XEP-0148: Instant Messaging Intelligence Quotient (IM IQ)

Peter Saint-Andre
mailto:xsf@stpeter.im
xmpp:peter@jabber.org
http://stpeter.im/

2005-04-01
Version 1.0

Status	Type	Short Name
Active	Humorous	iq-iq

This specification provides canonical documentation of the jabber:iq:iq namespace.

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

The early Jabber community was a hotbed of innovation and experimentation. Although the community produced a large number of interesting protocols and technologies, not all of them were widely adopted. For example, server-side message filtering (implemented in the mod_filter module of the jabberd [1] server) was one promising technology that simply did not scale up beyond a few hundred concurrent users. Another potentially helpful technology (though even less well-known) was that of the "Instant Messaging Intelligence Quotient" (IM IQ) as defined by the 'jabber:iq:iq' protocol. For the sake of historical completeness, this specification provides canonical documentation of that protocol.

2 Concepts and Approach

It is a harsh reality of the modern Internet that plenty of stupid people have found their way onto today's communication networks (email, Usenet, IM, and the like). Because the early Jabber developers were your typically elitist geeks (whose mantra seems to have been "not everyone can be as smart as we are"), they sought to shield themselves from the inevitable emergence of dumb Jabber users.

At the same time, recognizing the wisdom of the age-old saying "to err is human", they knew that even normally intelligent people sometimes say appallingly stupid things. In fact, such normally intelligent people (well, okay, geniuses) might be the developers themselves! Thus they sought to build protective measures into Jabber servers so that they could avoid appearing dumb.

The early Jabber developers sought to achieve both of these objectives through the design of server-side intelligence detection systems using dedicated protocol elements qualified by the 'jabber:iq:iq' namespace. This protocol implements procedures for discovering, monitoring, and getting feedback on the intelligence of one's own instant messages, as well as that of other users on the network. The "IM IQ" of each user is determined by server-side parsing of messages sent by all registered users of a server, using advanced linguistic analysis techniques (as described under Implementation Notes below) enforced by the mod_iq jabberd module.

3 Use Cases

3.1 Discover Another User's IM IQ

Before chatting with another user over the network or adding that user to one's Jabber roster, it can be helpful to get a sense of how intelligent or unintelligent that person is. This is done by requesting the person's IM IQ from that person's server by sending an IQ get qualified by the 'jabber:iq:iq' namespace to the person's bare JID (user@host) rather than full JID (similar

[1] The jabberd server is the original server implementation of the Jabber/XMPP protocols, first developed by Jeremie Miller, inventor of Jabber. For further information, see <http://jabberd.org/>.

in this regard to the functionality of vcard-temp (XEP-0054) [2]).

Listing 1: Requesting Someone's IM IQ

```
<iq type='get'
    from='kindanormal@example.com/IM'
    to='stupidnewbie@example.com'
    id='imiq1'>
  <query xmlns='jabber:iq:iq'/>
</iq>
```

The server then returns the person's IM IQ, expressed as a REQUIRED <num/> integer between zero and 256, and an OPTIONAL <desc/> containing a natural-language descriptive phrase associated with that range of integer values.

Listing 2: Receiving Someone's IM IQ

```
<iq type='result'
    from='stupidnewbie@example.com'
    to='kindanormal@example.com/IM'
    id='imiq1'>
  <query xmlns='jabber:iq:iq'>
    <num>66</num>
    <desc>moron</desc>
  </query>
</iq>
```

3.2 Discovering One's Own IM IQ

In order for a user to discover his or her own IM IQ, the user sends an IQ get without any 'to' address.

Listing 3: Requesting One's Own IM IQ

```
<iq type='get' id='myiq'>
  <query xmlns='jabber:iq:iq'/>
</iq>
```

Listing 4: Receiving One's Own IM IQ

```
<iq type='result' id='myiq'>
  <query xmlns='jabber:iq:iq'>
    <num>83</num>
    <desc>dull</desc>
  </query>
</iq>
```

[2] XEP-0054: vcard-temp <https://xmpp.org/extensions/xep-0054.html>.

A user may not agree with his or her IM IQ as computed by the server (after all, everyone thinks they are above average). Therefore it is possible that a user may attempt to change his or her IM IQ by sending an IQ set to the server:

Listing 5: Attempting to Set One's Own IM IQ

```
<iq type='set' id='myiq'>
  <query xmlns='jabber:iq:iq'>
    <num>143</num>
    <desc>genius</desc>
  </query>
</iq>
```

However, allowing users to change their own IM IQs is unacceptable, since it would make such information unreliable. Therefore, if a server receives such an IQ set, it MUST return a <not-allowed/> error to the user, and MAY further decrement the user's IM IQ as a result.

Listing 6: Server Returns Error to User on Attempted Set

```
<iq type='error' id='myiq'>
  <query xmlns='jabber:iq:iq'>
    <num>143</num>
    <desc>genius</desc>
  </query>
  <error code='405' type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

3.3 Messaging Hints

Even smart people say stupid things, and we are all familiar with the experience of having said something stupid (or just average) and realizing later that one could have said something exceedingly clever. To prevent people from saying stupid things and to help users appear as smart as possible, the mod_iq jabberd module provides hints to users regarding what to say at a given point in a conversation, based on the advanced linguistic analysis technologies described under Implementation Notes below. A user can ask for a hint by sending the complete message to the server itself, wrapped in a <query/> element qualified by the 'jabber:iq:iq' namespace. (While it may be argued that this functionality could be provided client-side, thus saving a roundtrip, it is consistent with the Jabber philosophy of "smart servers, dumb clients" as explained in XMPP Design Guidelines (XEP-0134) [3].)

Listing 7: Requesting IM IQ Information

[3] XEP-0134: XMPP Design Guidelines <https://xmpp.org/extensions/xep-0134.html>.

```
<iq type='get'
    from='kindanormal@example.com/IM'
    to='example.com'
    id='hint1'>
  <query xmlns='jabber:iq:iq'>
    <message to='stupidnewbie@example.com'>
      <thread>some-thread-id</thread>
      <body>d00d, u r dum -{}- RTFM, OK?</body>
    </message>
  </query>
</iq>
```

The server then determines a more intelligent message to send and returns the XML character data of the <body/> element to the user in a <hint/> element.

Listing 8: Server Hints at a More Intelligent Message

```
<iq type='result'
    from='example.com'
    to='kindanormal@example.com/IM'
    id='hint1'>
  <query xmlns='jabber:iq:iq'>
    <hint>
      I&apos;ve heard that there&apos;s this thing called the Internet
          , which
      contains incredible amounts of helpful information. Have you
          considered
      using it?
    </hint>
  </query>
</iq>
```

Messages checked with the server before sending SHOULD NOT affect the user's IM IQ computation; however, the server MAY decrement the user's IM IQ more significantly if the user ends up sending the original message rather than the smarter one provided by the server.

4 Implementation Notes

4.1 Intelligence Levels

The mod_iq jabberd module uses somewhat out-of-date terminology for intelligence levels [4], as shown in the following table.

[4] See, for example, D. Wechsler, The Measurement of Adult Intelligence (Baltimore: The Williams and Wilkins Company), 1944.

Number Range	Descriptive Label
140+	genius
120-139	very superior
110-119	superior
90-109	normal
80-89	dull
70-79	borderline deficiency
50-69	moron
20-49	imbecile
0-19	idiot

While once common, these terms are now considered politically incorrect. However, please note that this specification merely provides informational documentation of a protocol historically used within the Jabber community, and is not intended to stereotype individuals in any manner whatsoever. A given server implementation of the 'jabber:iq:iq' protocol MAY substitute more modern ranges and terminology if desired or leave out the descriptive phrases entirely, and a given client implementation MAY rename or disguise the descriptive phrases.
That said, it is true that many people on the Jabber network do act like morons, imbeciles, and even idiots.

4.2 Determination of IM IQ

Using the methods described in the next section, the mod_iq jabberd module assigns and dynamically updates a person's IM IQ based on all the messages sent by the user. Upon registration, each user is assigned a baseline IM IQ of 100 ("intelligent until proven an idiot"), unless the user made errors in the registration process or chose an especially stupid password (e.g., "password"), in which case the initially assigned IM IQ could be as low as 70 ("borderline deficiency"). In a manner similar to server-side "karma" ratings, the IM IQ is then modified dynamically based on the semantic value of the user's outbound messages, up to a high of 256 or down to a low of zero (0).
IM IQ is determined based on a user's actual message traffic only, not on other factors such as inane presence status text or the contacts added to the user's roster. While the latter functionality might have been useful, it would violate the rule of not assigning guilt based on association.

4.3 Analysis Methods

The mod_iq jabberd module makes use of several Analytical Language Engine (ALE) technologies for determining the intelligence of specific messages and thus also a user's IM IQ (the

average of all messages sent). These technologies include the following:

Phrasal Objectification of Realtime Threads (PORT) This is a parsing technique for breaking conversation threads into meaningful phrases, even across message boundaries.

Bayesian Estimation of Entropic Responses (BEER) Within information theory, entropy is a measure of the rate of information transfer; this technique uses Bayesian estimation methods to determine whether a given message imparts useful information or not.

Situational Analysis of Kladistic Evolution (SAKE) Kladistics (also spelled "cladistics"), from the Greek "klados" (meaning "branch"), is the study of grouping things into branches that diverge from a common origin; it is used in biology to study descent from a common ancestor, and also in the study of conversation threads to determine how a conversation would evolve depending on things said (or messages sent) at any point in the conversation flow.

Semantic Correlation and Observation of Truth in Conversation Handling (SCOTCH) A person may seem intelligent to the casual observer, but his or her messages may actually not provide deep insights or even track reality in a useful or consistent fashion; this technique builds on early semantic web insights to determine the truth value of a given message within the context of a realtime conversation.

Webs of Intelligent Network Endpoints (WINE) Any given person can engage in conversations with a large number of interlocutors, yet that person's status as an intelligent network endpoint is influenced by reputational factors across the full web of linguistic interactions, not just with any one person; this technique accounts for such reputational effects to paint a complete picture of the person's perceived intelligence across the network.

Naturally, because of the powerful and potentially unpredictable effects of these technologies, development of mod_iq was restricted to senior developers on the jabberd team, or at least (for developers in the U.S.) those over the age of 21. [5]

5 Internationalization Considerations

The descriptive phrases for various intelligence levels SHOULD be localized based on the user's preferred language; however, if the server does not support the 'xml:lang' attribute, this localization MAY be performed by the client.

[5] See Title 23, Chapter 1, Subchapter 1, Section 158 of the United States federal legal code as enacted by the National Minimum Drinking Age Act of 1984 <http://www.law.cornell.edu/uscode/23/158.shtml>.

6 Security Considerations

Most people become somewhat insecure when they realize that in fact they are not as smart as they thought they were; for this reason, querying the server for one's own IM IQ is NOT RECOMMENDED on a regular basis.

7 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [6].

8 XMPP Registrar Considerations

The XMPP Registrar [7] shall include the 'jabber:iq:iq' namespace in its registry of protocol namespaces.

9 XML Schema

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

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

  <xs:import namespace='jabber:client'/>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0148: http://www.xmpp.org/extensions/xep-0148.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='query'>
```

[6] 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/>.

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

```
    <xs:complexType>
      <xs:choice xmlns:jabber='jabber:client'>
        <xs:sequence>
          <xs:element name='num' type='xs:byte'/>
          <xs:element name='desc' type='xs:string' minOccurs='0'/>
        </xs:sequence>
        <xs:element ref='jabber:message'/>
        <xs:element name='hint' type='xs:string'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

</xs:schema>
```