), (2) "request" (with values of "info" to retrieve service discovery information and "items" to retrieve service discovery items), and (3) "type" (with values of "get" for IQ-gets and "set" for IQ-sets).

Listing 25: Service Discovery Information Request: IRI/URI

```
xmpp:romeo@montague.net?disco;type=get;request=info
```

Listing 26: Service Discovery Information Request: Resulting Stanza

```
<iq to='romeo@montague.net' type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 27: Service Discovery Items Request: IRI/URI

```
xmpp:romeo@montague.net?disco;type=get;request=items
```

Listing 28: Service Discovery Items Request: Resulting Stanza

```
<iq to='romeo@montague.net' type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

The following submission registers the "disco" querytype.

```
<querytype>
  <name>disco</name>
  <proto>http://jabber.org/protocol/disco</proto>
  <desc>enables interaction for the purpose of service discovery</desc
     >
  <doc>XEP-0030</doc>
  <keys>
    <key>
      <name>node</name>
      <desc>the (optional) service discovery node</desc>
    </key>
    <key>
      <name>request</name>
      <desc>the service discovery request type</desc>
```

[16] XEP-0147: XMPP URI Query Components <https://xmpp.org/extensions/xep-0147.html>.

```
      <values>
        <value>
          <name>info</name>
          <desc>a service discovery information (disco#info) request</
              desc>
        </value>
        <value>
          <name>items</name>
          <desc>a service discovery items (disco#items) request</desc>
        </value>
      </values>
    </key>
    <key>
      <name>type</name>
      <desc>the IQ type</desc>
      <values>
        <value>
          <name>get</name>
          <desc>an IQ get</desc>
        </value>
      </values>
    </key>
  </keys>
</querytype>
```

11 XML Schemas

11.1 disco#info

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

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

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0030: http://www.xmpp.org/extensions/xep-0030.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='query'>
    <xs:complexType>
      <xs:sequence minOccurs='0'>
```

```
        <xs:element ref='identity' maxOccurs='unbounded'/>
        <xs:element ref='feature' maxOccurs='unbounded'/>
      </xs:sequence>
      <xs:attribute name='node' type='xs:string' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='identity'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='empty'>
          <xs:attribute name='category' type='nonEmptyString' use='
              required'/>
          <xs:attribute name='name' type='xs:string' use='optional'/>
          <xs:attribute name='type' type='nonEmptyString' use='
              required'/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

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

  <xs:simpleType name='nonEmptyString'>
    <xs:restriction base='xs:string'>
      <xs:minLength value='1'/>
    </xs:restriction>
  </xs:simpleType>

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

</xs:schema>
```

11.2 disco#items

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

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

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0030: http://www.xmpp.org/extensions/xep-0030.html
    </xs:documentation>
  </xs:annotation>

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

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

  <xs:simpleType name='fullJIDType'>
    <xs:restriction base='xs:string'>
      <xs:minLength value='8'/>
      <xs:maxLength value='3071'/>
    </xs:restriction>
  </xs:simpleType>

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

</xs:schema>
```

12 Author Note

Peter Millard, a co-author of this specification from version 0.1 through version 2.2, died on April 26, 2006.