XEP-0077: In-Band Registration

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

2012-01-25
Version 2.4

Status	Type	Short Name
Final	Standards Track	iq-register

This specification defines an XMPP protocol extension for in-band registration with XMPP-based instant messaging servers and other services hosted on an XMPP network (such as groupchat rooms and gateways to non-XMPP IM services). The protocol is extensible via use of data forms, thus enabling services to gather a wide range of information during the registration process. The protocol also supports in-band password changes and cancellation of an existing registration.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

The Jabber protocols have long included a method for in-band registration with instant messaging servers and associated services. This method makes use of the 'jabber:iq:register' namespace and has been documented variously in Internet-Drafts and elsewhere. Because in-band registration is not required by RFC 2779 [1], documentation of the 'jabber:iq:register' namespace was removed from XMPP IM [2]. This specification fills the void for canonical documentation.

2 Requirements

In-band registration must make it possible for an entity to register with a host, cancel an existing registration with a host, or change a password with a host. By "host" is meant either of the following:

1. an instant messaging server, such as described in XMPP IM and identified by the "server/im" Service Discovery (XEP-0030) [3] category+type
2. an add-on service, such as a gateway (see Gateway Interaction (XEP-0100) [4]) or a Multi-User Chat (XEP-0045) [5] service

If needed, extensibility with regard to information gathered should be done using Data Forms (XEP-0004) [6].

3 Use Cases

3.1 Entity Registers with a Host

In order to determine which fields are required for registration with a host, an entity SHOULD first send an IQ get to the host. The entity SHOULD NOT attempt to guess at the required fields by first sending an IQ set, since the nature of the required data is subject to service provisioning.

Listing 1: Entity Requests Registration Fields from Host

[1] RFC 2779: A Model for Presence and Instant Messaging <http://tools.ietf.org/html/rfc2779>.
[2] RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence <http://tools.ietf.org/html/rfc6121>.
[3] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[4] XEP-0100: Gateway Interaction <https://xmpp.org/extensions/xep-0100.html>.
[5] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[6] XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.

```
<iq type='get' id='reg1' to='shakespeare.lit'>
  <query xmlns='jabber:iq:register'/>
</iq>
```

If the entity is not already registered and the host supports In-Band Registration, the host MUST inform the entity of the required registration fields. If the host does not support In-Band Registration, it MUST return a <service-unavailable/> error. If the host is redirecting registration requests to some other medium (e.g., a website), it MAY return an <instructions/> element only, as shown in the Redirection section of this document.

Listing 2: Host Returns Registration Fields to Entity

```
<iq type='result' id='reg1'>
  <query xmlns='jabber:iq:register'>
    <instructions>
      Choose a username and password for use with this service.
      Please also provide your email address.
    </instructions>
    <username/>
    <password/>
    <email/>
  </query>
</iq>
```

If the host determines (based on the 'from' address) that the entity is already registered, the IQ result that it sends in response to the IQ get MUST contain an empty <registered/> element (indicating that the entity is already registered), SHOULD contain the registration information currently on file for the entity (although the <password/> element MAY be empty), and SHOULD contain an <instructions/> element (whose XML character data MAY be modified to reflect the fact that the entity is currently registered).
If the host is an instant messaging server, it SHOULD assume that the requesting entity is unregistered at this stage unless the entity has already authenticated (for details, see the Registration with a Server section below).

Listing 3: Host Informs Entity of Current Registration

```
<iq type='result' id='reg1'>
  <query xmlns='jabber:iq:register'>
    <registered/>
    <username>juliet</username>
    <password>R0m30</password>
    <email>juliet@capulet.com</email>
  </query>
</iq>
```

If the entity is not already registered, the entity SHOULD provide the required information.

Listing 4: Entity Provides Required Information

```
<iq type='set' id='reg2'>
  <query xmlns='jabber:iq:register'>
    <username>bill</username>
    <password>Calliope</password>
    <email>bard@shakespeare.lit</email>
  </query>
</iq>
```

Note: The requesting entity MUST provide information for all of the elements (other than <instructions/>) contained in the IQ result.

Listing 5: Host Informs Entity of Successful Registration

```
<iq type='result' id='reg2'/>
```

Alternatively, registration may fail. Possible causes of failure include a username conflict (the desired username is already in use by another entity) and the fact that the requesting entity neglected to provide all of the required information.

Listing 6: Host Informs Entity of Failed Registration (Username Conflict)

```
<iq type='error' id='reg2'>
  <query xmlns='jabber:iq:register'>
    <username>bill</username>
    <password>m1cro$oft</password>
    <email>billg@bigcompany.com</email>
  </query>
  <error code='409' type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

If the requesting entity does not provide all of the requested information during registration [7] then the server or service MUST return a <not-acceptable/> error to the requesting entity.

Listing 7: Host Informs Entity of Failed Registration (Some Required Information Not Provided)

```
<iq type='error' id='reg2'>
  <query xmlns='jabber:iq:register'>
    <username>bill</username>
    <password>Calliope</password>
  </query>
  <error code='406' type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
```

[7] This includes providing an empty password element or a password element that contains no XML character data, i.e., either <password/> or <password></password>.

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

If the host requires additional information above and beyond the data elements specified in the schema, it SHOULD use Data Forms as described in the Extensibility section of this document. (The next three examples show registration of an existing account with a third-party service, not of an entity with a server for the purpose of account registration.)

Listing 8: Entity Requests Registration Fields from Host

```
<iq type='get'
    from='juliet@capulet.com/balcony'
    to='contests.shakespeare.lit'
    id='reg3'>
  <query xmlns='jabber:iq:register'/>
</iq>
```

Listing 9: Host Returns Registration Form to Entity

```
<iq type='result'
    from='contests.shakespeare.lit'
    to='juliet@capulet.com/balcony'
    id='reg3'>
  <query xmlns='jabber:iq:register'>
    <instructions>
      Use the enclosed form to register. If your Jabber client does
         not
      support Data Forms, visit http://www.shakespeare.lit/contests.
         php
    </instructions>
    <x xmlns='jabber:x:data' type='form'>
      <title>Contest Registration</title>
      <instructions>
        Please provide the following information
        to sign up for our special contests!
      </instructions>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:register</value>
      </field>
      <field type='text-single' label='Given Name' var='first'>
        <required/>
      </field>
      <field type='text-single' label='Family Name' var='last'>
        <required/>
      </field>
      <field type='text-single' label='Email Address' var='email'>
        <required/>
      </field>
      <field type='list-single' label='Gender' var='x-gender'>
```

```
        <option label='Male'><value>M</value></option>
        <option label='Female'><value>F</value></option>
      </field>
    </x>
  </query>
</iq>
```

The user then SHOULD return the form:

Listing 10: User Submits Registration Form

```
<iq type='set'
    from='juliet@capulet.com/balcony'
    to='contests.shakespeare.lit'
    id='reg4'>
  <query xmlns='jabber:iq:register'>
    <x xmlns='jabber:x:data' type='submit'>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:register</value>
      </field>
      <field type='text-single' label='Given Name' var='first'>
        <value>Juliet</value>
      </field>
      <field type='text-single' label='Family Name' var='last'>
        <value>Capulet</value>
      </field>
      <field type='text-single' label='Email Address' var='email'>
        <value>juliet@capulet.com</value>
      </field>
      <field type='list-single' label='Gender' var='x-gender'>
        <value>F</value>
      </field>
    </x>
  </query>
</iq>
```

3.1.1 Registration with a Server

Special care must be taken when an unregistered entity interacts with a server rather than a service. Normally, a server enables in-band registration so that entities can "bootstrap" their participation in the Jabber network; this bootstrapping happens when an unregistered and unauthenticated entity opens a TCP connection to a server and immediately completes the registration use case with the server, then authenticates using the newly-registered identity. As noted, when a server receives an IQ-get for registration information, it SHOULD assume that the requesting entity is unregistered unless the entity has already authenticated.
Depending on local service provisioning, a server MAY return a <not-acceptable/> stanza error if an unregistered entity attempts to register too many times before authenticating or if

an entity attempts to register a second identity after successfully completing the registration use case; a server MAY also return a <not-authorized/> stream error if the unregistered entity waits too long before authenticating or attempts to complete a task other than authentication after successfully completing the registration use case.

3.2 Entity Cancels an Existing Registration

The 'jabber:iq:register' namespace also makes it possible for an entity to cancel a registration with a host by sending a <remove/> element in an IQ set. The host MUST determine the identity of the requesting entity based on the 'from' address of the IQ get.

Listing 11: Entity Requests Cancellation of Existing Registration

```
<iq type='set' from='bill@shakespeare.lit/globe' id='unreg1'>
  <query xmlns='jabber:iq:register'>
    <remove/>
  </query>
</iq>
```

Listing 12: Host Informs Entity of Successful Cancellation

```
<iq type='result' to='bill@shakespeare.lit/globe' id='unreg1'/>
```

There are two scenarios:

1. If the entity cancels its registration with its "home" server (i.e., the server at which it has maintained its XMPP account), then the entity SHOULD NOT include a 'from' or 'to' address in the remove request the server SHOULD then return a <not-authorized/> stream error and terminate all active sessions for the entity. The server SHOULD perform the remove based on the bare JID <localpart@domain.tld> associated with the current session or connection over which it received the remove request. If the server is an instant messaging and presence server that conforms to XMPP IM [8], the server SHOULD also cancel all existing presence subscriptions related to that entity (as stored in the entity's roster).

2. If the entity cancels its registration with a service other than its home server, its home server MUST stamp a 'from' address on the remove request, which in accordance with XMPP Core will be the entity's full JID <localpart@domain.tld/resource>. The service MUST perform the remove based on the bare JID <localpart@domain.tld> portion of the 'from' address.

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

As specified below, several error cases are possible.

Condition	Description
<bad-request/>	The <remove/> element was not the only child element of the <query/> element.
<forbidden/>	The sender does not have sufficient permissions to cancel the registration.
<not-allowed/>	No sender is allowed to cancel registrations in-band.
<registration-required/>	The entity sending the remove request was not previously registered.
<unexpected-request/>	The host is an instant messaging server and the IQ get does not contain a 'from' address because the entity is not registered with the server.

Listing 13: Host Informs Client of Failed Cancellation (Bad Request)

```
<iq type='error' from='shakespeare.lit' to='bill@shakespeare.lit/globe
    ' id='unreg1'>
  <error code='400' type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Listing 14: Host Informs Client of Failed Cancellation (Forbidden)

```
<iq type='error' from='shakespeare.lit' to='bill@shakespeare.lit/globe
    ' id='unreg1'>
  <error code='401' type='cancel'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Listing 15: Server Informs Client of Failed Cancellation (Not Allowed)

```
<iq type='error' from='shakespeare.lit' to='bill@shakespeare.lit/globe
    ' id='unreg1'>
  <error code='405' type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

A given deployment MAY choose to require additional information (such as the old password or previously-gathered personal information) before cancelling the registration. In order to do so, the server or service SHOULD return a Data Form in the error stanza:

Listing 16: Server Returns Cancellation Form With Error

```
<iq type='error' from='shakespeare.lit' to='bill@shakespeare.lit/globe
    ' id='unreg1'>
  <query xmlns='jabber:iq:register'>
    <x xmlns='jabber:x:data' type='form'>
      <title>Cancel Registration</title>
      <instructions>Use this form to cancel your registration.</
          instructions>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:register:cancel</value>
      </field>
      <field type='text-single' label='Username' var='username'>
        <required/>
      </field>
      <field type='text-private' label='Password' var='password'>
        <required/>
      </field>
      <field type='text-single' label='Mother&apos;s Maiden Name' var=
          'x-mmn'>
        <required/>
      </field>
    </x>
  </query>
  <error code='405' type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

The user then SHOULD return the form:

Listing 17: User Returns Cancellation Form

```
<iq type='set' from='bill@shakespeare.lit/globe' to='shakespeare.lit'
    id='unreg2'>
  <query xmlns='jabber:iq:register'>
    <x xmlns='jabber:x:data' type='submit'>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:register:cancel</value>
      </field>
      <field type='text-single' var='username'>
        <value>bill@shakespeare.lit</value>
      </field>
      <field type='text-private' var='password'>
        <value>theglobe</value>
      </field>
      <field type='text-single' var='x-mmn'>
        <value>Throckmorton</value>
      </field>
    </x>
```

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

3.3 User Changes Password

The 'jabber:iq:register' namespace enables a user to change his or her password with a server or service. Once registered, the user can change passwords by sending an XML stanza of the following form:

Listing 18: Password Change

```
<iq type='set' to='shakespeare.lit' id='change1'>
  <query xmlns='jabber:iq:register'>
    <username>bill</username>
    <password>newpass</password>
  </query>
</iq>
```

Because the password change request contains the password in plain text, a client SHOULD NOT send such a request unless the underlying stream is encrypted (using SSL or TLS) and the client has verified that the server certificate is signed by a trusted certificate authority. A given deployment MAY choose to disable password changes if the stream is not properly encrypted, to disable in-band password changes, or to require additional information (such as the old password or previously-gathered personal information) before changing the password.
If the user provides an empty password element or a password element that contains no XML character data (i.e., either <password/> or <password></password>), the server or service MUST NOT change the password to a null value, but instead MUST maintain the existing password.

Listing 19: Host Informs Client of Successful Password Change

```
<iq type='result' id='change1'/>
```

Several error conditions are possible:

Condition	Description
<bad-request/>	The password change request does not contain complete information (e.g., the service requires inclusion of the <username/> element).
<not-authorized/>	The server or service does not consider the channel safe enough to enable a password change.
<not-allowed/>	The server or service does not allow password changes.
<unexpected-request/>	The host is an instant messaging server and the IQ set does not contain a 'from' address because the entity is not registered with the server.

The server or service SHOULD NOT return the original XML sent in IQ error stanzas related to password changes.

Listing 20: Host Informs Client of Failed Password Change (Bad Request)

```
<iq type='error' from='shakespeare.lit' to='bill@shakespeare.lit/globe
    ' id='change1'>
  <error code='400' type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Listing 21: Host Informs Client of Failed Password Change (Not Authorized)

```
<iq type='error' from='shakespeare.lit' to='bill@shakespeare.lit/globe
    ' id='change1'>
  <error code='401' type='modify'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Listing 22: Server Informs Client of Failed Password Change (Not Allowed)

```
<iq type='error' from='shakespeare.lit' to='bill@shakespeare.lit/globe
    ' id='change1'>
  <error code='405' type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

In order to require additional information before changing the password, the server or service SHOULD return a Data Form in the error stanza:

Listing 23: Server Returns Password Change Form With Error

```
<iq type='error' from='shakespeare.lit' to='bill@shakespeare.lit/globe
    ' id='change1'>
  <query xmlns='jabber:iq:register'>
    <x xmlns='jabber:x:data' type='form'>
      <title>Password Change</title>
      <instructions>Use this form to change your password.</
          instructions>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:register:changepassword</value>
      </field>
      <field type='text-single' label='Username' var='username'>
        <required/>
```

```
      </field>
      <field type='text-private' label='Old␣Password' var='
          old_password'>
        <required/>
      </field>
      <field type='text-private' label='New␣Password' var='password'>
        <required/>
      </field>
      <field type='text-single' label='Mother&apos;s␣Maiden␣Name' var=
          'x-mmn'>
        <required/>
      </field>
    </x>
  </query>
  <error code='401' type='modify'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

The user then SHOULD return the form:

Listing 24: User Returns Password Change Form

```
<iq type='set' from='bill@shakespeare.lit/globe' to='shakespeare.lit'
    id='change2'>
  <query xmlns='jabber:iq:register'>
    <x xmlns='jabber:x:data' type='submit'>
      <field type='hidden' var='FORM_TYPE'>
        <value>jabber:iq:register:changepassword</value>
      </field>
      <field type='text-single' var='username'>
        <value>bill@shakespeare.lit</value>
      </field>
      <field type='text-private' var='old_password'>
        <value>theglobe</value>
      </field>
      <field type='text-private' var='password'>
        <value>groundlings</value>
      </field>
      <field type='text-single' var='x-mmn'>
        <value>Throckmorton</value>
      </field>
    </x>
  </query>
</iq>
```

4 Extensibility

The fields defined for the 'jabber:iq:register' namespace are strictly limited to those specified in the schema. If a host needs to gather additional information, XEP-0004: Data Forms ("x:data") SHOULD be used according to the following rules:

1. A host MUST NOT add new fields to the 'jabber:iq:register' namespace; instead, extensibility SHOULD be pursued via the Data Forms protocol as specified herein.
2. The x:data form shall be contained as a child element of the <query xmlns='jabber:iq:register'/> element (it cannot be a child of the <iq/> stanza and comply with RFC 6120 [9]).
3. The x:data form SHOULD contain x:data fields that correspond to all of the iq:register fields (e.g., username and password).
4. The x:data form SHOULD use a hidden FORM_TYPE field for the purpose of standardizing field names within the form, as defined in Field Standardization for Data Forms (XEP-0068) [10].
5. The x:data form shall take precedence over the iq:register fields; if the submitting entity supports the Data Forms protocol, it SHOULD submit the data form rather than the predefined 'jabber:iq:register' fields, and MUST NOT submit both the data form and the predefined fields (see the Precedence Order section of this document).

Support for extensibility via Data Forms is RECOMMENDED but is not required for compliance with this document.

5 Redirection

A given deployment MAY wish to redirect users to another medium (e.g., a website) for registration, rather than allowing in-band registration. The recommended approach is to include only the <instructions/> element rather than the required fields or a data form in the IQ result, as well as a URL encoded using Out-of-Band Data (XEP-0066) [11] (see the Precedence Order section below for further details).

Listing 25: Host Redirects Entity to Web Registration

```
<iq type='result'
    from='contests.shakespeare.lit'
    to='juliet@capulet.com/balcony'
```

[9] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[10] XEP-0068: Field Data Standardization for Data Forms <https://xmpp.org/extensions/xep-0068.html>.
[11] XEP-0066: Out of Band Data <https://xmpp.org/extensions/xep-0066.html>.

```
    id='reg3'>
  <query xmlns='jabber:iq:register'>
    <instructions>
      To register, visit http://www.shakespeare.lit/contests.php
    </instructions>
    <x xmlns='jabber:x:oob'>
      <url>http://www.shakespeare.lit/contests.php</url>
    </x>
  </query>
</iq>
```

6 Precedence Order

Given the foregoing discussion, it is evident that an entity could receive any combination of iq:register, x:data, and x:oob namespaces from a service in response to a request for information. The precedence order is as follows:

1. x:data
2. iq:register
3. x:oob

(Naturally, if the receiving application does not understand any of the foregoing namespaces, it MUST ignore the data qualified by that namespace.)
Thus it is possible that the host may return any of the following combinations, in which case the submitting application MUST proceed as defined in the table below (it is assumed that "iq:register fields" means instructions plus fields, whereas "iq:register instructions" means the <instructions/> element only):

Included Data	Recommended Processing	Notes
iq:register fields	Submit the completed iq:register fields.	This is the normal processing model for "legacy" services and clients.

Included Data	Recommended Processing	Notes
x:data form + iq:register fields	Submit the completed x:data form if understood, otherwise submit the completed iq:register fields; however, an application MUST NOT submit both.	The iq:register fields would be included for ”legacy” clients; this combination SHOULD NOT be used if the x:data form includes required fields (e.g., a public key) that are not equivalent to defined iq:register fields (the next combination SHOULD be used instead).
x:data form + iq:register instructions	Submit the completed x:data form if understood, otherwise present the iq:register instructions to the user.	This combination SHOULD be used if the x:data form includes required fields (e.g., a public key) that are not equivalent to defined iq:register fields and an alternate URL is not available.
x:data form + x:oob URL	Submit the completed x:data form if understood, otherwise present the provided URL as an alternative.	This combination MAY be returned by a host, but a host SHOULD return the next combination instead in order to support the full range of legacy clients.

Included Data	Recommended Processing	Notes
x:data form + iq:register instructions + x:oob URL	Submit the completed x:data form if understood, otherwise present the iq:register instructions and provided URL as an alternative.	This combination SHOULD be used if the x:data form includes required fields (e.g., a public key) that are not equivalent to defined iq:register fields and an alternate URL is available.
iq:register instructions + x:oob URL	Present the iq:register instructions and provided URL as a redirect.	This combination MAY be returned by a host that wishes to redirect registration to a URL.
iq:register fields + x:oob URL	Submit the completed iq:register fields but optionally present the provided URL as an alternative.	This combination is NOT RECOMMENDED.

7 Key Element

This element is obsolete, but is documented here for historical completeness.
The <key/> element was used as a ”transaction key” in certain IQ interactions in order to verify the identity of the sender. In particular, it was used by servers (but generally not services) during in-band registration, since normally a user does not yet have a ’from’ address before registering. The flow is as follows:

1. Client sends IQ-get request to server.
2. Server returns required elements to client, including <key/> element containing a random string (often a SHA-1 hash).
3. Client sends registration information to server, including <key/> element with the same value provided by the server.

4. Server checks key information and processes registration request; if key is missing or the key value does not match the transaction key provided, the server returns an error.

The <key/> element was also used during registration removal.

8 Stream Feature

RFC 6120 [12] defines methods for advertising feature support during stream negotiation. For the sake of efficiency, it may be desirable for a server to advertise support for in-band registration as a stream feature. The namespace for reporting support within <stream:features/> is "http://jabber.org/features/iq-register". Upon receiving a stream header qualified by the 'jabber:client' namespace, a server returns a stream header to the client and MAY announce support for in-band registration by including the relevant stream feature:

Listing 26: Advertising registration as a stream feature

```
<stream:features>
    <register xmlns='http://jabber.org/features/iq-register'/>
  </stream:features>
```

A server SHOULD NOT advertise in-band registration to another server (i.e., if the initial stream header was qualified by the 'jabber:server' namespace).

9 Error Handling

As defined herein, the 'jabber:iq:register' namespace supports both the old (HTTP-style) error codes and the extensible error classes and conditions specified in XMPP Core. A compliant server or service implementation MUST support both old-style and new-style error handling. A compliant client implementation SHOULD support both. For mappings of HTTP-style errors to XMPP-style conditions, refer to Error Condition Mappings (XEP-0086) [13].

10 Determining Support

If an entity supports the protocol defined herein, it SHOULD report that by including a Service Discovery feature of "jabber:iq:register" in response to disco#info requests.

Listing 27: Service discovery information request

[12] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[13] XEP-0086: Error Condition Mappings <https://xmpp.org/extensions/xep-0086.html>.

```
<iq from='marlowe.lit'
    id='jzg2d175'
    to='shakespeare.lit'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 28: Service discovery information response

```
<iq from='shakespeare.lit'
    id='jzg2d175'
    to='marlowe.lit'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='jabber:iq:register'/>
  </query>
</iq>
```

Although connecting clients typically determine support during stream negotiation via the stream feature, the service discovery feature is helpful for server-to-server connections as well as for clients that are already connected (e.g., to determine if password changes are possible in-band).

11 Security Considerations

In-band registration is usually not included in other messaging protocols (for example, SMTP does not include a method for registering with an email server), often for reasons of security. The registration methods defined herein are known to be insecure and SHOULD NOT be used unless the channel between the registrant and the entity that accepts registration has been secured. For these reasons, the deployment of in-band registration is a policy matter and a given deployment MAY choose to disable in-band registration and password changes. Furthermore, this document should be deprecated as soon as a successor protocol is defined and implemented.

12 IANA Considerations

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

[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/>.

13 XMPP Registrar Considerations

13.1 Protocol Namespaces

The XMPP Registrar [15] includes the 'jabber:iq:register' namespace in its registry of protocol namespaces.

13.2 Stream Features

The XMPP Registrar includes the 'http://jabber.org/features/iq-register' namespace in its registry of stream feature namespaces.

13.3 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>).
As described below, the registered querytypes for registration management are "register" and "unregister".

13.3.1 register

The 'jabber:iq:register' namespace include a mechanism for creating a registration. The registered querytype for doing so is "register".

Listing 29: Register Action: IRI/URI

```
xmpp:marlowe.shakespeare.lit?register
```

Because registration is a two-step process, the application MUST then send an IQ-get to the entity in order to retrieve the required registration fields:

Listing 30: Retrieving Registration Fields

```
<iq to='marlowe.shakespeare.lit' type='get'>
  <query xmlns='jabber:iq:register'/>
</iq>
```

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

Listing 31: Receiving Registration Fields

```
<iq from='marlowe.shakespeare.lit' type='result'>
  <query xmlns='jabber:iq:register'>
    <username/>
    <password/>
  </query>
</iq>
```

The application MUST then present an appropriate interface that enables the user to complete the registration form. Once the user provides the information, the application MUST send an IQ-set to the entity.

Listing 32: Sending Registration Information

```
<iq to='marlowe.shakespeare.lit' type='set'>
  <query xmlns='jabber:iq:register'>
    <username>juliet</username>
    <password>R0m30</password>
  </query>
</iq>
```

The following submission registers the "register" querytype.

```
<querytype>
  <name>register</name>
  <proto>jabber:iq:register</proto>
  <desc>enables registering with a server or service</desc>
  <doc>XEP-0077</doc>
</querytype>
```

13.3.2 unregister

The 'jabber:iq:register' namespace include a mechanism for cancelling an existing registration. The registered querytype for doing so is "unregister".

Listing 33: Unregister Action: IRI/URI

```
xmpp:marlowe.shakespeare.lit?unregister
```

Listing 34: Unregister Action: Resulting Stanza

```
<iq to='marlowe.shakespeare.lit' type='set'>
  <query xmlns='jabber:iq:register'>
    <remove/>
  </query>
</iq>
```

The following submission registers the ”unregister” querytype.

```
<querytype>
  <name>unregister</name>
  <proto>jabber:iq:register</proto>
  <desc>enables cancellation of a registration with a server or
     service</desc>
  <doc>XEP-0077</doc>
</querytype>
```

13.4 Field Standardization

There is one FORM_TYPE and set of associated field types and names for field standardization in relation to each major use case for the ’jabber:iq:register’ namespace: registration, cancellation of registration, and change of password. The initial submissions for these FORM_TYPES are provided below (additional fields may be provided in future submissions).

13.4.1 Registration

```
<form_type>
  <name>jabber:iq:register</name>
  <doc>XEP-0077</doc>
  <desc>Standardization of fields related to registration use case.</
     desc>
  <field
      var='username'
      type='text-single'
      label='Account name associated with the user'/>
  <field
      var='nick'
      type='text-single'
      label='Familiar name of the user'/>
  <field
      var='password'
      type='text-private'
      label='Password or secret for the user'/>
  <field
      var='name'
      type='text-single'
      label='Full name of the user'/>
  <field
      var='first'
      type='text-single'
      label='Given name of the user'/>
  <field
      var='last'
```

```
      type='text-single'
      label='Family name of the user'/>
  <field
      var='email'
      type='text-single'
      label='Email address of the user'/>
  <field
      var='address'
      type='text-single'
      label='Street portion of a physical or mailing address'/>
  <field
      var='city'
      type='text-single'
      label='Locality portion of a physical or mailing address'/>
  <field
      var='state'
      type='text-single'
      label='Region portion of a physical or mailing address'/>
  <field
      var='zip'
      type='text-single'
      label='Postal code portion of a physical or mailing address'/>
  <field
      var='phone'
      type='text-single'
      label='Telephone number of the user'/>
  <field
      var='url'
      type='text-single'
      label='URL to web page describing the user'/>
  <field
      var='date'
      type='text-single'
      label='Some date (e.g., birth date, hire date, sign-up date)'/>
  <field
      var='misc'
      type='text-single'
      label='Free-form text field (obsolete)'/>
  <field
      var='text'
      type='text-single'
      label='Free-form text field (obsolete)'/>
  <field
      var='key'
      type='text-single'
      label='Session key for transaction (obsolete)'/>
</form_type>
```

13.4.2 Cancellation

```
<form_type>
  <name>jabber:iq:register:cancel</name>
  <doc>XEP-0077</doc>
  <desc>Standardization of fields related to cancellation use case.</
     desc>
  <field
      var='password'
      type='text-private'
      label='Password␣or␣secret␣for␣the␣user'/>
  <field
      var='username'
      type='text-single'
      label='Account␣name␣associated␣with␣the␣user'/>
</form_type>
```

13.4.3 Change Password

```
<form_type>
  <name>jabber:iq:register:changepassword</name>
  <doc>XEP-0077</doc>
  <desc>Standardization of fields related to change password use case.
     </desc>
  <field
      var='old_password'
      type='text-private'
      label='Old␣password␣for␣the␣user'/>
  <field
      var='password'
      type='text-private'
      label='Desired␣password␣for␣the␣user'/>
  <field
      var='username'
      type='text-single'
      label='Account␣name␣associated␣with␣the␣user'/>
</form_type>
```

14 XML Schemas

14.1 jabber:iq:register

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
```

```
  targetNamespace='jabber:iq:register'
  xmlns='jabber:iq:register'
  elementFormDefault='qualified'>

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

<xs:annotation>
  <xs:documentation>
    The protocol documented by this schema is defined in
    XEP-0077: http://www.xmpp.org/extensions/xep-0077.html
  </xs:documentation>
</xs:annotation>

<xs:element name='query'>
  <xs:complexType>
    <xs:sequence xmlns:xdata='jabber:x:data'
                 xmlns:xoob='jabber:x:oob'>
      <xs:choice minOccurs='0'>
        <xs:sequence minOccurs='0'>
          <xs:element name='registered' type='empty' minOccurs='0'/>
          <xs:element name='instructions' type='xs:string' minOccurs
              ='0'/>
          <xs:element name='username' type='xs:string' minOccurs='0'
              />
          <xs:element name='nick' type='xs:string' minOccurs='0'/>
          <xs:element name='password' type='xs:string' minOccurs='0'
              />
          <xs:element name='name' type='xs:string' minOccurs='0'/>
          <xs:element name='first' type='xs:string' minOccurs='0'/>
          <xs:element name='last' type='xs:string' minOccurs='0'/>
          <xs:element name='email' type='xs:string' minOccurs='0'/>
          <xs:element name='address' type='xs:string' minOccurs='0'/
              >
          <xs:element name='city' type='xs:string' minOccurs='0'/>
          <xs:element name='state' type='xs:string' minOccurs='0'/>
          <xs:element name='zip' type='xs:string' minOccurs='0'/>
          <xs:element name='phone' type='xs:string' minOccurs='0'/>
          <xs:element name='url' type='xs:string' minOccurs='0'/>
          <xs:element name='date' type='xs:string' minOccurs='0'/>
          <xs:element name='misc' type='xs:string' minOccurs='0'/>
          <xs:element name='text' type='xs:string' minOccurs='0'/>
          <xs:element name='key' type='xs:string' minOccurs='0'/>
        </xs:sequence>
        <xs:element name='remove' type='empty' minOccurs='0'/>
```

```
        </xs:choice>
        <xs:element ref='xdata:x' minOccurs='0'/>
        <xs:element ref='xoob:x' minOccurs='0'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

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

</xs:schema>
```

14.2 Stream Feature

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

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

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0077: http://www.xmpp.org/extensions/xep-0077.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='register' type='empty'/>

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

</xs:schema>
```