XEP-0026: Internationalization (I18N)

Max Horn
mailto:max@quendi.de
xmpp:black_fingolfin@jabber.org

2003-11-05
Version 0.2

Status	Type	Short Name
Retracted	Standards Track	N/A

NOTE WELL: this document was retracted on 2003-11-05 since the topic is addressed definitively in XMPP Core. Please refer to XMPP Core for further information.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Jabber is meant to allow people everywhere in the world to communicate with each other. However, people converse in many different languages, not just English. Many humans in fact don't even understand English. Hence, Jabber should not be tied to a particular language, but rather allow usage of any language, be it English, Chinese, Inuit, or anything else.
One important step towards this goal is that Jabber is based upon Unicode, allowing for many different languages. But that alone is not enough. Jabber promotes a server-based system for many of its components and services, like the JUD, or transports. Many of these have to interact with users in some way. Currently, they do so in only one fixed language (usually English). Even if the server admin is willing to translate the messages, forms, etc. involved, there can only be one localization active for a given server/component.
Hence, Jabber must support a way for clients to inform the server about their preferred language. In addition, the server and other components have to understand and honor this information. Only this way can we ensure that Jabber is able to work in a multi-national, multi-lingual environment.
Some examples on how this information could and should be used, include

- Forms (e.g. for registration or searching, refer also to Data Forms (XEP-0004) [1]) can be localized, so that instructions and field labels are in the native language of the person who has to fill them out
- Even if the form can't be sent in the proper language (e.g. simply because it hasn't yet been translated), the component still should tag its reply with the language being used
- Incoming messages in a different language could be automatically translated (server-side or client-side)
- Redirection of messages based on their language (think of a help desk which services world wide requests)
- Transports to services which are not unicode based could use the language information as a hint at the best encoding (least lossage) for converted messages

2 Implementation

The basic idea behind this proposal was to use existing standards where possible, and to make it fully backward compatible. Furthermore it was a goal to allow clients to support it now, even without any server support, while at the same time permitting improved functionality once servers start to implement this spec.

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

2.1 Encoding the locale

To encode the locale on any given XML packet, we use the xml:lang attribute, as defined in the XML specification. This in turn uses values as specified in RFC 1766 to encode languages and regions. This way, you can even distinguish between British and Australian English.

Listing 1: Example message with locale set to German

```
<message to='friedrich@jabber.org' xml:lang='de-DE'>
  <body>Ich bin ein Berliner!</body>
</message>
```

An xml:lang tag can be put onto any XML element; for the purposes of this document, however, we will limit its usage to the four central Jabber elements: <stream/>, <message/>, <iq/> and <presence/>.

2.2 Client support

A client claiming to support this document has to initiate server connection slightly differently by putting an xml:lang attribute in the initial <stream:stream> element.

Listing 2: Jabber session initiated with Canadian French as default

```
<?xml version="1.0" encoding="UTF-8" ?>
<stream:stream to='jabber.org' xmlns='jabber:client'
          xmlns:stream='http://etherx.jabber.org/streams' xml:lang='fr
              -CA'>
```

Servers not supporting this document will just ignore the additional attribute. Compliant server can be distinguished by the fact that their reply <stream:stream> element also contains an xml:lang attribute, indicating the main language of the server. A compliant client has to detect whether the server is compliant or not, and base its future behavior on this information.

Listing 3: Reply by an English-language Jabber server

```
<stream:stream from='jabber.org' id='12345' xmlns='jabber:client'
          xmlns:stream='http://etherx.jabber.org/streams' xml:lang='en
              '>
```

If the client thus determines that the server is compliant, then it doesn't have to do anything beyond this point. All its outgoing messages will automatically be flagged by the server with an xml:lang attribute if necessary. Thus writing a minimal compliant client is trivial.
If it is determined that the server does not support this document, and the client still wants to offer locale support, it may start flagging all its outgoing message/iq/presence elements with the xml:lang attribute, to ensure that other components/clients which do conform to this document can handle the localization despite the local server not doing so.

Finally, if for whatever reasons the client wants to flag particular messages with a different locale (e.g. if the user is bilingual), it can do so at any time by putting an appropriate xml:lang element in the outgoing data. This will override the previously set default locale for this message only.

2.3 Server support

A compliant server must detect the xml:lang attribute in incoming <stream:stream> elements. The server then has to store this information for later use, i.e. it has to remember the default language for each active session.
Additionally, a compliant server must attach an xml:lang attribute to the reply <stream:stream> element sent in response to a newly initiated connection. This attribute should reflect the default language of that server, and is used to indicate to clients that the server implements this document.
The server should not only allow user clients to specify a default language this way, but also server-side components, like the JUD should be allowed to do this.
Whenever a message leave the server, it has to tag the message automatically with the xml:lang attribute of the corresponding seesion, if any was specified, unless the message is already tagged this way. In that case, the already existing xml:lang attribute takes precedence, thus allowing for greater flexibility.
If a client send a message to another local client which uses the same xml:lang value, then no change is applied. But if the recipient uses a different xml:lang, and if the message has no xml:lang attribute attached yet, the xml:lang of the server has to be attached before delievey of the message.

2.4 Service support

Jabber based services that wish to comply to this document have to make sure that all information they send to clients is tagged with an xml:lang attribute corresponding to the language used in the outgoing data, if appropriate, even if the component supports no other localizations. An example for this is a search form based on XEP-0004.

Listing 4: Search form in English

```
<iq from='users.jabber.org' type='result' id='4' xml:lang='en'>
  <query xmlns='jabber:iq:search'>
    <instructions>
        Fill in a field to search for any matching Jabber users.
    </instructions>
    <nick/>
    <first/>
    <last/>
    <email/>
```

```
    <x xmlns='jabber:x:data'>
      <instructions>
          To search for a user fill out at least one
          of the fields below and submit the form.
      </instructions>
      <field type='text-single' label='First (Given)' var='first'/>
      <field type='text-single' label='Last (Family)' var='last'/>
      <field type='text-single' label='Nick (Alias)' var='nick'/>
      <field type='text-single' label='Email' var='email'/>
    </x>
  </query>
</iq>
```

This way, a client could for example offer to translate the form since it now knows the language the form was written in. Previously it could just guess the language was English, which never was guaranteed.
To be able to tailor replies to the user's preferred language, the component has to know this information. This is simply inferred from any xml:lang attribute on incoming requests. If none is present, the default locale is assumed. If the client's default locale diverges from that of the component, it is the server's responsibility to tag the query with an appropriate xml:lang attribute (refer to the "Server support" section). If on the other hand the server is not compliant, then any interested client will manually tag its queries with an xml:lang attribute. Thus it is sufficient to check for this attribute.

Listing 5: Request for a German-language search form

```
<iq to='users.jabber.org' type='get' id='5' xml:lang='de'>
  <query xmlns='jabber:iq:search'/>
</iq>
```

A more sophisticated component supporting multiple localizations of its forms/messages could now honor the requested language and send this search form instead of the English one shown previously:

Listing 6: Search form in German

```
<iq from='users.jabber.org' type='result' id='5' xml:lang='de'>
  <query xmlns='jabber:iq:search'>
    <instructions>
        Füllen Sie ein Feld aus um nach einem beliebigen
        passenden Jabber-Benutzer zu suchen.
    </instructions>
    <nick/>
    <first/>
    <last/>
    <email/>
```

```
    <x xmlns='jabber:x:data'>
      <instructions>
          Um nach einem Benutzer zu suchen, füllen Sie mindestens
             eines
          der folgenden Felder aus und schicken dann das Formular ab.
      </instructions>
      <field type='text-single' label='Vorname' var='first'/>
      <field type='text-single' label='Nachname' var='last'/>
      <field type='text-single' label='Spitzname' var='nick'/>
      <field type='text-single' label='Email' var='email'/>
    </x>
  </query>
</iq>
```

If the component doesn't have the requested localization available, it replies with the default localization (but of course with the matching xml:lang attribute tagged to it, and not the one of the request).