XEP-0389: Extensible In-Band Registration

Sam Whited
mailto:sam@samwhited.com
xmpp:sam@samwhited.com
https://blog.samwhited.com/

2020-11-17
Version 0.6.0

Status	Type	Short Name
Experimental	Standards Track	ibr2

This specification defines an XMPP protocol extension for in-band registration with instant messaging servers and other services with which an XMPP entity may initiate a stream. It aims to improve upon the state of the art and replace XEP-0077: In-Band Registration by allowing multi-factor registration mechanisms, and account recovery.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF’s Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy> or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1 Introduction **1**

2 Requirements **1**

3 Glossary **1**

4 Use Cases **2**

5 Discovering Support **2**

6 Flows **2**
- 6.1 Stream Feature . . . 3
- 6.2 Retrieving the Flows . . . 4
- 6.3 Selecting a Flow . . . 5
- 6.4 Issuing Challenges . . . 6
- 6.5 Completing Registration or Recovery . . . 7

7 Challenges **8**
- 7.1 Data Form . . . 8
- 7.2 Out of Band Data . . . 10
- 7.3 SASL . . . 10

8 Internationalization Considerations **11**

9 Security Considerations **11**

10 IANA Considerations **12**

11 XMPP Registrar Considerations **12**
- 11.1 Protocol Namespaces . . . 12
- 11.2 IBR Challenges Registry . . . 13
- 11.3 Challenge Types . . . 13
- 11.4 Namespace Versioning . . . 14

1 Introduction

Historically, registering with an XMPP service has been difficult. Each server either used customized out-of-band registration mechanisms such as web forms which were difficult to discover, or they used In-Band Registration (XEP-0077) [1] which could easily be abused by spammers to register large numbers of accounts and which allowed for only limited extensibility.
To solve these issues this specification provides a new in-band registration protocol that allows servers to present the user with a series of ”challenges”. This allows for both multi-stage proof-of-possession registration flows and spam prevention mechanisms such as proof-of-work functions.

2 Requirements

- The server MUST be able to present multiple challenges to the client.
- The server SHOULD be able reduce account registration spam.
- The server MAY present a challenge that requires the user to complete a step out-of-band.
- A client SHOULD be able to register an account without requiring the user to leave the client.
- A client MUST be able to use the same mechanism to register an account and to recover a forgotten password (subject to server policy).
- A client MUST be able to register with a server as well as external components.

3 Glossary

Challenge A challenge is an action taken during account registration or recovery that requires a response. For example, displaying a form to a user or asking for a token.

Challenge Type The type of a challenge is a unique string that identifies the type of payload that can be expected. For example, a challenge element with type ”jabber:x:data” can be expected to contain a data form. Challenge types must be defined and registered in the challenge types registry. When defining a challenge it is often convenient to reuse an XML namespace from the document defining the challenge.

Flow A flow, or more specifically a ”registration flow” or ”recovery flow”, is a collection of challenges that together can be used to gather enough information to register a new account or recover an existing account.

[1] XEP-0077: In-Band Registration <https://xmpp.org/extensions/xep-0077.html>.

4 Use Cases

- As a server operator, I want to prevent individual spammers from registering many accounts so I require registrants to perform a proof-of-work function before registration is completed.
- As a server operator I want to prevent bots from registering accounts so I require that registrants submit a form which requires user interaction.
- As a user I do not want to lose access to my account if I forget my password, so I provide my email and telephone number in response to the servers data form.
- As a server operator I do not want users to accidentally add an incorrect recovery address so I send an email with a unique link to the indicated account and require that they click the link before registration can continue.
- As a server operator I want to prevent SPIM using a proof-of-possession protocol so I present the user with a form asking for a mobile phone number and then send a verification code to that number via SMS and show another form requesting the verification code.

5 Discovering Support

Clients, servers, and other services such as components that support Extensible IBR MUST advertise the fact by including a feature of "urn::xmpp:register:0" in response to Service Discovery (XEP-0030) [2] information requests and in their Entity Capabilities (XEP-0115) [3] profiles.

Listing 1: Disco info response

```
<query xmlns='http://jabber.org/protocol/disco#info'>…

  <feature var='urn:xmpp:register:0' />…

</query>
```

6 Flows

Registration or recovery is completed after responding to a series of challenges issued by the server. Challenges are grouped in to "flows", a number of challenges that may be issued together to complete an action. For example, a registration flow might be created that issues a data form challenge which will be shown to the user to gather information, then issues a

[2] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[3] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

second data form challenge to let the user enter a confirmation code that was sent to their email.

6.1 Stream Feature

If a server supports registering for or recovering an account using Extensible IBR during stream negotiation, it MUST inform the connecting client when returning stream features during the stream negotiation process. This is done by including a <register/> element, qualified by the 'urn:xmpp:register:0' namespace for account registration, or a <recovery/> element qualified by the same namespace for account recovery. The register and recovery features are always voluntary-to-negotiate. The registration and recovery features MUST NOT be advertised before a security layer has been negotiated, eg. using direct TLS or opportunistic TLS. They SHOULD be advertised at the same time as the SASL authentication feature, meaning that after registration or recovery is completed SASL authentication can proceed.
For recovery or registration, the server MUST include a list of all challenges which the client may receive during the course of registering or recovering an account. These are grouped into "flows" and let the client pick a registration workflow that only contains challenges which the client supports. Each <flow/> element MUST have a unique "id" attribute which is used by the client to identify the flow being selected. The id attribute is only used during this particular flow negotiation and has no meaning after a flow has been selected. Flows must also have at least one <name/> element containing a short, human readable description of the flow. If multiple <name/> elements are present they MUST have unique values for the "xml:lang" attribute. Clients MAY use the name element to show the different flows to the user and ask them to pick between them. Each flow element must also contain an unordered set of <challenge/> elements representing the various challenge types that may be required to complete the registration or recovery flow. Each <challenge/> element contains a "type" attribute that uniquely identifies the challenge for the purpose of determining if it is supported. If a flow would offer the same challenge twice (eg. two data forms asking for different data), the challenge SHOULD only be listed once in the flow element.
For example, a server may advertise a "Verify with SMS" flow and a "Verify by Phone Call" flow that both show a data form asking for a phone number and then a second data form asking for a token provided to the user in a text message or phone call depending on which flow the user selects.

Listing 2: Host Advertises Stream Features

```
<stream:features>
  <mechanisms xmlns='urn:xmpp:sasl:0'>
    <mechanism>EXTERNAL</mechanism>
    <mechanism>SCRAM-SHA-1-PLUS</mechanism>
    <mechanism>SCRAM-SHA-1</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
```

```
  <register xmlns='urn:xmpp:register:0'>
    <flow id='0'>
      <name>Verify with SMS</name>
      <challenge type='jabber:x:data'/>
    </flow>
    <flow id='1'>
      <name>Verify by Phone Call</name>
      <challenge type='jabber:x:data'/>
    </flow>
    <flow id='2'>
      <name>Verify with the web</name>
      <challenge type='jabber:x:data'/>
      <challenge type='jabber:x:oob'/>
    </flow>
  </register>
  <recovery xmlns='urn:xmpp:register:0'>
    <flow id='0'>
      <name>Visit Password Recovery Site</name>
      <challenge type='jabber:x:oob'/>
    </flow>
  </recovery>
</stream:features>
```

Just because a challenge type is listed by the server in the initial flow element does not mean that it will be issued by the server. Servers MAY choose to issue more or fewer challenges based on the result of previous challenges and may not use every challenge type listed in the original flow.

6.2 Retrieving the Flows

Registration or recovery may also be completed after stream neogtiation if server policy allows it.
To find what flows an entity provides (if any) after stream negotiation is complete the requester can send an IQ of type ”get” containing a <register> or <recovery> element qualified by the ”urn:xmpp:register:0” namespace.

Listing 3: Registration flows query

```
<iq type='get'>
  <register xmlns='urn:xmpp:register:0'/>
</iq>
```

When responding to a query for registration or recovery flows the list of challenges MUST be included just as it would be during stream feature negotiation. That is, a ”register” or ”recovery” element containing a list of flows, each with an id, containing a name and a list of challenges.

Listing 4: Registration flows results

```
<iq type='result'>
  <register xmlns='urn:xmpp:register:0'>
    <flow id='0'>
      <name>Verify with SMS</name>
      <challenge type='jabber:x:data'/>
    </flow>
    <flow id='1'>
      <name>Verify by Phone Call</name>
      <challenge type='jabber:x:data'/>
    </flow>
    <flow id='2'>
      <name>Verify with the web</name>
      <challenge type='jabber:x:data'/>
      <challenge type='jabber:x:oob'/>
    </flow>
  </register>
</iq>
```

If an entity supports issuing challenges but does not provide any flows after stream negotiation is complete it MUST respond with an empty list. Similarly, an entity that supports this specification but does not support issuing challenges itself (for example, a client that only supports receiving challenges) it MUST respond successfully with an empty list.

Listing 5: Empty registration flows results

```
<iq type='result'>
  <register xmlns='urn:xmpp:register:0'/>
</iq>
```

6.3 Selecting a Flow

A client selects the registration or recovery feature for negotiation by replying with an element of the same name and namespace. The element MUST contain a <flow> element that MUST have an "id" attribute matching one of the flows advertised by the server. For example, to select the "Verify by Phone Call" registration flow from the previous example, the client would reply with:

Listing 6: Client selects a recovery flow

```
<register xmlns='urn:xmpp:register:0'>
  <flow id='1'/>
</register>
```

If during stream initialization the client attempts to select a flow that does not match one of the flows sent by the server, the server MUST respond with an "undefined-condition" stream error containing an "invalid-flow" application error qualified by the 'urn:xmpp:register:0'

namespace.

Listing 7: Server responds to an invalid selection during stream negotiation

```
<stream:error>
  <undefined-condition xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
  <invalid-flow xmlns='urn:xmpp:register:0'/>
</stream:error>
</stream:stream>
```

If the client is initiating registration or recovery after a stream has already been initiated it uses the same registration element wrapped in an IQ of type "set".

Listing 8: Client selects a recovery flow after stream negotiation

```
<iq type='set' id='foo'>
  <recovery xmlns='urn:xmpp:register:0'>
    <flow id='0'/>
  </recovery>
</iq>
```

If the client attempts to select a flow that does not match one of the flows sent by the server in response to an IQ after stream initialization the server MUST respond with a stanza error of type "item-not-found".

Listing 9: Server responds to an invalid selection after stream negotiation

```
<iq type='error'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

6.4 Issuing Challenges

If a valid flow is selected by the client the server then replies to the IQ or feature selection with a challenge. If replying to an IQ, the challenge must be wrapped in an IQ of type "result". Challenges take the form of a <challenge/> element qualified by the 'urn:xmpp:register:0' namespace with a 'type' attribute that uniquely identifies the type of payload a client might expect the element to contain.

Listing 10: Server issues a challenge

```
<challenge xmlns='urn:xmpp:register:0'
           type='urn:example:challenge'>
  <example xmlns='urn:example:challenge'>Payload</example>
</challenge>
```

After a challenge is received, the client replies to the challenge by sending a <response/> element qualified by the 'urn:xmpp:register:0' namespace or a cancellation as defined later in this document. If the client sends a response, it MUST also include the payload corresponding to the challenges 'type' element (which may be empty).

Listing 11: Client responds to a challenge

```
<response xmlns='urn:xmpp:register:0'>
  <result xmlns='urn:example:challenge'>Example Response</result>
</response>
```

After a response is received, if the server needs more information it MAY issue another challenge. For example, if the user has entered their email in response to a challenge, the server might send an email and then issue another challenge asking for the unique code sent in the email.

6.5 Completing Registration or Recovery

If after receiving a challenge or response a client or server does not wish to continue registration or recovery, it may send an empty <cancel/> element qualified by the 'urn:xmpp:register:0' namespace. This informs the client or server that registration is complete. This is the same as submitting a data form of type 'cancel' in response to a data form challenge.

Listing 12: User Cancels Registration or Recovery

```
<cancel xmlns='urn:xmpp:register:0'/>
```

If the IQ based registration or recovery flow is being used and the server wishes to cancel the flow, it MAY respond to any IQ with the cancel element and type "result".

Listing 13: Client or server cancels request

```
<iq type='result' id='bar'>
  <cancel xmlns='urn:xmpp:register:0'/>
</iq>
```

A server may also issue a cancelation IQ with type 'set' if it wishes to cancel after a request/response has been received (ie. when there is no existing IQ to respond to).

Listing 14: Server cancels flow

```
<iq type='set' id='bar'>
  <cancel xmlns='urn:xmpp:register:0'/>
</iq>
```

If the client successfully completes all required challenges during stream negotiation the server MUST return a <success/> element qualified by the 'urn:xmpp:register:0' namespace, at which point it may continue with the stream negotiation process. The success element MUST contain a <jid> element containing the bare JID as registered or recovered by the server and a <username> element containing the simple user name for use with SASL (normally this will be the same as the localpart of the JID).

Listing 15: Server indicates success during steam negotiation

```
<success xmlns='urn:xmpp:register:0'>
  <jid>mercutio@example.net</jid>
  <username>mercutio</username>
</success>
```

If the IQ based flow is being used and the server wishes to indicate success after a challenge has been completed it sends an IQ of type "set" containing the <success/> element.

Listing 16: Server indicates success after stream negotiation

```
<iq type='set' id='bar'>
  <success xmlns='urn:xmpp:register:0'>
    <jid>mercutio@example.net</jid>
    <username>mercutio</username>
  </success>
</iq>
```

7 Challenges

This document defines several challenges that use existing technologies.

7.1 Data Form

Challenges of type 'jabber:x:data' MUST always contain a data form (an 'x' element with type 'form') as defined by Data Forms (XEP-0004) [4].

Listing 17: Server issues a data form challenge

```
<challenge xmlns='urn:xmpp:register:0'
           type='jabber:x:data'>
  <x xmlns='jabber:x:data' type='form'>
    <title>Chat Registration</title>
    <instructions>
      Please provide the following information
```

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

```
      to sign up to view our chat rooms!
    </instructions>
    <field type='hidden' var='FORM_TYPE'>
      <value>urn:xmpp:register:0</value>
    </field>
    <field type='text-single' label='Given Name' var='first'/>
    <field type='text-single' label='Family Name' var='last'/>
    <field type='text-single' label='Nickname' var='nick'>
      <required/>
    </field>
    <field type='text-single' label='Recovery Email Address' var='
       email'>
      <required/>
    </field>
  </x>
</challenge>
```

The response to a "jabber:x:data" challenge MUST be a form submission (an 'x' element of type 'submit'). For instance, to reply to the data form challenge from the previous example a client might send:

Listing 18: Client submits a data form in response to the challenge

```
<response xmlns='urn:xmpp:register:0'>
  <x xmlns='jabber:x:data' type='submit'>
    <field type='hidden' var='FORM_TYPE'>
      <value>urn:xmpp:register:0</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='Nickname' var='nick'>
      <value>Jule</value>
    </field>
    <field type='text-single' label='Recovery Email Address' var='
       email'>
      <value>juliet@capulet.com</value>
    </field>
  </x>
</response>
```

7.2 Out of Band Data

Challenges of type ”jabber:x:oob” MUST contain an <x/> element qualified by the ”jabber:x:oob” namespace as defined in Out-of-Band Data (XEP-0066) [5].

Listing 19: Server issues an OOB challenge

```
<challenge xmlns='urn:xmpp:register:0'
           type='jabber:x:oob'>
  <x xmlns='jabber:x:oob'>
    <url>http://example.net/login?token=foo</url>
  </x>
</challenge>
```

If the client sends a response to the OOB challenge it MUST be empty.

Listing 20: Client acknowledges the OOB challenge

```
<response xmlns='urn:xmpp:register:0'/>
```

7.3 SASL

Servers can support changing passwords by providing a reset flow containing a SASL challenge. The SASL challenge re-uses the SASL profile from RFC 6120 [6]. The server begins by sending the mechanisms list, and the client responds by selecting a mechanism and possibly including initial data. Each step in the SASL process is issued as a new SASL challenge.

Listing 21: SASL challenge flow

```
<!-{}- Server -{}->
<challenge xmlns='urn:xmpp:register:0'
           type='urn:ietf:params:xml:ns:xmpp-sasl'>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>EXTERNAL</mechanism>
    <mechanism>SCRAM-SHA-1-PLUS</mechanism>
    <mechanism>SCRAM-SHA-1</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
</challenge>

<!-{}- Client -{}->
<response xmlns='urn:xmpp:register:0'>
  <auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl"
        mechanism="SCRAM-SHA-1">
    biwsbj1qdWxpZXQscj1vTXNUQUF3QUFBQU1BQUFBTlAwVEFBQUFBQUJQVTBBQQ==
```

[5] XEP-0066: Out of Band Data <https://xmpp.org/extensions/xep-0066.html>.
[6] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

```
  </auth>
</response>

<!-{}- Server -{}->
<challenge xmlns='urn:xmpp:register:0'
           type='urn:ietf:params:xml:ns:xmpp-sasl'>
  <challenge xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
    cj1vTXNUQUF3QUFBQU1BQUFBTlAwVEFBQUFBQUJQVTBBQWUxMjQ2OTViLTY5Y
    TktNGRlNi05YzMwLWI1MWIzODA4YzU5ZSxzPU5qaGtZVE0wTURndE5HWTBaaT
    AwTmpkbUxUa3hNbVV0TkRsbU5UTm1ORE5rTURNeixpPTQwOTY=
  </challenge>
</challenge>

<!-{}- Client -{}->
<response xmlns='urn:xmpp:register:0'>
  <response xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
    Yz1iaXdzLHI9b01zVEFBd0FBQUFNQUFBQU5QMFRBQUFBQUFCUFUwQUFlMTI0N
    jk1Yi02OWE5LTRkZTYtOWMzMC1iNTFiMzgwOGM1OWUscD1VQTU3dE0vU3ZwQV
    RCa0gyRlhzMFdEWHZKWXc9
  </response>
</response>
```

8 Internationalization Considerations

When providing instructions in a data form, or in the name element of a registration or recovery flow, the server SHOULD use the language specified in the XML stream's current xml:lang, or the closest language for which the server has a translation (eg. based on mutual intelligibility between scripts and languages).
For more information about language tags and matching, see BCP 47 [7].

9 Security Considerations

Servers that allow in-band registration need to take measures to prevent abuse. Common techniques to prevent spam registrations include displaying CAPTCHAs or requiring proof-of-possession of a valid email address or telephone number by sending a unique code (e.g. an HMAC that can later be verified as having originated at the server) to the users email and requiring that they enter the code before continuing. Servers that do not take such measures risk being black listed by other servers in the network.

[7] BCP 47: Tags for Identifying Languages <http://tools.ietf.org/html/bcp47>.

10 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [8].

11 XMPP Registrar Considerations

11.1 Protocol Namespaces

This specification defines the following XML namespace:

- urn:xmpp:register:0

Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [9] shall add the foregoing namespace to the registries located at <https://xmpp.org/registrar/stream-features.html>, and <https://xmpp.org/registrar/disco-features.html> as described in Section 4 of XMPP Registrar Function (XEP-0053) [10].

```
<var>
  <name>urn:xmpp:register:0</name>
  <desc>Support for in band registration and password reset.</desc>
  <doc>&xep0389;</doc>
</var>
```

```
<feature>
  <ns>urn:xmpp:register:0</ns>
  <name>Extensible In-Band Registration</name>
  <element>register</element>
  <desc>Support for in band registration and password reset.</desc>
  <doc>&xep0389;</doc>
</feature>
```

The XMPP Registrar [11] shall also add the foregoing namespace to the Jabber/XMPP Protocol Namespaces Registry located at <https://xmpp.org/registrar/namespaces.html>. Upon

[8] 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/>.

[9] 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] XEP-0053: XMPP Registrar Function <https://xmpp.org/extensions/xep-0053.html>.

[11] 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/>.

advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [12] shall remove the provisional status from this registry entry.

```
<ns>
  <name>urn:xmpp:register:0</name>
  <doc>&xep0389;</doc>
  <status>provisional</status>
</ns>
```

11.2 IBR Challenges Registry

The XMPP Registrar shall maintain a registry of IBR challenges. Challenges defined within the XEP series MUST be registered with the XMPP Registrar.
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>:

```
<challenge>
  <type>A name that uniquely identifies the challenge.</type>
  <desc>A natural-language summary of the challenge.</desc>
  <doc>
    The document (or documents) in which the IBR challenge and its
       payload are
    defined.
  </doc>
</challenge>
```

For an example registration, see the next section.

11.3 Challenge Types

This specification defines the following IBR challenge:

- jabber:x:data
- jabber:x:oob

Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [13] shall add the following definitions to the IBR challenges registry, as

[12] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

[13] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

described in this document:

```
<challenge>
  <type>jabber:x:data</type>
  <desc>Requests that the client fill out an XEP-0004 data form.</desc
     >
  <doc>&xep0389;, &xep0004;</doc>
</challenge>
```

```
<challenge>
  <type>jabber:x:oob</type>
  <desc>Requests that the client execute a URI.</desc>
  <doc>&xep0066;</doc>
</challenge>
```

11.4 Namespace Versioning

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