XEP-0038: Icon Styles

Adam Theo
mailto:theo@theoretic.com
xmpp:theo@theoretic.com

2003-06-02
Version 0.5

Status	Type	Short Name
Deferred	Standards Track	None

A protocol for specifying exchangeable styles of emoticons and genicons within Jabber IM clients.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction 1

2 Requirements 1

3 Discovery 2

4 Specification 2
- 4.1 Definition File . . . 3
- 4.2 Meta Elements . . . 3
- 4.3 Text Strings . . . 3
  - 4.3.1 Internationalization . . . 4
  - 4.3.2 Whitespace . . . 4
  - 4.3.3 Case Sensitivity . . . 4
- 4.4 Objects . . . 4
- 4.5 X Tags . . . 4
- 4.6 Hierarchy . . . 5
- 4.7 Packaging . . . 5

5 Implementation Notes 5
- 5.1 Procedure . . . 6
- 5.2 Text Strings . . . 6
- 5.3 Objects . . . 7
- 5.4 Combining Styles . . . 7
- 5.5 Core Icons . . . 8

6 Security Considerations 9

7 IANA Considerations 9

8 XMPP Registrar Considerations 9

9 Formal Definition 9
- 9.1 Schema . . . 9
- 9.2 icondef.xml file example . . . 11

10 Future Considerations 12

1 Introduction

This proposal standardizes the use of graphical emoticons and ”genicons” in Jabber IM clients to prevent confusion and competing conventions that will soon arise, enabling client developers to spend their energy on more important parts of their projects.
Emoticons are the text string ’smiley’ or ’frowny’ faces such as :-) and :-(that people often use in email or instant messaging to represent emotions. Genicons are a term being coined here to mean non-emotive text pictures (often called ’ASCII Art’) that serve to replace typing the full word or to simply be cute or creative in the conversation.
Many new Internet users demand graphical emoticon and genicon support in their IM clients. We should satisfy their needs if we ever wish to see them use Jabber instead of another IM system.
While traditionally emoticons and genicons have been typed and displayed as text, the recent trend of using graphics, and sometimes sounds, instead of text to represent these pictures will be assumed for purposes of this proposal. Also, the term ”icon” will be used in place of ”emoticon” and ”genicon” for purposes of convenience.

2 Requirements

The following issues must be solved for in this specification.

- **Here and Now**: The convention should not rely on technologies such as feature negotiation or generic pub/sub which do not exist yet in Jabber in order to work properly. The convention should be a ”here and now” solution, relying only on the current Jabber protocol.
- **Meaningful**: The convention for creating the icons must be meaningful to users of clients without icon support, or even users on the other side of transports. Whatever method is used, it should not be cryptic or difficult to understand in text-only mode, in case the user does not have or want the capability to view the icons.
- **Component Friendly**: In order for gateways, bots, and other ”semi-intelligent” components to be able to convert or otherwise use the specification (such as to and from the MSN icon style, or use icons in conversations with Artificially Intelligent bots), a specific list of icons must be possible. Since these bots do not have an intelligent human behind them deciding their every action, they cannot deal with unlimited possibilities.
- **Protect Privacy**: Sender-defined external data must not be required to properly display the icons because it can be used by the sender to track the user, circumventing their privacy. When the sender can define their own graphics or sounds to be used for the icons, they can use unique URLs and monitor when these unique URLs are accessed. External data may be allowed as long as it is not required for proper use of the icons, but it would be best to leave external data completely out of the system.

- **Conserve Bandwidth**: The amount of markup used within the messages to define the icons should be kept to a minimum, as well as keeping the media files used for the icons from being transferred over the wire with the messages themselves. These easily consume too much bandwidth, a precious resource to most of the Internet.
- **Be Passive**: The convention should not force transports or recipients to take any action to properly display the icons. Not only does it put extra strain on the components, but it would also unnecessarily break old clients. Transports and recieving clients may have the option of taking some actions, but they should not be forced to.
- **Internationalization**: The convention must be inherently international in order to ease adoption of Jabber throughout the world and prevent later growing pains. The use of English must be kept to a minimum, restricted to ”meta information” about the icons themselves, and must dynamically allow for non-English language sets.

Because icons in Jabber should be easy to use, extensible, and customizable, they will be created using style definition files which can be exchanged between users and supporting clients. The specification will not allow external data, in order to protect the privacy of users, and will not rely on file transfers or directory services in order to not break old clients or components.

3 Discovery

To find out if an entity supports Jabber Icon Styles, look for the feature category of http://jabber.org/protocol/icon-styles using jabber:iq:browse or http://jabber.org/protocol/disco. The same feature category can be used with feature negotiation.

4 Specification

Because icons in Jabber should be easy to use, extensible, and customizable, they will be created using style definition files which can be exchanged between users and supporting clients. The specification will not require external data, in order to protect the privacy of users, and will not rely on file transfers or directory services in order to not break old clients or components. How these icon styles are exchanged - as well as advertised - is out of the scope of this specification. The text strings representing the icons will be sent like any other text (this document doesn’t require extra tags or attributes in the messages being sent).
All icons are created by defining each icon then grouping them together into ”Icon Definition Files”. These files, along with the object files associated with the icons, are called ”icon styles”. Icon styles may be traded and shared among users of all supporting clients like skins or themes, similar to WinAmp, XMMS, GNOME, and other customizable applications. This creates a platform-independent system, providing a great degree of customization for the

user, and allowing client developers to focus on other features.

4.1 Definition File

Each icon in a style is defined and grouped together in an XML document, the "Icon Definition File". Each definition file for all styles is named "icondef.xml". There is only one such Icon Definition File per style. The W3C Schema for the Icon Definition File plus an example finished Icon Definition File can be found under Schema, below.

4.2 Meta Elements

The meta elements contain information about the Icon Style itself, rather that the individual icons. They are contained within the <meta> element, which is directly under the root element. There is one and only one the <meta> element.

- **Name**: This is an arbitrary text string (spaces allowed) which is the full name of the Icon Style. Only one name can be included.
- **Version**: This is an arbitrary text string (no spaces allowed) which is the version number of the Icon Style. Any version format is allowed, but the major.minor or major.minor.trivial formats are recommended. Only one version can be included.
- **Description**: This is the full description of the Icon Style. It summarizes the look and types of the icons, and can be used for keywords in searching. This is optional, but recommended.
- **Author**: This is the full name of the author. Multiple authors are allowed.
- **Creation**: This is the date of the creation of this version of the Icon Style. The date is in the ISO 8601 standard format.
- **Home**: This is the full HTTP web address of the Icon Style's homepage. This is optional.

4.3 Text Strings

The <text/> element defines what text string(s) are recognized by the client as an icon. There may be multiple <text/> elements in an <icon/>, such as for different languages or simply for multiple text strings producing the same result (for example: :-) and :))

4.3.1 Internationalization

Each may have an xml:lang attribute, as defined in Section 2.12 in the official XML 1.0 reference document. The xml:lang attribute allows for two-letter language codes, dialects, and custom ”languages” to define foreign IM protocols, for example.

4.3.2 Whitespace

In order to be more accurate in recognizing text strings intended to be icons from those that are just coincidences in normal conversation, the client should follow the ”whitespace rule” of making sure there is some form of whitespace on at least one side of the text string. This is only a guideline; individual clients can implement different rules as needed. A newline and tabs count as whitespace in addition to spaces. This is to make sure that chunks of code and URIs are not accidentally converted into graphical pictures. Also, text strings cannot include newlines or tabs. All characters must be on the same line with only spaces within the string, and extra spaces should not be ignored. This is to make it much easier on text parsers looking for these text strings.

4.3.3 Case Sensitivity

The text strings must be case sensitive. This is a rule that compliant clients must follow. ”Cat” cannot be used in place of ”cat” or ”CAT”. All three are separate text strings, and therefore must have separate <text/> elements, although they may of course use the same objects.

4.4 Objects

The <object/> element defines what multimedia objects are inserted into the displayed message, replacing the text string(s) defined in <text/>. An object may be a bitmap graphic, vector graphic, audio clip, rich text formatting rules, or any other media that can be stored in a separate file. The <object/> element is identical to the OBJECT element used in XHTML 2.0, and the specification used there should be used to govern the <object/> element here. Note that because the XHTML 2.0 OBJECT specification is quite complex (although very flexible and future-proof), client developers are encouraged to only implement compliant sub-sets of the OBJECT specification for their clients. There may be one or multiple <object/> elements in an <icon/>, such as for alternative file formats (such as GIF vs. PNG), or multiple objects to use at the same time (such as graphic and sound files).

4.5 X Tags

The <x/> element allows any type of extensions to the icondef.xml file, such as to specify how the user’s nickname can be colored in multi-user chat windows or defining additional

data about the style or authors. Each must have an xmlns attribute with a namespace that the extension operates under. Multiple <x/> elements may be in the <icon/> or <icondef/> elements. This functionality is optional for clients to support, and clients should ignore all extensions they do not understand.

4.6 Hierarchy

The icondef.xml file must all be located in the root directory, which is named after the style and version (example: ./happy_days-1.2 or ./gold_angelic-1.0.0). There is only one root directory per style. The object files may either be in the root directory as well, or be in sub-directories for categorization and easier maintenance reasons. If sub-directories are used, they must match the URIs used in the <object/> element in the icondef.xml file.

Listing 1: Gold Angelic v1.0.0 Hierarchy Example

```
./gold_angelic-1.0.0
./gold_angelic-1.0.0/icondef.xml
./gold_angelic-1.0.0/alert.ogg
./gold_angelic-1.0.0/red-exclamation-mark.svg
./gold_angelic-1.0.0/shocked.svg
./gold_angelic-1.0.0/woman.svg
./gold_angelic-1.0.0/pngs/happy.png
./gold_angelic-1.0.0/pngs/sad.png
./gold_angelic-1.0.0/pngs/man.png
./gold_angelic-1.0.0/pngs/woman.png
./gold_angelic-1.0.0/wavs/boohoo.wav
./gold_angelic-1.0.0/wavs/choir.wav
./gold_angelic-1.0.0/wavs/alert.wav
```

4.7 Packaging

The icondef.xml file and all object files must be packaged in the ZIP format following the above hierarchy (the directory must exist in the package, with all files in it). The package must have the file extension .jisp (Jabber Icon Style Package), and the MIME type application/vnd.jisp as defined in the official IANA registration. This allows Jabber clients to automatically install icon styles through web browsers. When the client installs the package, it should probably be kept in the archived format, instead of unzipped. This not only saves disk space, but also makes the packages easier to manage and exchange.

5 Implementation Notes

Icons styles should be easy to create, distribute, and most importantly, use. The packaging and official MIME type helps with the first steps, but it is ultimately up to the client developers

to fully support the specification and make sure it is easy for users to manage.

5.1 Procedure

The procedure for using Jabber icons is simple and straightforward.

1. The user installs and turns on (”activates”) an icon style (using their client’s GUI, most likely) they have received off the Internet or somewhere else.
2. The user selects the desired icon they wish to include in their message from their client’s GUI (a drop-down list of installed icons, for example).
3. The client translates the selected icon into its main text string equivelent, which is found in the icon’s <text> element. If there are multiple <text> elements, follow the procedure under ”Text Strings”, below.
4. The client sends the message with the text string to the intended recipient.
5. The recipient client receives the message with the text string.
6. If the recipient has an icon style activated that defines that text string, it is converted into the appropriate object. If there are multiple objects files for each icon, follow the procedure under ”Object Files”, below. If there are multiple icon styles installed which define that text string, follow the procedure under ”Combining Styles”, below. If the recipient does not have such an icon activated, then they simply see the text string without any changes.

5.2 Text Strings

Because icons may have multiple text strings associated with them, clients need to be able to figure out which one to use when a user selects the desired icon from their GUI. This is ultimately completely up to the implementation, but here is a suggestion:

1. First look for any text strings that are the same language as what the user is using (if the user is using english, then look for English text strings, ignoring languages like German or Simplified Chinese). Optionally, if the client is smart enough, it can instead look for text strings in the language that the intended recipient is using (using Browse/Disco or simply keeping track of languages used in an ongoing conversation). The language for text strings is found in the xml:lang attribute for <text/> elements.
2. If there are no text strings that match the user’s language, then look for one that has no defined language.
3. If there are multiple options even after the above steps, select the one that comes first in the <icon/> element.

4. If there are no text strings that are in the user's language or no language defined at all, then it is probably best for the client to not even display the icon as a choice at all. But if the client still wants to allow it, then simply select the text string of any language that comes first in the <icon/> element.

5.3 Objects

Like multiple text strings, icons can have multiple object data files associated with them, and therefore clients also need to be able to figure out which ones to use when a user selects the desired icon from their GUI. Here is a suggestion of how those files can be chosen among multiple options, although this is completely an implementation issue (as with multiple text strings).

1. First look for any of these objects that are of MIME types the client can understand and use. For example, most clients could only use image/bmp and maybe audio/x-wav, but some newer clients may be able to support image/png, image/svg+xml and audio/x-ogg.
2. If there are still multiple object MIME types which the client can understand and use, then the client should rank the MIME types it can support, choosing its favorite.
3. If there are still multiple objects of the same MIME type, then the client should choose the first one in order in the <icon/> element.
4. If there are no files of any MIME type the client can understand, then it should ignore using any objects or even the entire icon.

The Rules for processing objects in the XHTML 2.0 OBJECT specification may also be of help in coding the procedure of choosing an object to use, especially when it comes to nested and author-preferred objects.
The client should also take note of the file sizes. The client should set (possibly as a user-defined option) the maximum file size in kilobytes for object files. Anything above this amount implies the file will be too big to properly render, and the icon style developer is probably being abusive.
Also, if you are developing an icon style, please make sure the MIME types specified in your icondef.xml file are correct. And also make sure that the files you use are reasonable in any byte, pixel, and timelength size. And although any file format can be supported, try to use BMP, PNG, SVG, WAV, OGG formats because they are open, free, and becoming increasingly supported in developer tools and programming languages.

5.4 Combining Styles

A client may permit the user to activate multiple icon styles at one time. This would be useful for styles which make use of different text strings, and the user wants them all. The client should force the user to rank the multiple styles for purposes of conflict resolution between

icons. The highest ranking style gets preference over lower ranking styles. This ranking doesn't have to be anything more than simply dragging the style names into top-to-bottom rows, with the styles on top being higher ranked than those below.

5.5 Core Icons

Although any text string can be turned into an icon by defining it in an icondef.xml file, it is highly reccomended they either follow traditional ASCII Art (smileys and frownys, for example) or full keywords in simple markup such as double-colons. If you want to design icons, always keep in mind that not every Jabber user uses graphics to "translate" this to something visual, as explained in the "Meaningful" requirement, above. Here is a short list of recommended "core" icons that should be in most definitions, as well as possibly be used by transports:

- **:-)** and **:)** - A smiling face.
- **:-(** and **:(** - A frowing face.
- **;-)** and **;)** - A winking smiling face.
- **:'-(** and **:'(** - A crying face.
- **>:-(** and **>:(** - An angry face.
- **:yes:** - A thumbs-up or checkmark.
- **:no:** - A thumbs-down or a crossmark.
- **:wait:** - An open hand, palm outstretched.
- **@->--** and **:rose:** - A rose flower.
- **:telephone:** - A telephone.
- **:email:** - An electronic-looking envelope.
- **:jabber:** - Jabber's lightbulb logo.
- **:cake:** - A birthday cake.
- **:kiss:** - A pair of puckered lips.
- **:heart:** and **:love:** - A heart.
- **:brokenheart:** - A broken heart.
- **:music:** - A musical note or instrument.
- **:beer:** - A beer mug.

- **:coffee:** - A cup of coffee.
- **:money:** - A gold coin.
- **:moon:** - A moon.
- **:sun:** - A sun.
- **:star:** - A star.

6 Security Considerations

There are no security features or concerns related to this proposal.

7 IANA Considerations

The XMPP Standards Foundation (XSF) [1] shall register and maintain a MIME type and file extension for icon style packages with the IANA. Ones have already been registered by Sebastiaan Deckers (aka 'CBAS') as application/vnd.jisp and .jisp, respectively. The registration can be found at http://www.iana.org/assignments/media-types/application/vnd.jisp. Sebastiaan's registration shall be considered the official MIME type and file extension of this specification.
Also, this specification uses other MIME types that are maintained by IANA for the object and xml files that are included in the icon style packages.

8 XMPP Registrar Considerations

JANA shall register the namespace http://jabber.org/protocol/icon-styles as an official feature category.
Also, JANA may choose to define IM-specific xml:lang "language codes" for use within Jabber (in addition to those defined in the XML specification). Such language codes would allow Jabber developers to support icons from MSN, Yahoo, and popular web message programs.

9 Formal Definition

9.1 Schema

[1] The XMPP Standards Foundation (XSF) is an independent, non-profit membership organization that develops open extensions to the IETF's Extensible Messaging and Presence Protocol (XMPP). For further information, see <https://xmpp.org/about/xmpp-standards-foundation>.

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="icondef">
    <xs:complexType>
      <xs:sequence>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="version" type="xs:string"/>
            <xs:element name="description" type="xs:string"/>
            <xs:element name="author" type="xs:string" maxOccurs="
                unbounded">
              <xs:complexType>
                <xs:attribute name="jid" type="xs:string" use="
                    optional"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="creation" type="xs:date"/>
          </xs:sequence>
        </xs:complexType>
        <xs:element name="icon" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="text" type="xs:string" minOccurs="1"
                  maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="xml:lang" type="xs:language" use
                      ="optional"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="object" type="xs:string" minOccurs="1"
                   maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="mime" type="xs:string" use="
                      required"/>
                </xs:complexType>
              </xs:element>
              <xs:element name="x" type="xs:string" minOccurs="0"
                  maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="xmlns" type="xs:anyURI" use="
                      required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
```

```
        <xs:element name="x" type="xs:string" minOccurs="0" maxOccurs=
            "unbounded">
          <xs:complexType>
            <xs:attribute name="xmlns" type="xs:anyURI" use="required"
                />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

9.2 icondef.xml file example

```
<?xml version='1.0' encoding='UTF-8'?>
<icondef>
  <meta>
    <name>Gold Angelic</name>
    <version>1.0.0</version>
    <description>Angelic faces and themes with gold highlights.</
        description>
    <author jid="theo@theoretic.com">Adam Theo</author>
    <author jid="cbas@rhymbox.com">Sebastiaan Deckers</author>
    <author jid="cobnet@jabber.org">Mattias Campe</author>
    <creation>2002-10-31</creation>
    <home>http://icon-styles.jabberstudio.org/Gold_Angelic</home>
  </meta>
  <icon>
    <text>:-)</text>
    <text>:)</text>
    <object mime="image/png">happy.png</object>
    <object mime="audio/x-wav">choir.wav</object>
  </icon>
  <icon>
    <text>:-(</text>
    <text>:(</text>
    <object mime="image/png">sad.png</object>
    <object mime="audio/x-wav">boohoo.wav</object>
  </icon>
  <icon>
    <text>:-O</text>
    <text>:O</text>
    <object mime="image/svg+xml">shocked.svg</object>
  </icon>
  <icon>
    <text xml:lang="en">::man::</text>
    <text xml:lang="de">::mann::</text>
    <text xml:lang="cs">::muz::</text>
```

```
    <text xml:lang="x-msn">(z)</text>
    <object mime="image/png">man.png</object>
  </icon>
  <icon>
    <text xml:lang="en">::woman::</text>
    <text xml:lang="x-msn">(x)</text>
    <object mime="image/svg+xml">woman.svg</object>
    <object mime="image/png">woman.png</object>
  </icon>
  <icon>
    <text xml:lang="en">::alert::</text>
    <object mime="image/svg+xml">red-exclamation-mark.svg</object>
    <object mime="audio/x-ogg">alert.ogg</object>
    <object mime="audio/x-wav">alert.wav</object>
  </icon>
</icondef>
```

10 Future Considerations

1. This proposes a standard, simple, plaintext-based icon convention. It does not attempt to create a powerful XML-based convention wherein XML tags are used to extensibly represent the icons. However, this proposal isn't exclusive, and does not prevent such an XML-based convention from being created at a later date. It may also be possible to build such an extensible XML-based convention on top of this plaintext one.

2. The Icon Styles that the sender used to create the message may be of use to the recipient. A separate specification can be developed for the transferring of the name and version of the Icon Styles used by the sender. The recipient's client may then be smart enough to automatically use the same icons as the sender intended, possibly even fetching and downloading the icon styles as needed. This would likely require directory services, so has not been included in this proposal.

3. The functionality of the icondef.xml files may be useful for variable text strings. Text strings that may change from session to session, such as the user's nickname in a multi-user chat. A set of variables could be defined that represent these variable datas, therefore allowing them to be manipulated in the same way set text strings are.

4. This convention may be adopted beyond the scope of Jabber into web browsers, email clients, and Wiki sites. There is great potential for a generalized, exchangeable convention for emoticons and genicons.

5. A "SVG Transport" may be created to enable clients to send SVG icons to it and get back icons in another format of their choice, such as PNG or JPG. This would speed up the adoption of SVG within Jabber greatly.

6. An online Web- (or even Jabber-) based editor to create icon style packages from the individual components would be a great tool for developers. This would be a great project for JabberStudio or even the Ransom model.