Jabber Technical White Paper

History

Title	Date	Author	Publisher
Jabber Technical White Paper	April 25, 2000	Catherine Dodson	Jabber.com, Inc.

Table of Contents

Executive Summary

Jabber instant messaging software enables users to communicate in text-based conversations in real time. Instant messaging (IM) is expected to exceed the demand of email, and Jabber is positioned to offer the fastest, smartest, and most convenient tool for instant communications.

Jabber's unique position in the market is a result of its XML-based architecture. Benefits of the XML architecture include:

- It integrates easily into other programs and systems.
- It provides more structure and intelligence than binary protocols.
- It functions across platforms and operating systems.

Jabber is a server-based program that runs on a system of distributed servers. Each user's roster and preferences are stored on the server. A user can log on from any client and access user preferences, subscription lists, and messages.

In the current version of Jabber, users can communicate with AIM, ICQ, and Jabber servers. They can search for users on ICQ and Jabber servers. In the future, support for other instant messaging systems will be added. Jabber's server-based architecture enables major upgrades to take place without necessitating a redistribution of the client software.

Jabber comes from the open source programming movement (http://www.opensource.com). In this movement, programmers from all over the world participate in the development, coding, and testing of software. In Jabber's case, over 400 programmers are registered developers for the software, and dozens participate on a daily basis. This level of participation enhances the overall quality of the software. The time between new bug fixes, security patches, and releases is greatly accelerated. The vast array of systems in use by the developers ensures that clients are developed for and tested in a wide spectrum of user environments. Jabber.com looks forward to collaboration between the open source developers and its own programmers.

The result of Jabber's flexibility is that you can communicate friends and colleagues in multiple IM's and organize all contacts in one piece of software. *

* COMPATIBILITY DISCLAIMER Unlike the proprietary Instant Messaging services in existence today, Jabber represents a platform for instant communications applications; it can be used to create consumer IM services, but has far broader uses. Jabber.org is an open-source development project designed and founded on the principles of open-source, open-standards and XML architecture. Jabber.com is a commercial entity created to provide value added products and services to prospective users of this open-source platform. Our mission is to provide the extended products and services that empower and enable the development of services that can propel the IM industry far beyond today's world of instant chat.

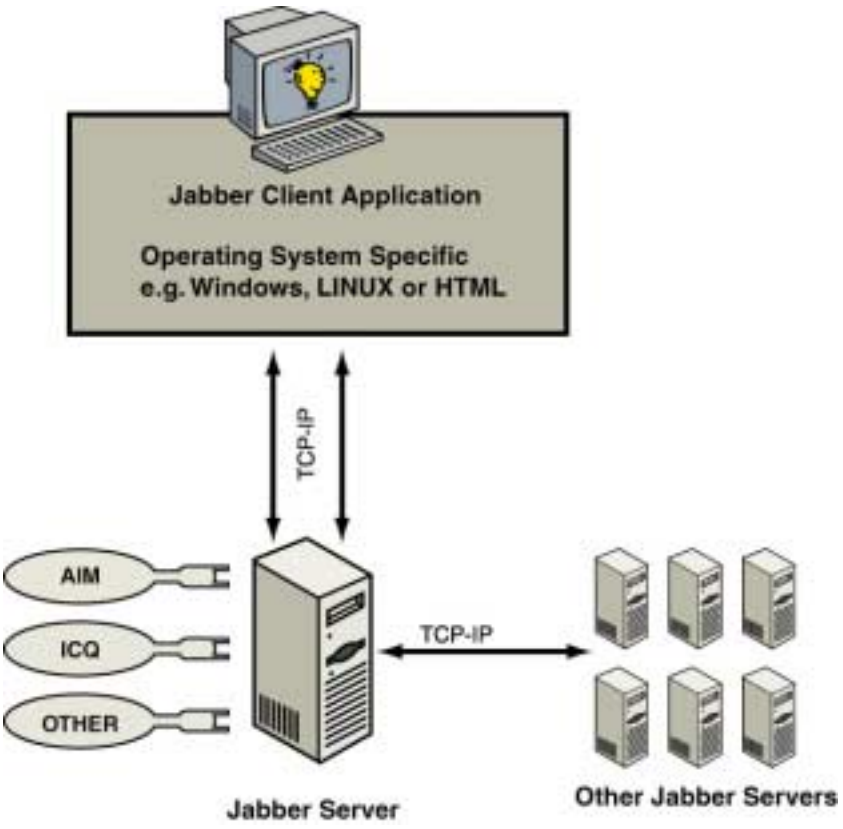
We believe that interoperability standards for IM are highly important, and ultimately inevitable. Jabber.org is actively involved with and closely engineering the Jabber system to follow the IM standardization directions of the Internet Engineering Task Force (IETF). Jabber.org is also actively engineering interfaces to every IM service of significance. However, during this transition period where proprietary IM services engage in blocking and tackling, we cannot guarantee interoperability with any proprietary IM network.

Jabber Architecture

Jabber architecture is almost identical to that of email. Each user has a local server which receives information for them. The various local servers transfer the messages among themselves for delivery to users. Any number of servers can be supported, with each server representing a unique and separate "community" or domain. Jabber Identifiers are typically expressed identically to email: user@server.com.

Jabber clients and servers connect through XML streams. Communications occur between clients and servers and between multiple servers. In client/server communications, the XML stream is always initiated by the client to the server. The architecture can support simple clients (e.g., a direct telnet connection). XML is an integral part of the architecture making it fundamentally extensible and able to express almost any structured data.

Server-to-server communication consists of simply routing the elements of the protocol over an XML stream from one server to another. There is no special server-to-server protocol or features.



Jabber specifically requires *no* direct client-to-client connections. All Jabber messages and data from one client to another must go through the server. Any client is free to negotiate direct connections to another client, but those connections are for application specific usage only. There are specific instances where this is encouraged, such as file transfers. Jabber will deliver the file transfer information in the form of a URL, but how a client either serves or retrieves the file is completely at the will of the user/client.

Protocol

The Jabber protocol consists of XML fragments passed over XML streams between clients and servers. There are three primary protocols that define the basic types of XML fragments used in Jabber:

- Messages
- Presence
- Info/Query

Each of these protocols is defined in detail later in this document.

Jabber Clients

Jabber clients are currently being developed for multiple platforms, including Windows, Macintosh, Linux, Java, and the World Wide Web. All clients will be able to:

- Communicate to the server through TCP sockets
- Parse well-formed XML
- Understand the Message data type

In addition to the required features, clients can optionally perform the following actions:

- Express presence (online/offline/unavailable) information to the server and understand incoming presence data.
- Understand the Info/Query data type and have some preset queries such as logging in, rosters, searching, and setting user information.
- Display preferences for messages and presence data, such as icons or interface styles (chat, group chat, etc.).
- Contain MIME fields in messages.
- Transfer files out-of-band to other clients either via the server or directly.

Whether or not the client supports these features depends on the client developer. A complete list of the features in the JabberIM 1.0 client is available on page 45.

Jabber Servers

The Jabber server is the component that accepts incoming XML streams from clients, manages their online presence, and delivers messages to and for those clients.

A Module API enables the server to use external modules to handle user functionality, such as filtering messages, storage facilities (offline messages, rosters, user info), and user authentication.

A Service API extends the functionality of the server to enable the integration of security, special connections for alternate clients, and message logging.

Transport servers are used to bridge the Jabber protocol to other services (IRC, ICQ, AIM, etc.). Etherx, an XML middleware component, enables the server to communicate quickly between other transports on the same machine and transparently to other servers over the Internet. Etherx is an implementation-specific, middleware utility, and is not required by the overall Jabber protocol or for server-to-server communication.

The Message Protocol

Jabber uses the message protocol to send instant messages as an XML stream. Messages may be sent between two clients, a client and a server, or between two servers. Only entities which have a Jabber Identifier can send and receive Jabber instant messages.

The basic message protocol adheres to the following format:

```
<message> </message>
```

It may be modified by a number of attributes, as described in the next section.

Attributes of <message/>

The message protocol may be modified by the following attributes:

Attribute	Function	Example

to="*" from="*"	Identifies the sender and recipient. Their addressing is based on the Jabber Identifiers specification. This attribute is required in all instant messages.	`<message to="jsmith@example.com">`
		`<body>Do you have the new report?</body>`
		`</message>`

id="*"	Applies a unique identifier to the message. The client can use the id to identify the message in case the message generates error messages. It is optional and is not used elsewhere in the system.	`<message to="jsmith@example.com" id="1001">`
		`<body>Do you have the new report?</body>`
		`</message>`

[default}	Indicates that the message is a normal message. By default, the client assigns this type if no other type attribute is given.	`<message to="jsmith@example.com">`
		`<body>Do you have the new report?</body>`
		`</message>`

type="error"	Indicates that the message is a special error message. The actual error is described in an <error></error> element within the message.	`<message to="jsmith@example.com" type="error">`
		`<error type="404">Not found</error>`
		`</message>`

type="chat"	Indicates that the message should be displayed in a line-by-line chat interface (1-to-1 chat).	`<message to="jsmith@example.com" type="chat">`
		`<body>Do you have the new report?</body>`
		`</message>`

Attribute	Function	Example

type="groupchat"	Indicates that the message should be displayed in a chat room interface.	`<message to="jsmith@example.com" type="groupchat">`
		`<body>Do you have the new report?</body>`
		`</message>`

Elements within <message/>

The following tags are used to define elements within a Jabber message. The first set of tags define what the content of the message is. The second set embeds metadata within the message.

Content

<body></body>

This element surrounds the main text of the message. The <body/> element only may exist once within every message and may contain only plain text.

<x xmlns="jabber:x:*"></x>

This element is used to send commands between clients or as an extension mechanism. Each time the element is used, the namespace (xmlns) must be defined. A single message may have multiple instances of the <x/> element.

For example, the Out Of Band (oob) namespace extension can be used to transfer files between applications:
`<x xmlns="jabber:x:oob"></x>.`

<error type="nnn"></error>

This element is included when the message type attribute is set to "error". The actual error is defined by a type="nnn" attribute that contains a number indicating what the error is.

- 302 - Redirect
- 400 - Bad Request
- 401 – Unauthorized
- 402 - Payment Required
- 407 - Registration Required
- 408 - Request Timeout
- 409 – Conflict
- 500 - Internal Server Error

- 403 – Forbidden
- 404 - Not Found
- 405 - Not Allowed
- 406 - Not Acceptable
- 501 - Not Implemented
- 502 - Remote Server Error
- 503 - Service Unavailable
- 504 – Remote Server Timeout

The content of the error element is a textual description of the specific error. For example, a bad request would have the following format:

```
<error type="400">Bad Request</error>
```

Metadata

<subject></subject>

This element contains the subject of the message.

<thread></thread>

The recipient client always returns the identical contents for this element when it replies directly to a message. This enables the sender and recipient to identify replies and create a conversation thread. The thread is usually a unique, random, ID string.

Examples of the Message Protocol

The following Jabber instant messages provide examples of the Message Protocol attributes and elements in use.

In the first example, the <message/> protocol is used in a simple message. Notice that only the recipient attribute is used.

```
<message to="jsmith@example.com">
  <body>Hi</body>
</message>
```

In the second example, the instant message contains subject and thread metadata. The thread element is maintained between messages received and their replies so that a conversation thread is created.

```
<message from="michelle@example.com/work">
  <subject>project 241</subject>
```

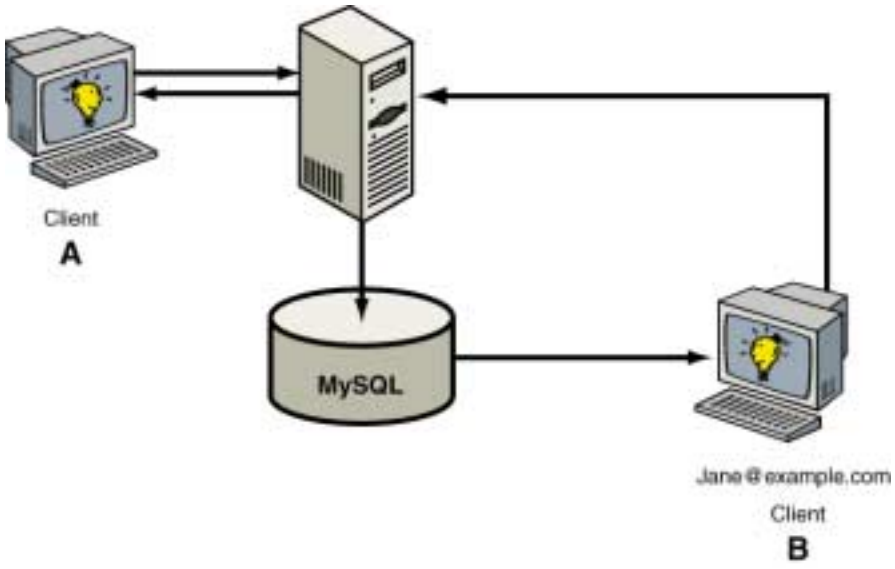
```
  <body>Did you make any progress on it
today?</body>
  <thread>A54E33</thread>
</message>
```

The Presence Protocol

The Presence protocol provides availability information about a Jabber entity. Any entity identified by a Jabber Identifier can communicate a message to any other entity, including:

- Client to client
- Client to server
- Server to server, or
- Any arrangement of any two entities.

All entities expressing presence are either available or unavailable. The Available state implies that the sender can immediately receive information. The Unavailable state indicates that the sender cannot receive any data at the current time.



The following attribute is used to make a client unavailable:

```
type="unavailable"
```

By default, all presence expresses availability unless it contains the `type="unavailable"` attribute.

Subscriptions

A Jabber client can subscribe to the presence of any entity (anything with a Jabber ID). A subscription is an agreement to forward presence changes to the subscriber. For example, you can subscribe to a friend's online presence so that whenever that friend comes online, you are notified. Likewise, whenever the friend disconnects, updated presence information is sent out.

In addition, a user can subscribe to an IM service. In this case, whenever the user logs into Jabber, presence information is sent to the IM service, effectively logging the user into the service as well.

The following table shows how subscription statements are used within Jabber.

Attribute	Function	Example
type="subscribe"	Jane requests that George send her his presence information when it changes.	`<presence to="George@aim.jabber.com" from="Jane@jabber.org" type="subscribe"/>`
type="subscribed"	George accepts Jane's request. His client will send Jane his presence information when it changes.	`<presence to="Jane@jabber.org" from="George@aim.jabber.com" type="subscribed"/>`
type="unsubscribe"	Jane requests that George stop sending her his presence information when it changes.	`<presence to="George@ aim.jabber.com" from="Jane@jabber.org" type="unsubscribe"/>`
type="unsubscribed"	George's client removes Jane's subscription. It will no longer send Jane any presence information.	`<presence to="Jane@jabber.org" from="George@aim.jabber.com" type="unsubscribed"/>`

Subscriptions fall into the following categories:

- To – sends your presence info to another entity.
- From – receives presence info from another entity.
- Both – both sends and receives presence info.

Probes

A server uses the Probe presence packet to request the presence information for a specific entity. In this way, it determines if a specific entity is available or unavailable. The probed entity must grant permission for the presence information to be sent.

The Probe Presence request is sent using the following element:

```
<presence type="probe"/>
```

Elements within <presence/>

The following elements are used within the <presence/> protocol.

State

<status></status>

The Status element displays a textual status description that is suitable for users to view directly. For example, the client might display a text description saying "I'm at lunch" or "Be back in 5 minutes".

<priority></priority>

The Priority element prioritizes the numerous presences associated with a single entity. For example, a single entity (jsmith@example.com) may be logged into multiple resources (e.g., home computer and work computer). This element assigns a numerical priority to each resource. The resource with the highest number is the default or primary resource. All messages and communications go to the resource with the highest priority number.

When the highest priority resource becomes unavailable, messages and communications are sent to the resource with the next highest priority. Negative priority indicates that the resource should not be used for direct or immediate contact.

Display

<show></show>

The Show element indicates how to display an available user's online status to other clients. The following four options are available:

Tag	Meaning
<show>chat</show>	The client is available for immediate contact.
<show>away</show>	The client is online, but momentarily away (e.g., at lunch or a meeting).
<show>xa</show>	The client is online, but has been inactive for a long time.
<show>dnd</show>	The client is in Do Not Disturb mode.

The <show> tag is typically accompanied by a <status> tag which contains a more descriptive reason. The <show> tag is optional. If it is not present, clients will indicate that the user is in a "normal" state.

Extensions

<x xmlns="jabber:x:*"></x>

This element is used to send commands between applications or as an extension mechanism. Each time the element is used, the namespace (xmlns) must be defined. A single message may have multiple instances of the <x/> element.

Examples of the Presence Protocol

The following example shows a generic use of the presence protocol:

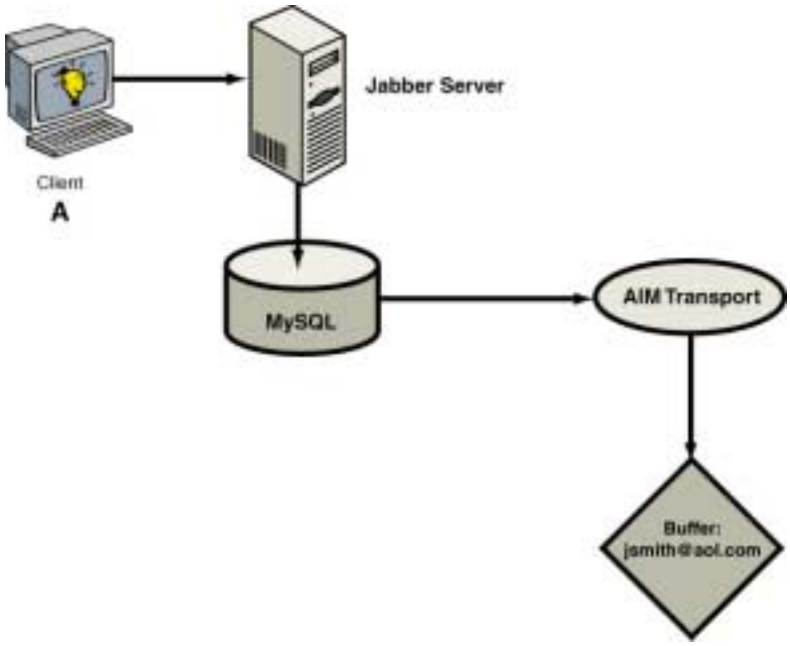
```
<presence type="unavailable">
 <show>xa</show>
</presence>
```

The next example shows the presence protocol when a client receives status information from another resource:

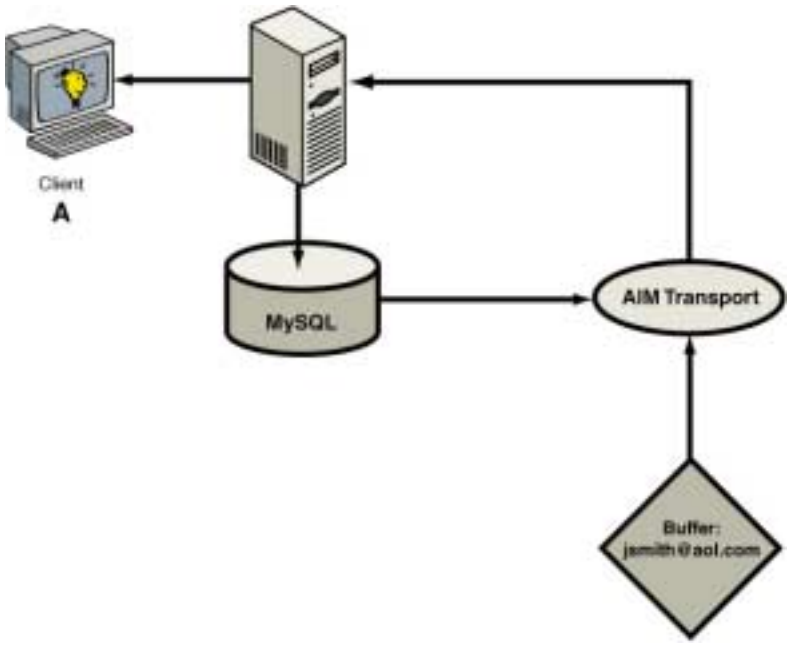
```
<presence from="joe@server.com/work">
  <status>At work...</status>
</presence>
```

Presence can also involve more complex interactions, as demonstrated in the following illustrations. In the first illustration, the client pushes its presence to the Jabber server. The server responds by sending the

client's presence to everyone on its subscription list, and retrieving presence from them. If the client is not yet logged onto a transport, the transaction cannot be completed. A temporary buffer stores the request for presence information from individuals on the transport's service.



Once the client sends its presence to the transport, thereby logging on, the temporary buffer passes the request for presence information back to the transport. The transport can check for the contacts on the subscription list, and send their presence information back to the client by way of the Jabber server.



Info/Query Protocol

Info/Query (IQ) is a simple client/server framework within Jabber. It structures a rudimentary conversation between any two entities in Jabber and allows them to pass XML-formatted queries and responses back and forth. The primary use is for fetching or setting common user information such as name, email, and address. However, its flexible design permits any kind of structured conversation to occur. Any entity identified by a Jabber Identifier can participate in an IQ conversation with any other entity.

Attributes of <iq/>

The following attributes are used to modify the IQ conversation.

Attribute	Function	Example
to="*" from="*"	Identifies the sender and recipient. Their addressing is based on the Jabber Identifiers specification. This attribute is required in all instant messages.	`<iq to="jsmith@example.com"/>`
id="*"	Applies a unique identifier to the message. The client can use the id to identify the message in case the message generates error messages. It is optional and is not used elsewhere in the system.	`<iq to="jsmith@example.com" id="1001"/>`
type="get"	Retrieves information associated with a query namespace. When communicating with other services (e.g., AIM or IRC), a blank Get query may be sent to retrieve information. A list of fields for which the client needs to provide information will be returned to the Jabber server. This attribute is included in the query by default if no type attribute is set.	`<iq type="get" to="jsmith@example.com"/>`

Attribute	Function	Example
type="set"	Indicates that the message is a query containing data intended to set values or replace existing values.	`<iq type="set" to="jsmith@example.com"/>`
type="result"	Indicates that the message is a successful response to a Get or Set query.	`<iq type="result" from="jsmith@example.com"/>`

type="error"	Indicates that the query failed. The actual error is described in an Error element within the IQ.	`<iq type="error" to="jsmith@example.com">`
		`<error type="404">Not found</error>`
		`</iq>`

The Query Element

Within each IQ, a namespace further defines the type of query to perform. The namespace is defined in the Query element, as shown below. Only one Query element may exist in an IQ.

```
<query xmlns="*"/>
```

For example, a client sends a Set query with the Client Authentication namespace to the server to log on:

```
<iq type="set" to="jsmith@example.com">
<query xmlns="jabber:iq:auth">
</query>
</iq>
```

The Query element requires the namespace be defined in the `xmlns="*"` attribute. The jabber:iq:* namespace is reserved for the core Jabber protocols. However, developers can expand IQ functions by adding to the jabber:x:* namespace. The use of "query" in the namespace is not required. For example, the vCard query does not contain the term "query".

```
<iq type="get" id="1001" to="jsmith@jabber.org">
<query xmlns="vcard-temp">
</query>
</iq>
```

The following table lists the standard namespaces available for Jabber.

Namespace	Explanation	Example

jabber:iq:auth	The Simple Client Authentication namespace is a simple mechanism for the clients to authenticate and create a resource representing their connection to the server.
	Successful authentication results in an IQ type="result" response. Errors are returned in the IQ error element.
	If no username or password is sent, the server will create an anonymous resource if it is supported.
		`<iq type="set" id="uniquevalue">`
		`  <query xmlns="jabber:iq:auth">`
		`  <username>jsmith</username>`
		`  <password>secret</password>`
		`  <resource>HomePC</resource>`
		`  </query>`
		`</iq>`

jabber:iq:roster	The Contact List Management namespace is used by clients to manage their roster on the server. The roster is the authoritative list of subscription information for this account, shared between the client and server.
	The roster consists of a list of items. Each item element may have attributes describing it. Each item element may contain group elements for each group of which it is a part. The attribute descriptions are:
	• **jid="jabberuser@server"** is the Jabber ID of the item.
	• **subscription="none"** is the current status of the subscription related to this item. It can be none (no subscription), to (we have a subscription to this item), from (they have a subscription to us), or both (to and from).
	• **ask="subscribe"**: The current status of a request to this item. It can be Subscribe or Unsubscribe indicating that a request is being sent to this item for a subscription or a subscription cancellation.
	• **name="JackS"**: A nickname.
	Clients can control only the JID and name attributes, the group elements, and create/remove items. All other attributes are managed by the server depending on how a client responds to presence subscription requests.
		`<iq type="set" id="uniquevalue">`
		`  <query xmlns="jabber:iq:auth">`
		`<item jid="jsmith@example.com"`
		`name="John Smith"`
		`subscription="to"`
		`ask="subscribe">`
		`  <group>friends</group>`
		`  <group>school</group>`
		`</item>`

Namespace	Explanation	Example
jabber:iq:agents	The Available Agents namespace obtains a list of entities that have special properties and can perform functions for another entity. Most commonly, this is used to show the list of transports available on a server.	<iq id="wjAgents" type="result" from="Jabber.org"><query xmlns="jabber:iq:agents"> <agent jid="users.jabber.org"> <name>User Directory</name> <description>You may register and create a public searchable profile, and search for other registered Jabber users.</description> <service>jud</service> <register/> <search/> </agent> <agent jid="aim.jabber.org"> <name>AIM Transport</name> <description>This is the AIM Transport</description> <transport>AIM/AOL ScreenName</transport> <service>aim</service> <register/> </agent> </query> </iq>
jabber:iq:agent	The Agent Properties namespace obtains the properties of one agent. This is usually done after a jabber:iq:agents query (see above), to register with a specific service, agent or transport. It might also be used to examine the detailed properties of a specific agent. For example, the client can determine if open registration is allowed.	<iq id="wjAgent" type="result" from="Jabber.org"><query xmlns="jabber:iq:agent"> <agent jid="aim.jabber.org"> <name>AIM Transport</name> <description>This is the AIM Transport</description> <transport>AIM/AOL ScreenName</transport> <service>aim</service> <register/> </agent> </query> </iq>
jabber:iq:register	The Registration Requests namespace registers with a server or service. The Registration namespace is also used to update or remove a registration.	<query xmlns="jabber:iq:register"> <instructions>Some instructions to be displayed when the user is filling out the form.</instructions>

Namespace	Explanation	Example
		<username/> <password/> <email/> <date/> <key/> </query>
jabber:iq:oob	The Out Of Band Data namespace gives clients a standard way to do client-to-client file transfers. A distinct namespace will be implemented for server passthrough/proxy transfer.	<iq type="set" to="jsmith@jabber.org" id="file_1"> <query xmlns="jabber:iq:oob"> <url>http://192.168.1.1:5890/building1a.gif</url> <desc>Here's the blueprint.</desc> </query> </iq>
jabber:iq:time	The Client Time namespace gives clients a standard way to exchange local time.	<iq type="result" from="jsmith@jabber.org"> <query xmlns="jabber:iq:time"> <utc>20000424T14:55:06</utc> <display>4/24/00 7:55:06 PM</display> </query> </iq>
jabber:iq:version	The Client Version namespace gives clients a standard way to find out version information for another user's client.	<iq type="result" from="jsmith@jabber.org/JabberIM"> <query xmlns="jabber:iq:version"> <name>JabberIM</name> <version>Version 1.0</version> <os>95 4.10</os> </query> </iq>
jabber:iq:search	The Search namespace is used to initiate a search. Any agent can be a search agent. For example, you can have JUD, which searches for Jabber users, or you can have the ICQ transport search for ICQ users. See below for a complete description of this namespace.	<iq type="get" id="1001"to="users.jabber.org" from="jsmith@jabber.org/winjab"> <query xmlns="jabber:iq:search"/> </iq>

The Search namespace

Any agent can be a search agent. For example, JUD searches for Jabber users, and the ICQ transport searches for ICQ users.

To initiate a search, the client sends a Get query to obtain the searchable fields:

```
<iq type="get" id="1001" to="users.jabber.org"
from="jsmith@jabber.org/winjab">
<query xmlns="jabber:iq:search"/>
</iq>
```

The search agent returns the fields that can be searched:

```
<iq type="result" id="1001" from="users.jabber.org">
<query xmlns="jabber:iq:search">
<instructions>Fill in a field to search for any
matching Jabber User</instructions>
<name/>
<first/>
<last/>
<nick/>
<email/>
<key>067941fd96a6a2752a21abcb6d737130dd51dd50</key>
</query>
</iq>
```

Notice that the fields are returned in a form with instructions. A key is included to provided security to the transaction (see below). The user can now enter search criteria into the form based on the available fields. The client sends a Set query back to the agent to have it actually perform the search:

```
<iq type="set" id="1002" to="users.jabber.org"
from="jsmith@jabber.org/winjab">
<query xmlns="jabber:iq:search">
<last>Smith</last>
<key>11b830e604215c3a2a24652c69fd4efa2a7a5746</key>
</query>
</iq>
```

The server returns the results from the query:

```
<iq type="result" id="1002" from="users.jabber.org">
<query xmlns="jabber:iq:search">
<item jid="jsmith@jabber.org">
<name>Jane Smith</name>
<first>Jane</first>
```

```
<last>Smith</last>
<nick>SamplePerson</nick>
<email></email>
</item>
<item jid="janes@jabber.org">
<name>Jane Smith</name>
<first>Jane</first>
<last>Smith</last>
<nick>Janey</nick>
<email>janesmith@example-corp.com</email>
</item>
</query>
</iq>
```

Notice that there are two sets of items tags containing identical information. This is due to the fact that there are two ways for agents to send results:

- A single result tag
- Multiple results "pushed" to the client, similar to roster pushes, i.e., one record at a time.

The manner in which the results are sent is a property of the search agent. For example:

```
<iq type="set" from="icq.jabber.org" id="1003">
<query xmlns="jabber:iq:search">
<item jid="11117280@icq.jabber.org">
<email>jsmith@example.net</email>
<nick>Janey</nick>
<given>Jane</given>
<family>Smith</family>
</item>
</query>
</iq>
```

Multiple results can be pushed by the server. When all data has been sent, the server sends the following result:

```
<iq type="result" from="icq.jabber.org" id="1003">
<query xmlns="jabber:iq:search"/>
</iq>
```

The client receives multiple "sets", one per record, and then a final "result" indicating the "end of data". In each <item> tag, the JID attribute is mandatory.

The Error Element

This element is included when the message type attribute is set to "error":

```
<error type="nnn"></error>
```

The actual error is defined by a type="nnn" attribute that contains a number indicating what the error is. The content of the error element is a textual description of the specific error. (See page 13 for a complete list of error codes and messages.)

The Key Element

The <key/> element provides a layer of security to client-server interactions. It is used with the jabber:iq:register and jabber:iq:search namespaces.

When a client initiates an interaction with the server, the server sends the client a <key> tag containing a unique value. When the client returns information for the <iq type="set">, it echoes back the unique value in the <key> field. In this way, the server can verify that the client is the same entity that received the original key.

Fetching Information (type="get")

Information is retrieved from a client or server by sending an IQ with a type attribute of "get". This query attribute retrieves information associated with a query namespace.

Empty Elements

To retrieve a list of empty elements, use the Get attribute with no namespace defined. For example, to retrieve a client's contact list (or roster), the following IQ with a type="get" could be sent:

```
<iq type="get">
<query xmlns="jabber:iq:roster"/>
</iq>
```

Complete Response (type="result")

When the results for a "get" are returned, the empty element tags contain a value. Any errors resulting from the "get" are specific to the namespace of the query and should be expressed in an appropriate syntax.

Setting Information (type="set")

The Set attribute indicates that the message is a query containing data intended to set values or replace existing values.

Elements/Data

The Set query is used to record information or make changes in information. The data contained within the elements of a Set query should be stored as a future result to a Get query in that namespace.

Response (type="result")

The result of a successful Set query is an empty response. The result should be matched by the sender using the id="" attribute on the original Set query.

Examples of Information/Query Interactions

Simple Query

The following is a simple query for user information.

Query

```
<iq type="get" to="user@server.com">
  <query
xmlns="jabber:iq:roster"><name/><email/></query>
</iq>
```

Response

```
<iq type="result" from="user@server.com">
  <query xmlns="jabber:iq:roster"><name>John Smith
</name><email>jsmith@example.com</email></query>
</iq>
```

Setting Values

The following is a query for vCard values.

Sent by the client:

```
<iq type="get" id="1001" to="jsmith@jabber.org">
<query xmlns="vcard-temp">
</query>
</iq>
```

Received from the server:

```
<iq type="result" id="1001" from="jsmith@jabber.org">
<vCard version="3.0" prodid="-//HandGen//NONSGML vGen
v1.0//EN" xmlns="vcard-temp">
<FN/>
<N>
<given>John</given>
<family>Smith</family>
</N>
<nickname>JBird</nickname>
<url>www.jbirdontheweb.com</url>
<bday/>
<org><orgname/>
<orgunit/>
</org>
<title/>
<role/>
<tel><voice/>
<home>201-555-1212</home>
</tel>
<tel><fax/>
<home/>
</tel>
<tel><msg/>
<home/>
</tel>
<adr><home/>
<extadd/>
<street>12315 St. Laurent Ave.</street>
<locality>Summerville</locality>
<region>CO</region>
<pcode>80020</pcode>
<country>USA</country>
</adr>
<tel><voice/>
<work/>
</tel>
<tel><fax/>
<work/>
</tel>
<tel><msg/>
<work/>
</tel>
<adr><work/>
<extadd/>
<street/>
<locality/>
<region/>
<pcode/>
<country/>
```

```
</adr>
<email><internet/>
<pref/>
Jsmith@aol.com</email>
</vCard>
</iq>
```

Jabber Identifiers

Within Jabber there are many different entities that need to communicate with each other. These entities can represent clients of alternate messaging systems, group chat rooms, or a single Jabber user. Jabber Identifiers are used both externally and internally to express ownership or routing information.

Characteristics of Jabber Identifiers include:

- They uniquely identify individual objects or entities for communicating instant messages and presence information.
- They are easy for users to remember and express in the real world.
- They are flexible enough to enable the inclusion of other IM and presence schemes.

Jabber IDs (JIDs)

Each Jabber ID (or JID) contains a set of ordered elements. The IDs are formed of a domain, node, and resource in the following format:

```
user@host/resource
```

or

```
[node@]domain[/resource]
```

John Smith might have the following JID on his home system: `Jsmith@example.com/home.` He can have multiple IDs representing multiple resources, for example Home, Work, and Notebook: `Jsmith@example.com/mobile.` Resources are temporary descriptions that exist only while the user is logged on.

The ID elements are defined as follows:

- The Domain Name is the primary identifier. It represents the Jabber server to which the entity connects.
- The Node is the secondary identifier. It represents the "user". All Nodes live within a specific Domain.

- The Resource is the optional third identifier. All Resources belong to a Node. Within Jabber the Resource is used to identify specific objects that belong to a user, such as online sessions or paths of communication.

Identifier Syntax

URI-based

The Jabber Identifier has the same general form as an email address. Each user has their local server which receives information for them. The various local servers transfer the messages between each other. Any number of servers can be supported with each server representing a unique and separate "community" or domain. Jabber Identifiers are typically expressed as: user@server.host.

Syntax

The syntax of the URI-based identifier is as follows:

```
[node@]domain[/resource]
```

Friendly Identifier

For usage outside of Jabber and human-to-human identity exchanges, the address breaks down into a simple email-like address.

```
node@domain
```

Implementation

The Jabber Identifier conforms to the following standards.

FQDN

Every usable Jabber Domain should resolve to a Fully Qualified Domain Name (FQDN).

DNS and MX Records

Since Jabber Identifiers use the same structure as email, they need to have a mechanism of redirecting to another server (besides the FQDN) for the actual service. Email uses the Mail Exchange (MX) record in DNS, and because Jabber is closely affiliated with email and uses a similar addressing scheme, it will also use the MX record. To prevent Jabber from interfering with normal SMTP routing and to enable an administrator to have separate servers for SMTP and Jabber, the Jabber

server does a reverse-sort (highest priority first) on the MX records available.

Client/Server Interactions

The following sections provide a detailed description of communication between clients and servers in Jabber.

Connecting

To connect to a Jabber server, a client opens a standard XML stream to the server on port 5222. The default namespace for Jabber clients is "jabber:client".

For example, a client could send the following XML stream to connect to jabber.org:

```
<stream:stream to="jabber.org" xmlns="jabber:client"
xmlns:stream="http://etherx.jabber.org/streams/">
```

The server responds over the open socket. For example, the server could send the following response to the client:

```
<stream:stream from="jabber.org"
xmlns="jabber:client" id="0123ABCD"
xmlns:stream="http://etherx.jabber.org/streams/">
```

This XML stream is kept open during the lifetime of the session, and all communications between the client and server are sent and received over this socket.

Authentication / Logging In

The client uses an IQ conversation to authenticate the user to the server. The IQ conversation also creates a session resource representing their connection to the server. Successful authentication results in a normal IQ type="result" response. Errors are returned in the normal IQ error element.

If the client does not send a username/password, the server assumes that it is requesting the creation of an anonymous resource

(server.com/resource). If it supports anonymous resources, it will respond appropriately.

Jabber uses a one-way secure hash function (or, SHA1) to authenticate the user. This function creates a value by hashing the session ID and the user password. The server verifies that the correct hash value was returned by the client.

While the use of SHA1 authentication is optional, clients should always attempt a digest authentication before going with plain-text.

Sample Authentication Query

```
<iq type="set" id="1">
  <query xmlns="jabber:iq:auth">
    <username>jsmith</username>

<digest>2E4E532CED2E4E532CED154212BA452145B2A154</dig
est>
    <resource>HomePC</resource>
  </query>
</iq>
```

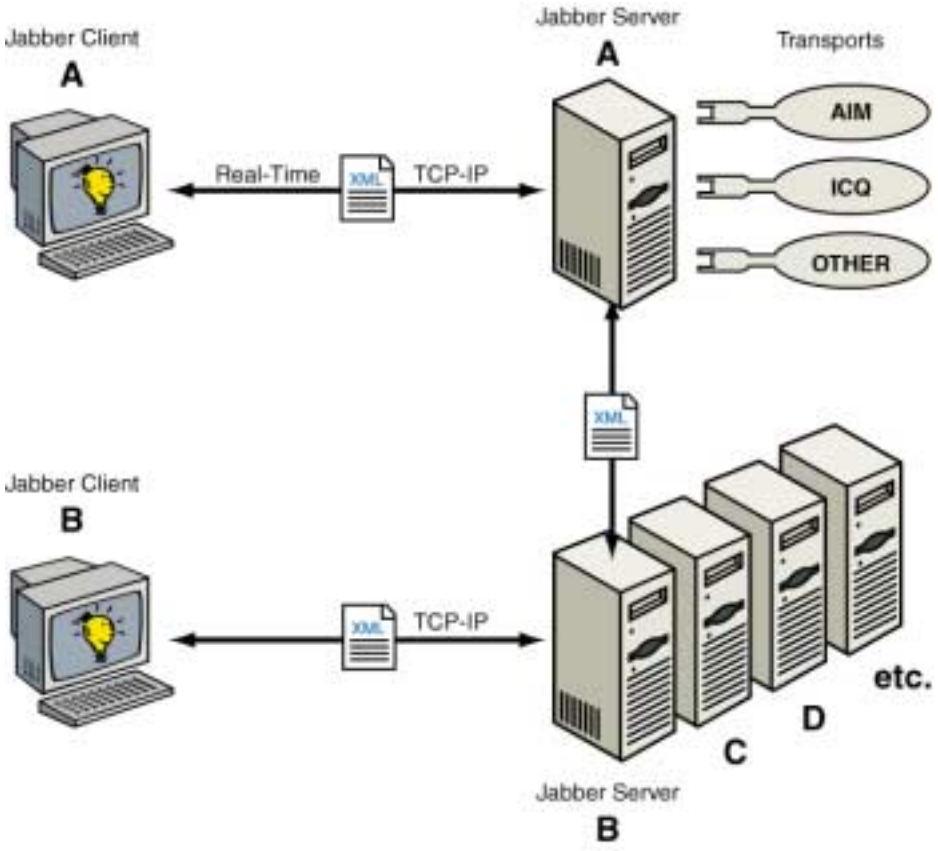
Sample Authentication Response

If authenticated, the server would respond with:

```
<iq type="result" id="1"/>
```

Sending and Receiving Messages

The following illustration shows the interaction between the client and server when a message is sent or received.



Sending messages to other users:

```
<message to="jsmith@jabber.org">
  <subject>Testing</subject>
  <body>This is a test message</body>
</message>
```

Receiving messages from other users:

```
<message from="kpeterson@aim.jabber.com/HomePC">
  <body>This is a test message as well.</body>
</message>
```

More information about the interaction between the client and server is included in "Basic Transport Operations" on page 41.

Rosters

Rosters (or contact lists) are stored on the server so that any client can access them. They are accessed through the jabber:iq:roster namespace.

Clients use the Roster "Push" IQ to manage the roster on the server. The roster is the authoritative list of subscription information for the client's account, including the user's nickname and contact list. It is shared between the client and server.

The roster contains a list of items. Each item element has attributes describing it, and contains group elements for each group it is part of. For example, `name="*"` is an attribute that contains the client's nickname. Clients can control only the Jabber ID (JID), the name attributes, the group elements, and create/remove items. All other attributes are managed by the server depending on how a client responds to presence subscription requests.

Whenever a roster item changes in any way on the server, it is pushed to a client. This push is a normal IQ to the client of the type "set":

```
<iq type="set">
  <query xmlns="jabber:iq:roster">
  <item jid="sjohnson@jabber.com" subscription="to"/>
  </query>
</iq>
```

In the above example, the server sends a roster push to a client to express that the client has a subscription to the presence of sjohnson@jabber.com. A roster push can happen at any time during the connection as a result of another connection modifying the resource or the server modifying the subscription attribute. The client only uses roster pushes to modify the display of the roster. It does not react or prompt the user for every roster push.

Presence

A client sends presence information to the server, which then sends it to all of the people on the roster who are subscribed to the client's presence.

Subscriptions

Presence subscriptions are managed by the server and are stored in the roster. When a user logs onto the Jabber server, it sends an update on that user's presence to all of the people on the user's subscription list.

Server Distribution

- **Singular Model** - The client expresses one presence to the server, which the server then delivers to the subscriptions as listed in the roster.
- **Group/Individual Model** - The client expresses presence individually to each item or group in a roster, and can express different presence statuses to different entities. The client is then responsible for notifying each subscription/item appropriately of their presence and broadcasting it when it changes.
- **Disconnect Handling** - Any time a client sends any type of available presence (presence without a type attribute) the server remembers that the client has notified that recipient of its availability. Upon a server/network disconnect, all those that were notified of an available presence are sent an unavailable presence.

Primary Resource

The server has a knowledge of a "primary" resource for any user account. This is determined using the priority element in presence. When the your client expresses presence to you@your.server, it is broadcast out to all of your own resources and used to determine whether the current client is the primary resource (this is built in when using a singular presence model). The highest priority will be the primary resource. A client with no priority defaults to 0, and can still be the primary resource if no other resources are available. You can only be primary resource when available.

For example, you can leave a Jabber client logged on at home (jsmith@jabber.org/homepc) with a priority of 2. If you log on at work (jsmith@jabber.org/workpc), and set the work client to have a priority of 3, all instant messages are sent to the work client.

Basic Transport Operations

A transport is a special server with the sole purpose of bridging from Jabber to other services (IRC, ICQ, AIM, etc.[*]). When a user logs onto Jabber, a thread is created in the transport to handle all communications to and from that user. In addition, a separate thread is created in the transport for each service that the user is subscribed to. For example, the user would have a separate ICQ thread and a separate AIM thread. Each of these threads is in the appropriate native protocol. For example, the ICQ thread is in the ICQ native protocol.

Each service is handled by one transport process. Within each process, there is a thread for Jabber communications and a thread for the service's native protocol.

Connections

Each connection, using its native protocol library, lives within its own thread. For example, the ICQ connection uses the ICQ protocol library, and all ICQ messages are sent and received in this thread. The thread blocks on the I/O for that connection, and blocks on a message port for the Jabber side.

All incoming Jabber data arrives via Etherx. Every incoming Jabber packet is checked for the `from="jid"` to ascertain the owner of the data. A global table contains all the existing connection threads. The owner of the incoming packet is checked against that list. If there is an existing connection from that owner, the packet is sent via a message port to that connection thread. If the global table contains an owner with an ID of "user@host", any "From" IDs of the type "user@host/resource" will also match and be forwarded.

Registration

Each IQ registration submission starts a special thread to handle the submission. If the native protocol supports new account creation, a

[*] See "Compatibility Disclaimer" on page 7.

thread can be started to create new accounts. If the registration is for an existing account, a thread is started to authenticate and check the credentials submitted. If the registration is successful, the thread confirms the IQ registration (type="result"). If it fails, it returns an error (type="error").

Contact Management

When a user logs on to the server, a notice is sent to each client that has subscribed to the user's presence information. The user may also be subscribed to various IM services so that the presence information is also sent to them. This effectively logs the user onto these services as well. As soon as the user is logged onto a separate IM service, Jabber creates a Contact List that will store the names of the people that the user is subscribed to on the service. For example, when the user sends subscription information to AIM, Jabber creates a contact list to store the names of the people on the user's buddy list for AIM. This list of ID's is used as the approved contact list for the native protocol.

If a packet cannot be delivered to a connection using the Connection protocol, the server stores it in a temporary file called the "buffer". For example, if the user sends notice of its presence to AIM users before sending its presence to the AIM server (i.e. logging on), Jabber stores the subscription in the buffer. The buffer stores all incoming presence availability packets for a short period of time (a minute or two). If the user cannot communicate with the appropriate server, the buffer times out. When connections are established with the appropriate server, the list passes from the buffer into a Contact List.

Presence

With a successful registration, the transport needs to know whenever the owner comes online, so it sends a presence subscription request to the submitter. The special presence subscription is sent with a "from" attribute generated by the transport, with embedded data needed to login to the native protocol. The "from" of the presence subscription might look like:

```
"transport.example.net/registered?from=user@host&user
name=Jane&password=secret"
```

New presence announcements to the special registered resource will send a new connection thread and create an entry in the global table with the registered owner ID. This connection thread, upon logging in, should send presence back to the owner. The connection thread can then use the buffer to extract the approved roster for the native protocol, and should accept new incoming presence announcements from the owner and modify this list appropriately. Any received presence from the native protocol regarding any of the ID's on the approved roster should be sent to the owner with the correct "from" ID.

Messages are bridged from the owner to the appropriate mapped ID in the connection thread. Incoming native protocol messages are sent to the owner with the correct "from" information.

All incoming presence sent to the special registered address should be stored in a presence proxy until it is determined if the owner is available based on the availability of all their resources.

Group Chat

Group chat allows more than two people to participate in a single conversation. The GUI often resembles IRC: a list of participants on one side, the main window for the messages, and a small entry box at the bottom. Within Jabber, a group chat participant is identified via the format `groupname@groupchatserver/nickname`. The single room is identified as `groupname@groupchatserver`, and each participant is identified with a unique resource representing their nickname.

Conclusion

The technology described in this white paper can be implemented in a number of ways. Because Jabber is open-source, a variety of clients have been developed by programmers in the open-source movement. Development of new clients continues on an on-going basis.

Jabber.com, Inc. is also writing client software for Jabber. It is currently releasing JabberIM 1.0 for Windows with the following feature set:

- Communicate with other instant messaging services, including AIM, ICQ, and Yahoo! Messenger*.
- Manage messages, including sending, receiving, filtering, and threading them.
- Queue messages when your client is offline.
- Manage contacts, including finding, viewing, adding, editing, and removing them.
- Send invitations to contacts to participate in conversations.
- Manage groups, including creating and removing them.
- Participate in one-to-one chats, or in group chats.
- Manage your client settings, including changing nicknames, creating accounts, changing server connections, and logging on or off.
- Learn to use JabberIM using online Help.
- Manage your profile to include either minimal or more extensive vCard information.

More information on this technology is available by contacting Jabber.com, Inc. at 303-296-9200 or at http://www.jabber.com.

* See "Compatibility Disclaimer" on page 7.

Glossary

Attribute – An attribute provides further information about an XML element.

Authenticate – Verify that the client is a valid user on the system.

Buffer – A temporary file that holds presence information until the user is logged onto the relevant service.

Client – Software installed on the end user's system that can communicate with Jabber servers and participate in Jabber chat sessions.

Distributed Servers – A system of interlinked, but independent servers. Each server can perform the full functions of the software without reliance on another server. For example, email runs on a system of distributed servers.

DNS - Domain Name Server

Elements – elements describe a document's content and structure. Most elements come in pairs.

Entity – Any separate and distinct presence on Jabber (i.e. a client or a server).

FQDN - Fully Qualified Domain Name

Group Chat – A chat session with three or more participants.

GUI - Graphical User Interface.

Instant Messaging (IM) - text-based conversations in real time.

IQ (Info/Query) – a command for getting information from an entity or setting information for an entity.

IETF (Internet Engineering Task Force) – the group that records recommended protocols for internet development.

IRC – Internet Relay Channel, a service for instant messaging.

Jabber – An XML based system for instant messaging.

Jabber Client Libraries – A client access library currently under development to assist programmers with developing new clients for Jabber®. It will provide the transport information necessary to communicate with Jabber servers, and enable programmers to focus on coding the GUI of their client.

JID (Jabber ID) - Jabber Identifiers are used both externally and internally to express ownership or routing information. The IDs are formed of a domain, node, and resource in the following format: `user@host/resource or[node@]domain[/resource].`

MX – Mail Exchange Server

Native Protocol – The protocol native to a specific service. For example, The AIM transport communicates with AIM using AIM's native protocol.

Open Source – a programming movement characterized by creating "open" code that can be modified or distributed without paying a licensing fee. Members of the movement participate voluntarily in the development of the code and typically publish new code on the Internet for common use. More information about this movement can be found at http://www.opensource.org/osd.html.

Presence – Information indicating whether or not a user is online and available. For example, a user may be online, and have a status of "away".

Primary Resource – The resource to which Jabber directs all communications for a user. For example, a user may be logged onto a home PC and a work PC. The Jabber server directs all communications to the resource with the highest priority (the primary resource).

Probe – A probe searches for presence information.

Protocol – A formalized set of instructions that developers follow when creating Jabber applications.

Resource – The hardware used to access Jabber. For example, a home PC, a work PC, and a palm pilot are separate resources.

Roster – The client's preferences, including the subscription list, stored in the client's account on the Jabber server.

Server – The Jabber server is the software that connects and communicates with other servers. All communications for a client are directed to the server and then passed to the client. All client preferences and data are stored on the server.

Subscription – A subscription is a request to send and/or receive presence information from contacts each time you log onto Jabber.

Syntax – Rules that must be followed when using a protocol to ensure that the attributes, elements and other information is arranged in the correct order to be interpreted by entities on the system.

Transport – A special server with the sole purpose of bridging to other IM services (IRC, ICQ, AIM, etc.).

URI - Uniform Resource Identifier.

XML – eXtensible Markup Language. A specification for designing web documents that allows developers to transmit data between applications or organizations.

Inquiries:
303.296.9200
Fax 303.295.3584
www.webb.net