Network Working Group J.
Miller
Internet-Draft P. Saint-
Andre
Expires: August 22, 2002 Jabber Software
Foundation
J.
Barry
Jabber,
Inc.
February 21,
2002

Jabber
draft-miller-jabber-00

Status of this Memo

This Internet-Draft will expire on August 22, 2002.

Copyright Notice

Abstract

This informational document describes the Jabber protocols, a set of open, XML-based protocols developed over a number of years mainly to provide instant messaging and presence services. In addition, this document describes the known deficiencies of the Jabber protocols.

Table of Contents

1. Introduction

1.1 Overview

Jabber is a set of open, XML-based protocols for which there exist multiple implementations. These implementations have been used mainly to provide instant messaging and presence services that are currently deployed on thousands of domains worldwide and are accessed by millions of IM users daily. Because a standard description of the Jabber protocols is needed to describe this new traffic growing over the Internet, the current document defines the Jabber protocols as they exist today. In addition, this document describes the known deficiencies of the Jabber protocols; however, this document does not address those deficiencies, since they are being addressed through a variety of standards efforts.

1.2 Historical Context

Broad adoption of the Internet occurred only after clear, simple protocols had been developed and accepted by the technical community. These include SMTP [1] for electronic mail and the tandem of HTTP [2] and HTML [3] for document publishing and interactive services offered over the World Wide Web. The authors of this document see two major additional emerging uses of the Internet:

- the near-real-time exchange of text messages (as well as more advanced content) among individuals and applications, enabled by the concepts of presence and availability

- the flexible exchange of structured data between applications, enabled by XML [4] along with related technologies such as XML-RPC [5] and SOAP [6]

The standard transport mechanisms for XML-RPC, SOAP, and other forms of XML data interchange are HTTP and, to a lesser extent, SMTP; yet neither of these mechanisms provides knowledge about the availability of network endpoints, nor are they particularly optimized for the often asynchronous nature of data interchange, especially when such data comes in the form of relatively small payloads as opposed to the larger documents originally envisioned to be the main beneficiaries of XML. By contrast, the existing instant messaging (IM) services have developed fairly robust methods for routing small information payloads to presence-aware endpoints (having built text messaging systems that scale up to millions of concurrent users), but their data formats are unstructured and they have for the most part shunned the standard addressing schemes afforded by URIs [7] and the DNS [8] infrastructure.

Given these considerations, the developers of the Jabber system saw the need for open protocols that would enable the exchange of structured information in an asynchronous, near-real-time manner between any two or more network endpoints, where each endpoint is addressable as a URI and is able to know about the presence and availability of other endpoints on the network. Such protocols, along with associated implementations, would not only provide an alternative (and in many cases more appropriate) transport mechanism for XML data interchange, but also would encourage the development of instant messaging systems that are consistent with Internet standards related to network addressing (URIs, DNS) and structured information (XML).

The Jabber protocols provide just such functionality, since they support asynchronous XML-based messaging and the presence or availability of network endpoints.

1.3 Evolution of Jabber

Definition of the Jabber protocols began in early 1998 with the open-source Jabber server project (jabberd [9]) and associated IM clients. The purpose of the Jabber project was to create an open IM system that would be capable of functioning over diverse networks (e.g., through firewalls) and provide a level of interoperability between other messaging systems. One of the design goals was that a client would need to understand only simple XML data types for messages and presence, with most of the complexity residing on the server. The protocols co-evolved with the server and clients, and the core protocols reached steady-state with release 1.0 in May 2000. Several critical protocol enhancements (most importantly the Dialback Protocol (Section 8.2)) were added with the 1.2 version released in October 2000. It is the protocols as of that date which are documented herein.

Since that time, interest in the Jabber protocols has continued to grow. For example, there now exist at least four server implementations of the protocols as well as countless clients for a wide variety of platforms. In addition, Jabber services have been deployed at thousands of domains on the public Internet and on private intranets, and it is estimated that there are well over a million IM users of Jabber instant messaging services worldwide.

Given the level of interest in Jabber, the authors have decided to document the Jabber protocols and offer the resulting document to the IETF for historical purposes.

1.4 Requirement Levels

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [10].

2. Generalized Architecture

2.1 Overview

Although the Jabber protocols are not wedded to any specific network architecture, to this point they have usually been implemented via a typical client-server architecture, wherein a client utilizing the Jabber protocols accesses a server over a TCP [11] socket. While it is helpful to keep that specific architecture in mind when seeking to understand the Jabber protocols, we have herein abstracted from any specific architecture and have described the architecture in a more generalized fashion.

The following diagram provides a high-level overview of this generalized architecture (where "-" represents communications that use the Jabber XML protocols and "=" represents communications that use any other protocol).

Connection Map

```
    S1       S2
     \      /
N1 -  H1 - H2 - N3
     /  \
N2 -     G1 = I1 = F1
```

The symbols are as follows:

- N1, N2, N3 - Nodes on the Jabber network

- H1, H2 - Hosts on the Jabber network

- S1, S2 - Services that add functionality to a primary host

- G1 - A gateway that translates between the Jabber XML protocols and the protocol(s) used on a non-Jabber instant messaging network

- I1 - A non-Jabber instant messaging network

- F1 - A foreign node on a non-Jabber instant messaging network

2.2 Host

A host acts as an intelligent abstraction layer for core Jabber communications. Its primary responsibility is to manage connections from or sessions for other entities (authorized nodes, services, and other hosts) and to route appropriately-addressed XML data among such

entities. Most Jabber hosts also assume responsibility for the storage of data that is used by nodes or services (e.g., the contact list for each IM user, called in Jabber a "roster"); in this case, the XML data is processed directly by the host itself on behalf of the node or service and is not routed to another entity.

2.3 Node

Most nodes connect directly to a host over a TCP socket and use the Jabber XML protocols to take full advantage of the functionality provided by a host and its associated services. (Nodes on non-Jabber instant messaging networks are also part of the architecture, made accessable via a gateway to that network.) Multiple resources (e.g., devices or locations) may connect to a host on behalf of each authorized node, with each resource connecting over a discrete TCP socket and differentiated by the resource identifier of a Jabber Identifier (Section 3) (e.g., node@host/home vs. node@host/work). The port assigned by the IANA [12] for connections between a Jabber node and a Jabber host is 5222.

2.4 Service

In addition to the core functionality provided by a host, additional functionality is made possible by connecting trusted services to a host. Examples include multi-user chat (a.k.a. conferencing), real-time alert systems, custom authentication modules, database connectivity, and translation to non-Jabber messaging protocols. There is no set port on which services communicate with hosts; this is left up to the administrator of the service or host.

2.4.1 Gateway

A gateway is a special-purpose service whose primary function is to translate the Jabber XML protocols into the protocol(s) of another messaging system, as well as translate the return data back into Jabber XML. Examples are gateways to Internet Relay Chat (IRC), Short Message Service (SMS), SMTP, and non-Jabber instant messaging networks such as Yahoo!, MSN, ICQ, and AIM.

2.5 Network

Because each Jabber host is identified by a network address (typically a DNS hostname) and because host-to-host communications are a simple extension of the Jabber node-to-host protocol, in practice the Jabber system consists of a network of Jabber hosts that inter-communicate. Thus node-a@host1 is able to exchange messages, presence, and other information with node-b@host2. This pattern is familiar from other standards-based messaging protocols, such as

SMTP. The usual method for providing a connection between two Jabber hosts is to open a TCP socket on the IANA-assigned port 5269 and negotiate a connection using the Dialback Protocol (Section 8.2).

3. Jabber Entities

3.1 Overview

Any entity that can be considered a network endpoint (i.e., an ID on the network) and that can communicate using the Jabber protocols is considered a Jabber Entity. All Jabber Entities are uniquely addressable in a form that is nearly consistent with the URI specification (see Section 13.3 for details). In particular, a valid Jabber Identifier (JID) contains a set of ordered elements formed of a domain identifier, node identifier, and resource identifier in the following format: [node@]domain[/resource].

All Jabber Identifiers are based on the foregoing structure. The most common use of this structure is to identify an IM user, the host to which the user connects, and the user's active sessions in the form of user@host/resource. However, other nodes are possible; for example, room-name@conference-service is a specific conference room that is offered by a multi-user chat service.

3.2 Domain Identifier

The domain identifier is the primary identifier and is the only required element of a Jabber Identifier (a simple domain identifier is a valid Jabber Identifier). It usually represents the network gateway or "primary" host to which other entities connect for core XML routing and data management capabilities. However, the entity referenced by a domain identifier is not always a host, and may be a service that is addressed as a subdomain of a host and that provides functionality above and beyond the capabilities of a host (e.g., a multi-user chat service or a gateway to a non-Jabber messaging system). The domain identifier for every host or service that will communicate over a network should resolve to a Fully Qualified Domain Name, and a domain identifier should conform to the specification for DNS names. Comparison of domain identifiers occurs without regard to case for Basic Latin (U+0041 to U+007A) characters.

3.3 Node Identifier

The node identifier is an optional secondary identifier. It usually represents the entity requesting and using network access provided by the host (e.g., a client), although it can also represent other kinds of entities (e.g., a multi-user chat room associated with a conference service). The entity represented by a node identifier is addressed within the context of a specific domain. Node identifiers are restricted to 255 characters. Any Unicode character higher than U+0020 may be included in a node identifier, with the exception of the following:

- o U+0022 (")

- o U+0026 (&)

- o U+0027 (')

- o U+003a (:)

- o U+003C (<)

- o U+003E (>)

- o U+0040 (@)

Comparison of node identifiers occurs directly without regard to case or other syntatic differences.

3.4 Resource Identifier

The resource identifer is an optional third identifier. It represents a specific session, connection (e.g., a device or location), or object (e.g., a participant in a multi-user chat room) belonging to a node. A node may maintain multiple resources simultaneously. There are no restrictions on the length of a resource identifier and any valid XML character is allowed in a resource identifer (as defined in Section 2.2 of the XML 1.0 specification [4], and as suitably escaped if necessary for inclusion within an XML stream). Comparison of resource identifiers is case sensitive for Basic Latin (U+0041 to U+007A) characters.

4. XML Usage within the Jabber Protocols

4.1 Overview

In essence, Jabber consists of three interrelated protocols:

1. XML streams (Section 5), which provide a means for transporting data in an asynchronous manner from one Jabber Entity to another

2. common data types (Section 6) (message, presence, and iq), which provide a framework for communications between Jabber Entities

3. standard extended namespaces (Section 7), which address more specific areas of functionality such as registration, authentication, and the handling of information related to nodes and services

XML [4] is used to define each of these protocols, as described in detail in the following sections.

4.2 Namespaces

XML Namespaces [13] are used within all Jabber XML to create strict boundaries of data ownership. The basic function of namespaces is to separate different vocabularies of XML elements that are structurally mixed together. Ensuring that Jabber's XML is namespace-aware enables any XML to be structurally mixed with any data element within the protocols. This feature is relied upon frequently within the protocols to separate the XML that is processed by different services. There are two main uses of XML namespaces within Jabber: to define XML Streams (Section 5) and to define Standard Extended Namespaces (Section 7).

4.3 Validation

A Jabber host is not responsible for validating the XML elements forwarded to a node; an implementation may choose to provide only validated data elements but is not required to do so. Nodes and services should not rely on the ability to send data which does not conform to the schemas, and should handle any non-conformant elements or attributes on the incoming XML stream by ignoring them.

5. XML Streams

5.1 Overview

Two fundamental concepts make possible the rapid, asynchronous exchange of relatively small payloads of structured information between presence-aware entities: XML streams and, as a result, discrete units of structured information that are referred to as "XML chunks". (Note: in this overview we use the example of communications between a node and host, however XML streams are more generalized and are used in Jabber for communications between a wide range of entities [see Section 5.2].)

On connecting to a Jabber host, a node initiates an XML stream by sending a properly namespaced <stream:stream> tag, and the host replies with a second XML stream back to the node. Within the context of an XML stream, a sender can route a discrete semantic unit of structured information to any recipient. This unit of structured information is a well-balanced XML chunk, such as a message, presence, or iq chunk (a chunk of an XML document is said to be well-balanced if it matches production [43] content of XML 1.0 specification [4]). These chunks exist at the direct child level (depth=1) of the root stream element. The start of any XML chunk is unambiguously denoted by the element start tag at depth=1 (e.g., <presence>) and the end of any XML chunk is unambiguously denoted by the corresponding close tag at depth=1 (e.g., </presence>). Each XML chunk may contain child elements or CDATA sections as necessary in order to convey the desired information from the sender to the recipient. The session is closed at the node's request by sending a closing </stream:stream> tag to the host.

Thus a node's session with a host can be seen as two open-ended XML documents that are built up through the accumulation of the XML chunks that are sent over the course of the session (one from the node to the host and one from the host to the node). In essence, an XML stream acts as an envelope for all the XML chunks sent during a session. We can represent this graphically as follows:

```
|-------------------|
| open stream       |
|-------------------|
| <message to=''>   |
|   <body/>         |
| </message>        |
|-------------------|
| <presence to=''>  |
|   <show/>         |
| </presence>       |
|-------------------|
| <iq to=''>        |
|   <query/>        |
| </iq>             |
|-------------------|
| close stream      |
|-------------------|
```

5.2 Scope

XML streams function as containers for any XML chunks sent asynchronously between network endpoints. (We now generalize those endpoints by using the terms "initiating entity" and "receiving entity".) XML streams are used within Jabber for the following types of communication:

- o Node to Host
- o Host to Host
- o Service to Host

These usages are differentiated through the inclusion of a namespace declaration in the stream from the initiating entity, which is mirrored in the reply from the receiving entity:

- o For node-to-host (and by extension host-to-node communications), the namespace declaration is "jabber:client".
- o For host-to-host commmunications, the namespace declaration is "jabber:server".
- o Communications between a host and a trusted service are slightly more complex, since there are two ways that a service and a host can communicate (for detailed information, see Section 8.3):
  - * The service initiates communications to the host. In this case

the namespace declaration is "jabber:component:accept" (since the host "accepts" communications from the service).

- The host initiates communications to the service. In this case the namespace declaration is "jabber:component:connect" (since the host "connects" to the service).

The common data types (Section 6) are consistent across all three of these namespaces, as are many of the standard extended namespaces (Section 7) (though not all of the latter are relevant to each type of communication; use of the standard extended namespaces is optional for any given implementation).

5.3 Restrictions

XML streams are used to transport a subset of XML. Specifically, XML streams should not contain processing instructions, non-predefined entities (as defined in Section 4.6 of the XML 1.0 specification [4]), comments, or DTDs. Any such XML data should be ignored.

5.4 Formal Definition

The attributes of the stream element are as follows:

- to - The 'to' attribute is normally used only in the XML stream from the initiating entity to the receiving entity, and is set to the Jabber Identifier of the receiving entity. Normally there is no 'to' attribute set in the XML stream by which the receiving entity replies to the initiating entity, however there is no prohibition on such attributes and the should be ignored.

- from - The 'from' attribute is normally used only in the XML stream from the receiving entity to the initiating entity, and is set to the Jabber Identifier of the receiving entity granting access to the initiating entity. Normally there is no 'from' attribute on the XML stream sent from the initiating entity to the receiving entity; however, if a 'from' attribute is included it should be ignored.

- id - The 'id' attribute is normally used only in the XML stream from the receiving entity to the initiating entity. It is a unique identifier created by the receiving entity to function as a session key for the initiating entity's session with the receiving entity. Normally there is no 'id' attribute on the XML stream sent from the initiating entity to the receiving entity; however, if an 'id' attribute is included it should be ignored.

We can summarize these values as follows:

	initiating to receiving	receiving to initiating
to	JID of receiver	ignored
from	ignored	JID of receiver
id	ignored	session key

The stream element also contains the following namespace declarations:

- xmlns - The 'xmlns' namespace declaration is required and is used in both XML streams in order to scope the allowable first-level children of the stream element for both streams. This namespace declaration must be the same for the initiating stream and the responding stream so that both streams are scoped consistently. For allowable values of this namespace declaration, see Section 5.2.

- xmlns:stream - The 'xmlns:stream' namespace declaration is required in both XML streams. The only allowable value is "http://etherx.jabber.org/streams".

The stream element may also contain the following child element:

- error - signifies that a stream-level error has occurred (see Section 5.7)

5.5 DTD

```
<!ELEMENT stream (#PCDATA | error?)*>
<!ATTLIST stream
  to             CDATA  #REQUIRED
  from           CDATA  #IMPLIED
  id             CDATA  #IMPLIED
<!ELEMENT error (#PCDATA)>
```

5.6 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://etherx.jabber.org/streams'
       xmlns='http://etherx.jabber.org/streams'
       elementFormDefault='qualified'>

     <xsd:element name='stream'>
       <xsd:complexType mixed='true'>
         <xsd:element ref='error' minOccurs='0' maxOccurs='1'/>
         <xsd:choice>
           <xsd:any namespace='jabber:client' maxOccurs='1'/>
           <xsd:any namespace='jabber:component:accept' maxOccurs='1'/>
           <xsd:any namespace='jabber:component:connect' maxOccurs='1'/
>
           <xsd:any namespace='jabber:server' maxOccurs='1'/>
         </xsd:choice>
         <xsd:attribute name='to' type='xsd:string' use='optional'/>
         <xsd:attribute name='from' type='xsd:string' use='optional'/>
         <xsd:attribute name='id' type='xsd:string' use='optional'/>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='error' type='xsd:string'/>

   </xsd:schema>
```

5.7 Stream Errors

Errors may occur at the level of the stream. Examples are the sending of invalid XML, the shutdown of a host, an internal server error such as the shutdown of a session manager, and an attempt by a node to authenticate as the same resource that is currently connected. If an error occurs at the level of the stream, the Jabber Entity (initiating entity or receiving entity) that detects the error should send a stream error to the other entity specifying why the streams are being closed and then send a closing </stream:stream> tag. XML of the following form is sent within the context of an existing stream:

```
   <stream:stream ...>
   ...
   <stream:error>
     Error message (e.g., "Invalid XML")
   </stream:error>
   </stream:stream>
```

5.8 Example

The following is a simple stream-based session of a node on a host (where the NODE lines are sent from the node to the host, and the HOST lines are sent from the host to the node):

A simple session:

```
NODE: <stream:stream
          to='host'
          xmlns='jabber:client'
          xmlns:stream='http://etherx.jabber.org/streams'>
HOST: <stream:stream
          from='host'
          id='id_123456789'
          xmlns='jabber:client'
          xmlns:stream='http://etherx.jabber.org/streams'>
[authentication]
NODE:   <message from='node@host' to='receiving-ID'>
NODE:     <body>Watson come here, I need you!</body>
NODE:   </message>
HOST:   <message from='receiving-ID' to='node@host'>
HOST:     <body>I'm on my way!</body>
HOST:   </message>
NODE: </stream:stream>
HOST: </stream:stream>
```

These are in actuality a sending stream and a receiving stream, which could be viewed as two XML documents (i.e., a-chronologically) in the following way:

```
   NODE: <stream:stream
             to='host'
             xmlns='jabber:client'
             xmlns:stream='http://etherx.jabber.org/streams'>
   NODE:   <message from='node@host' to='receiving-ID'>
   NODE:     <body>Watson come here, I need you!</body>
   NODE:   </message>
   NODE: </stream:stream>

   HOST: <stream:stream
             from='host'
             id='id_123456789'
             xmlns='jabber:client'
             xmlns:stream='http://etherx.jabber.org/streams'>
   HOST:   <message from='receiving-ID' to='node@host'>
   HOST:     <body>I'm on my way!</body>
   HOST:   </message>
   HOST: </stream:stream>
```

A session gone bad:

```
   NODE: <stream:stream
             to='host'
             xmlns='jabber:client'
             xmlns:stream='http://etherx.jabber.org/streams'>
   HOST: <stream:stream
             from='host'
             id='id_123456789'
             xmlns='jabber:client'
             xmlns:stream='http://etherx.jabber.org/streams'>
   [authentication]
   NODE: <message><body>Bad XML, no closing body tag!</message>
   HOST: <stream:error>Invalid XML</stream:error>
   HOST: </stream:stream>
```

6. Common Data Types

6.1 Overview

The common data types (Section 6) for Jabber communications are message, presence, and iq. These data types are sent as XML chunks (i.e., direct children) of the root stream element.

6.2 The Message Element

This section describes the valid attributes and child elements of the message element.

6.2.1 Attributes

A message chunk may possess the following attributes:

- to - Specifies the intended recipient of the message chunk. Within the context of communications between a node and host, the absence of a 'to' attribute implies that the XML chunk is addressed to the node@host sending the chunk. Chunks lacking a 'to' attribute or addressed to node@host are processed by the host on behalf of the node@host. Chunks addressed to node@host/ resource are sent to a specific connected resource associated with the node.

- from - Specifies the sender of the message chunk. Within the context of communications between a node and host, the absence of a 'from' attribute implies that the XML chunk is addressed from the node@host/resource sending the chunk. A node may specify any resource, but the host must verify that the node@host matches that of the connected node (this is to prevent spoofing).

- id - An optional unique identifier for the purpose of tracking messages. The sender of the message chunk sets this attribute, which may be returned in any replies.

- type - Used to capture the conversational context of the message, thus providing a hint regarding presentation (e.g., in a GUI). If no type is set or if the type is set to a value other than those specified here, the value should be defaulted by the host to "normal". The type should be one of the following:

  * normal - Single message

  * chat - Traditional two-way chat between two entities

- groupchat - Chat among multiple entities (for details, see

Multi-User Chat (Section 11))

- headline - Ticker or active list of items (e.g., news, sports scores, stock market quotes)

- error - See the error element (Appendix A)

6.2.2 Children

A message chunk may contain zero or one of each of the following child elements (which may not contain mixed content):

- body - The textual contents of the message (must have no attributes); normally included but not required

- subject - The (optional) subject of the message (must have no attributes)

- thread - An optional random string that is generated by the originating node and that may be copied back in replies; it is used for tracking a conversation thread (must have no attributes)

- error - See the error element (Appendix A).

Note: a message element may house an optional element containing content that extends the meaning of the core message (e.g., an encrypted form of the message body). In Jabber usage this child element is often the <x/> element but can be any element. The child element must possess an 'xmlns' namespace declaration (other than those defined for XML streams) that defines all elements contained within the child element. The XML data contained within this element is outside the scope of this document, except for the specific uses of the <x/> element defined in standard extended namespaces (Section 7). If an entity does not understand such a child element or its namespace, it must ignore the associated XML data.

6.2.3 DTD

```
   <!ELEMENT message (( body? | subject? | thread? |
                        error? | (#PCDATA) )*)>

   <!ATTLIST message
     to CDATA #IMPLIED
     from CDATA #IMPLIED
     id CDATA #IMPLIED
     type ( normal | chat | groupchat | headline | error ) #IMPLIED
   >

   <!ELEMENT body (#PCDATA)>
   <!ELEMENT subject (#PCDATA)>
   <!ELEMENT thread (#PCDATA)>
   <!ELEMENT error (#PCDATA)>
   <!ATTLIST error code CDATA #REQUIRED>
```

6.2.4 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='message'>
        <xsd:complexType mixed='true'>
           <xsd:choice>
              <xsd:element ref='body' minOccurs='0' maxOccurs='1'/>
              <xsd:element ref='subject' minOccurs='0' maxOccurs='1'/>
              <xsd:element ref='thread' minOccurs='0' maxOccurs='1'/>
              <xsd:element ref='error' minOccurs='0' maxOccurs='1'/>
              <xsd:any
                  namespace='##other'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
           </xsd:choice>
           <xsd:attribute name='to' type='xsd:string' use='optional'/>
           <xsd:attribute name='from' type='xsd:string' use='optional'/
>
           <xsd:attribute name='id' type='xsd:string' use='optional'/>
           <xsd:attribute name='type' use='optional' default='normal'>
             <xsd:simpleType>
               <xsd:restriction base='xsd:NCName'>
                 <xsd:enumeration value='normal'/>
                 <xsd:enumeration value='chat'/>
```

```
                 <xsd:enumeration value='groupchat'/>
                 <xsd:enumeration value='headline'/>
                 <xsd:enumeration value='error'/>
               </xsd:restriction>
             </xsd:simpleType>
           </xsd:attribute>
        </xsd:complexType>
     </xsd:element>

     <xsd:element name='body' type='xsd:string'/>
     <xsd:element name='subject' type='xsd:string'/>
     <xsd:element name='thread' type='xsd:string'/>
     <xsd:element name='error'>
       <xsd:complexType>
         <xsd:attribute
             name='code'
             type='xsd:nonNegativeInteger'
             use='required'/>
       </xsd:complexType>
     </xsd:element>

   </xsd:schema>
```

6.2.5 Examples

The following examples have been processed by each sender's host and contain the 'from' attribute (node@host/resource) of the sender. (For examples of messages of type "groupchat", see Section 11.)

A simple message:

```
<message to="romeo@montague.net" from="juliet@capulet.com/balcony">
  <subject>Imploring</subject>
  <body>Wherefore art thou, Romeo?</body>
</message>
```

A simple threaded conversation:

```
<message
    to="romeo@montague.net/orchard"
    from="juliet@capulet.com/balcony"
    type="chat">
  <body>Art thou not Romeo, and a Montague?</body>
  <thread>283461923759234</thread>
</message>

<message
    to="juliet@capulet.com/balcony"
    from="romeo@montague.net/orchard"
    type="chat">
  <body>Neither, fair saint, if either thee dislike.</body>
  <thread>283461923759234</thread>
</message>

<message
    to="romeo@montague.net/orchard"
    from="juliet@capulet.com/balcony"
    type="chat">
  <body>How cam'st thou hither, tell me, and wherefore?</body>
  <thread>283461923759234</thread>
</message>
```

6.3 The Presence Element

Presence is used to express a Jabber Entity's current availability status and communicate that status to other entities.

6.3.1 Attributes

A presence chunk may possess the following attributes:

- to - Specifies the intended recipient of the presence chunk. Within the context of communications between a node and host, the absence of a 'to' attribute implies that the XML chunk is addressed to the node@host sending the chunk. Chunks lacking a 'to' attribute or addressed to node@host are processed by the host on behalf of the node@host. Chunks addressed to node@host/ resource are sent to a specific connected resource associated with the node.

- from - Specifies the sender of the presence chunk. Within the context of communications between a node and host, the absence of a 'from' attribute implies that the XML chunk is addressed from

the node@host/resource sending the chunk. A node may specify any resource, but the host must verify that the node@host matches that of the connected node (this is to prevent spoofing).

- id - An optional unique identifier for the purpose of tracking presence. The sender of the presence chunk sets this attribute, which may be returned in any replies.

- type - Describes the type of presence. No 'type' attribute, or inclusion of a type not specified here, implies that the sender is available. The type should be one of the following:

  * unavailable - Signals that the node is no longer available for communications.

  * subscribe - The sender wishes to subscribe to the recipient's presence.

  * subscribed - The sender has allowed the recipient to receive their presence.

  * unsubscribe - A notification that an entity is unsubscribing from another entity's presence. The host handles the actual unsubscription, but nodes receive a presence element for notification reasons.

  * unsubscribed - The subscription has been cancelled.

  * probe - A host-to-host query to request an entity's current presence.

  * error - See the error element (Appendix A).

Note: a presence element may house an optional element containing content that extends the meaning of the core presence (e.g., a signed form of the availability status). In Jabber usage this child element is often the <x/> element but can be any element. The child element must possess an 'xmlns' namespace declaration (other than those defined for XML streams) that defines all elements contained within the child element. The XML data contained within this element is outside the scope of this document, except for the specific uses of the <x> element defined in standard extended namespaces (Section 7). If an entity does not understand such a child element or its namespace, it must ignore the associated XML data.

6.3.2 Children

A presence chunk may contain zero or one of each of the following

child elements:

- show - Describes the availability status of a node or specific resource. The values should be one of the following (values other than these four are typically ignored):

    - away - Node or resource is temporarily away.

    - chat - Node or resource is free to chat.

    - xa - Node or resource is away for an extended period ("eXtended Away").

    - dnd - Node or resource is busy ("Do Not Disturb").

- status - An optional natural-language description of availability status. Normally used in conjunction with the show element to provide a detailed description of an availability state (e.g., "In a meeting").

- priority - A positive integer representing the priority level of the connected resource, with zero as the lowest priority.

- error - See the error element (Appendix A).

6.3.3 DTD

```
<!ELEMENT presence (( show? | status? | priority? | error? )*)>

<!ATTLIST presence
  to CDATA #IMPLIED
  from CDATA #IMPLIED
  type ( subscribe | subscribed | unsubscribe |
         unsubscribed | available | unavailable | error ) #IMPLIED
>

<!ELEMENT show (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
<!ELEMENT error (#PCDATA)>
<!ATTLIST error code CDATA #REQUIRED>
```

6.3.4 Schema

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

```
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='presence'>
       <xsd:complexType>
         <xsd:choice>
           <xsd:element ref='show' minOccurs='0' maxOccurs='1'/>
           <xsd:element ref='status' minOccurs='0' maxOccurs='1'/>
           <xsd:element ref='priority' minOccurs='0' maxOccurs='1'/>
           <xsd:element ref='error' minOccurs='0' maxOccurs='1'/>
           <xsd:any
               namespace='##other'
               minOccurs='0'
               maxOccurs='unbounded'/>
         </xsd:choice>
         <xsd:attribute name='to' type='xsd:string' use='optional'/>
         <xsd:attribute name='from' type='xsd:string' use='optional'/>
         <xsd:attribute name='type' use='optional'>
           <xsd:simpleType>
             <xsd:restriction base='xsd:string'>
               <xsd:enumeration value='subscribe'/>
               <xsd:enumeration value='subscribed'/>
               <xsd:enumeration value='unsubscribe'/>
               <xsd:enumeration value='unsubscribed'/>
               <xsd:enumeration value='available'/>
               <xsd:enumeration value='unavailable'/>
               <xsd:enumeration value='error'/>
             </xsd:restriction>
           </xsd:simpleType>
         </xsd:attribute>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='show'>
       <xsd:simpleType>
         <xsd:restriction base='xsd:string'>
           <xsd:enumeration value='away'/>
           <xsd:enumeration value='chat'/>
           <xsd:enumeration value='xa'/>
           <xsd:enumeration value='dnd'/>
         </xsd:restriction>
       </xsd:simpleType>
     </xsd:element>
     <xsd:element name='status' type='xsd:string'/>
     <xsd:element name='priority' type='xsd:nonNegativeInteger'/>
     <xsd:element name='error'>
```

```
      <xsd:complexType>
        <xsd:attribute
            name='code'
            type='xsd:nonNegativeInteger'
            use='required'/>
      </xsd:complexType>
    </xsd:element>

  </xsd:schema>
```

6.3.5 Examples

Initial presence sent to host upon login to express default availability:

```
<presence/>
```

Receiving detailed presence from another node:

```
<presence to="romeo@montague.net" from="juliet@capulet.com">
  <show>xa</show>
  <status>sleeping</status>
  <priority>1</priority>
</presence>
```

Presence sent to host upon logging off to express unavailable state:

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

A request to subscribe to a node's presence:

```
<presence
    to="juliet@capulet.com"
    from="romeo@montague.net"
    type="subscribe"/>
```

Acceptance of a presence subscription request:

```
<presence
    to="romeo@montague.net"
    from="juliet@capulet.com"
    type="subscribed"/>
```

Denial of a presence subscription request, or cancellation of a previously granted subscription request:

```
<presence
    to="romeo@montague.net"
    from="juliet@capulet.com"
    type="unsubscribed"/>
```

Notification of unsubscribing from a node's presence:

```
<presence
    to="romeo@montague.net"
    from="juliet@capulet.com"
    type="unsubscribe"/>
```

6.4 The IQ Element

Info/Query, or IQ, is a simple request-response mechanism. Just as HTTP is a request-response medium, the iq element enables an entity to make a request and receive a response from another entity.

The actual content of the request and response is defined by the namespace declaration of a direct child element of the iq element. Any direct child element of the iq element must possess an 'xmlns' namespace declaration (other than those defined for XML streams) that defines all elements and attributes contained within that child element. For details, see Section 7.

6.4.1 Attributes

An iq chunk may possess the following attributes:

- to - Specifies the intended recipient of the iq chunk. Within the context of communications between a node and host, the absence of a 'to' attribute implies that the XML chunk is addressed to the node@host sending the chunk. Chunks lacking a 'to' attribute or addressed to node@host are processed by the host on behalf of the node@host. Chunks addressed to node@host/resource are sent to a specific connected resource associated with the node.

- from - Specifies the sender of the iq chunk. Within the context of communications between a node and host, the absence of a 'from' attribute implies that the XML chunk is addressed from the node@host/resource sending the chunk. A node may specify any resource, but the host must verify that the node@host matches that of the connected node (this is to prevent spoofing).

- id - An optional unique identifier for the purpose of tracking the information exchange. The sender of the IQ chunk sets this attribute, which may be returned in any replies.

- type - The required 'type' attribute specifies a distinct step within a request-response conversation. Should be one of the following (all other values are ignored):

  * get - Indicates that the current request is a question or search for information.

  * set - This request contains data intended to set values or replace existing values.

  * result - This is a successful response to a get/set request. If the request was successful, the iq element of type "result" is normally empty.

  * error - The request failed. See the error element (Appendix A).

6.4.2 Children

In the strictest terms, the iq element contains no children since it is a vessel for XML in another namespace. In operation, a query element is usually contained within the iq element as defined by its own separate namespace. See Standard Extended Namespaces (Section 7).

6.4.3 DTD

```
<!ELEMENT iq ( error | (#PCDATA) )*>

<!ATTLIST iq
  to CDATA #IMPLIED
  from CDATA #IMPLIED
  id CDATA #IMPLIED
  type ( get | set | result | error ) #REQUIRED
>

<!ELEMENT error (#PCDATA)>
<!ATTLIST error code CDATA #REQUIRED>
```

6.4.4 Schema

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

```
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='iq'>
       <xsd:complexType mixed='true'>
         <xsd:choice>
           <xsd:element ref='error' minOccurs='0' maxOccurs='1'/>
           <xsd:any
               namespace='##other'
               minOccurs='0'
               maxOccurs='unbounded'/>
         </xsd:choice>
         <xsd:attribute name='to' type='xsd:string' use='optional'/>
         <xsd:attribute name='from' type='xsd:string' use='optional'/>
         <xsd:attribute name='id' type='xsd:string' use='optional'/>
         <xsd:attribute name='type' use='required'>
           <xsd:simpleType>
             <xsd:restriction base='xsd:string'>
               <xsd:enumeration value='get'/>
               <xsd:enumeration value='set'/>
               <xsd:enumeration value='result'/>
               <xsd:enumeration value='error'/>
             </xsd:restriction>
           </xsd:simpleType>
         </xsd:attribute>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='error'>
       <xsd:complexType>
         <xsd:attribute
             name='code'
             type='xsd:nonNegativeInteger'
             use='required'/>
       </xsd:complexType>
     </xsd:element>

   </xsd:schema>
```

6.4.5 Examples

Most IQ examples follow a common pattern of structured data exchange such as get/result or set/result:

```
   Requesting               Responding
     Entity                   Entity
   ----------               ----------
       |                        |
       |    <iq type="get">     |
       | ---------------------> |
       |                        |
       |   <iq type="result">   |
       | <--------------------- |
       |                        |
       |    <iq type="set">     |
       | ---------------------> |
       |                        |
       |   <iq type="result">   |
       | <--------------------- |
       |                        |
```

For specific examples, see Standard Extended Namespaces (Section 7).

7. Standard Extended Namespaces

7.1 Overview

While the common data types provide a basic level of functionality for instant messaging and presence, Jabber uses XML namespaces to extend the common data types for the purpose of providing additional functionality. The extended namespaces accepted by the Jabber Software Foundation all begin with 'jabber:'. (In addition, any of the core data types can be the target of a custom extension using a namespace determined by the creator of that custom extension; however, custom extended namespaces are beyond the scope of this document.) The XML contained in the extension must be defined in the namespace specified in the element that is included as a direct child element of the relevant common data type. This information is often sent within an appropriately-namespaced <query/> element that is a direct child of the iq element, but it can be sent in any element.

There are two types of standard extended namespaces. Namespaces of the first type, which begin with the string 'jabber:iq:', are used within the iq element to facilitate requests and responses between Jabber Entities. These requests usually embody both the action being requested and the data needed for that request, if any.

Namespaces of the second type, which begin with the string 'jabber:x:', are used within the message element (and less frequently the presence element) to send structured information that is specifically related to messages and presence. Jabber Entities can use this type of namespace to effect registration or authentication, send URLs, roster items, offline options, encrypted data, and other information. This information is sent within an appropriately-namespaced <x/> element that is a direct child of the message or presence element.

Many (but not all) of the Standard Extended Namespaces are relevant to communications from node to host, host to host, and service to host. It is up to the implementation to determine which namespaces to support for each type of communication.

The standard iq and x namespaces are described in detail in this section.

7.2 jabber:iq:agent - Agent Properties

The jabber:iq:agent namespace is used to obtain the properties of a specific service associated with a host.

7.2.1 Children

Information about agent properties is contained within a <query/> element that is scoped by the jabber:iq:agent namespace. That query element may contain the following children:

- agent - the reply to a request of type "get" in the jabber:iq:agent namespace contains zero or one <agent/> elements. The <agent/> element has a required 'jid' attribute that contains the Jabber Identifier of the agent. The <agent/> element in turn may contain the following children:

    - name - a natural-language name for the service

    - description - a short phrase describing the service

    - transport - inclusion of this element indicates that the service is a gateway to a non-Jabber instant messaging system

    - groupchat - inclusion of this element indicates that the service is multi-user chat service

    - service - what type of service this is -- values normally included specify the type of gateway (aim, icq, msn, yahoo), the type of conferencing service (public or private), or user directory (jud)

    - register - the service supports registration

    - search - the service supports searching

7.2.2 DTD

```
   <!ELEMENT query (agent)?>

   <!ELEMENT agent (name | description | transport |
                    groupchat | service | register | search)?>
   <!ATTLIST agent
       jid    CDATA    #REQUIRED
   >
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT description (#PCDATA)>
   <!ELEMENT transport (#PCDATA)>
   <!ELEMENT groupchat (#PCDATA)>
   <!ELEMENT service (#PCDATA)>
   <!ELEMENT register (#PCDATA)>
   <!ELEMENT search (#PCDATA)>
```

7.2.3 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='query'>
       <xsd:complexType>
         <xsd:sequence minOccurs='0' maxOccurs='1'>
           <xsd:element ref='agent'/>
         </xsd:sequence>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='agent'>
       <xsd:complexType>
         <xsd:choice minOccurs='0' maxOccurs='1'>
           <xsd:element ref='name'/>
           <xsd:element ref='description'/>
           <xsd:element ref='transport'/>
           <xsd:element ref='groupchat'/>
           <xsd:element ref='service'/>
           <xsd:element ref='register'/>
           <xsd:element ref='search'/>
         </xsd:choice>
         <xsd:attribute name='jid' type='xsd:string' use='required'/>
       </xsd:complexType>
```

```
     </xsd:element>

     <xsd:element name='name' type='xsd:string'/>
     <xsd:element name='description' type='xsd:string'/>
     <xsd:element name='service' type='xsd:string'/>
     <xsd:element name='transport' type='xsd:string'/>
     <xsd:element name='groupchat' type='xsd:string'/>
     <xsd:element name='register' type='xsd:string'/>
     <xsd:element name='search' type='xsd:string'/>

   </xsd:schema>
```

7.2.4 Examples

Request for agent information:

```
   <iq id="i_agent_001" type="get" to="host">
     <query xmlns="jabber:iq:agent"/>
   </iq>
```

Reply from host describing a conferencing component:

```
   <iq id="i_agent_001" type="result" from="host">
     <query xmlns="jabber:iq:agent">
       <agent jid="conference-service">
         <name>Conferencing Service</name>
         <description>
           This service provides multi-user chatrooms.
         </description>
         <service>public</service>
         <groupchat/>
       </agent>
     </query>
   </iq>
```

7.3 jabber:iq:agents - Available Agents

The jabber:iq:agents namespace is used to obtain a list of entities associated with a Jabber Entity. Most commonly this is the list of trusted services associated with a specific host.

7.3.1 Children

Information about available agents properties is contained within a <query/> element that is scoped by the jabber:iq:agents namespace. That query element may contain the following children:

- agent - the reply to a request of type "get" in the jabber:iq:agents namespace contains zero or more <agent/> elements. The <agent/> element has a required 'jid' attribute that contains the Jabber Identifier of each agent. The <agent/> element in turn may contain the following children:

  * name - a natural-language name for the service

  * description - a short phrase describing the service

  * transport - inclusion of this element indicates that the service is a gateway to a non-Jabber instant messaging system

  * groupchat - inclusion of this element indicates that the service is multi-user chat service

  * service - what type of service this is -- values normally included specify the type of gateway (aim, icq, msn, yahoo), the type of conferencing service (public or private), or user directory (jud)

  * register - the service supports registration

  * search - the service supports searching

7.3.2 DTD

```
   <!ELEMENT query (agent)*>

   <!ELEMENT agent (name | description | transport |
                    groupchat | service | register | search)?>
   <!ATTLIST agent
       jid    CDATA    #REQUIRED
   >
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT description (#PCDATA)>
   <!ELEMENT transport (#PCDATA)>
   <!ELEMENT groupchat (#PCDATA)>
   <!ELEMENT service (#PCDATA)>
   <!ELEMENT register (#PCDATA)>
   <!ELEMENT search (#PCDATA)>
```

7.3.3 Schema

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

```
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='query'>
       <xsd:complexType>
         <xsd:sequence minOccurs='0' maxOccurs='unbounded'>
           <xsd:element ref='agent'/>
         </xsd:sequence>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='agent'>
       <xsd:complexType>
         <xsd:choice minOccurs='0' maxOccurs='1'>
           <xsd:element ref='name'/>
           <xsd:element ref='description'/>
           <xsd:element ref='transport'/>
           <xsd:element ref='groupchat'/>
           <xsd:element ref='service'/>
           <xsd:element ref='register'/>
           <xsd:element ref='search'/>
         </xsd:choice>
         <xsd:attribute name='jid' type='xsd:string' use='required'/>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='name' type='xsd:string'/>
     <xsd:element name='description' type='xsd:string'/>
     <xsd:element name='service' type='xsd:string'/>
     <xsd:element name='transport' type='xsd:string'/>
     <xsd:element name='groupchat' type='xsd:string'/>
     <xsd:element name='register' type='xsd:string'/>
     <xsd:element name='search' type='xsd:string'/>

   </xsd:schema>
```

7.3.4 Examples

Request for agents list:

```
<iq id="i_agents_001" type="get" to="host">
  <query xmlns="jabber:iq:agents"/>
</iq>
```

Reply from host:

```
<iq
    to="node@host/resource"
    from="host"
    type="result"
    id="i_agents_001">
  <query xmlns="jabber:iq:agents">
    <agent jid="user-directory">
      <name>Jabber User Directory</name>
      <service>jud</service>
      <search/>
      <register/>
    </agent>
    <agent jid="conference-service">
      <name>Conferencing Service</name>
      <service>public</service>
      <groupchat/>
    </agent>
  </query>
</iq>
```

7.4 jabber:iq:auth - Node Authentication

The jabber:iq:auth namespace provides a simple mechanism for nodes to authenticate and create a resource representing their connection to the host.

7.4.1 Children

- username - the unique username for this node (usually an IM user).

- password - the secret key or passphrase for the node's access to the host.

- digest - the concatenation of the stream id and the password, encrypted according to the SHA1 Secure Hash Algorithm [14] and represented as all lowercase hex.

- resource - unique value to represent current connection.

7.4.2 DTD

```
   <!ELEMENT query ((username? | (password | digest)? | resource)*)>

   <!ELEMENT username (#PCDATA)>
   <!ELEMENT password (#PCDATA)>
   <!ELEMENT digest (#PCDATA)>
   <!ELEMENT resource (#PCDATA)>
```

7.4.3 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='query'>
       <xsd:complexType>
         <xsd:choice minOccurs='0' maxOccurs='unbounded'>
           <xsd:element ref='username'/>
           <xsd:choice minOccurs='0' maxOccurs='1'>
             <xsd:element ref='password'/>
             <xsd:element ref='digest'/>
           </xsd:choice>
           <xsd:element ref='resource'/>
         </xsd:choice>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='username' type='xsd:string'/>
     <xsd:element name='password' type='xsd:string'/>
     <xsd:element name='digest' type='xsd:string'/>
     <xsd:element name='resource' type='xsd:string'/>

   </xsd:schema>
```

7.4.4 Examples

The following is a complete example of how a node authenticates with a host.

Node queries host as to what information is required:

```
<iq type="get" id="i_auth_001">
  <query xmlns="jabber:iq:auth">
    <username>juliet</username>
  </query>
</iq>
```

Host replies:

```
<iq type="result" id="i_auth_001">
  <query xmlns="jabber:iq:auth">
    <username>juliet</username>
    <password/>
    <digest/>
    <resource/>
  </query>
</iq>
```

Node sends authentication information (plaintext password):

```
<iq type="set" id="i_auth_002">
  <query xmlns="jabber:iq:auth">
    <username>juliet</username>
    <password>r0m30</password>
    <resource>balcony</resource>
  </query>
</iq>
```

Node sends authentication information (hashed password):

```
<iq type="set" id="i_auth_002">
  <query xmlns="jabber:iq:auth">
    <username>juliet</username>
    <digest>64d60e40febe09264c52bc9cbddd5dd1147fae97</digest>
    <resource>balcony</resource>
  </query>
</iq>
```

Host confirms login:

```
<iq type="result" id="i_auth_002"/>
```

7.5 jabber:iq:oob - Out-of-Band Data

The jabber:iq:oob namespace provides a standard way to perform node-to-node transmission of information outside the context of the host

(e.g., for the purpose of file transfers). Note that information can be transferred out of band within an iq element using the jabber:iq:oob namespace or within a message element using the jabber:x:oob namespace. It is expected that a Jabber Entity will perform an HTTP HEAD request to determine the MIME type and size of any file before retrieving it from a URL.

7.5.1 Children

- url - a Uniform Resource Locator for the file

- desc - a natural-language description of the file

7.5.2 DTD

```
   <!ELEMENT query ((url? | desc?)*)>

   <!ELEMENT url (#PCDATA)>
   <!ELEMENT desc (#PCDATA)>
```

7.5.3 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='query'>
       <xsd:complexType>
         <xsd:choice minOccurs='0' maxOccurs='unbounded'>
           <xsd:element ref='url'/>
           <xsd:element ref='desc'/>
         </xsd:choice>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='url' type='xsd:string'/>
     <xsd:element name='desc' type='xsd:string'/>

   </xsd:schema>
```

7.5.4 Examples

Node transmits information to another node:

```
   <iq type="set" to="horatio@denmark" from="sailor@sea"
id="i_oob_001">
     <query xmlns="jabber:iq:oob">
       <url>http://denmark/act4/letter-1.html</url>
       <desc>There's a letter for you sir.</desc>
     </query>
   </iq>
```

7.6 jabber:iq:register - Registration

Through the jabber:iq:register namespace, nodes can register with a Jabber host itself or with trusted services of that host.

7.6.1 Children

Note that while numerous fields are available, only the ones returned by a host or service (other than "instructions") are required in order to register with that host or service.

- instructions
- username
- password
- name
- email
- address
- city
- state
- zip
- phone
- url
- date
- misc

- text

- remove - request to unregister

7.6.2 DTD

```
<!ELEMENT query ((instructions? | username? |
    password? | name? | email? | address? |
    city? | state? | zip? | phone? | url? |
    date? | misc? | text? | remove?)*)>

<!ELEMENT instructions (#PCDATA)>
<!ELEMENT username (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT misc (#PCDATA)>
<!ELEMENT text (#PCDATA)>
<!ELEMENT remove EMPTY>
```

7.6.3 Schema

```
<?xml version='1.0' encoding='UTF-8'?>
<xsd:schema
    xmlns:xsd='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://www.jabber.org/protocol'
    xmlns='http://www.jabber.org/protocol'
    elementFormDefault='qualified'>

  <xsd:element name='query'>
    <xsd:complexType>
      <xsd:choice minOccurs='0' maxOccurs='unbounded'>
        <xsd:element ref='instructions'/>
        <xsd:element ref='username'/>
        <xsd:element ref='password'/>
        <xsd:element ref='name'/>
        <xsd:element ref='email'/>
        <xsd:element ref='address'/>
        <xsd:element ref='city'/>
```

```
          <xsd:element ref='state'/>
          <xsd:element ref='zip'/>
          <xsd:element ref='phone'/>
          <xsd:element ref='url'/>
          <xsd:element ref='date'/>
          <xsd:element ref='misc'/>
          <xsd:element ref='text'/>
          <xsd:element ref='remove'/>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name='instructions' type='xsd:string'/>
    <xsd:element name='username' type='xsd:string'/>
    <xsd:element name='password' type='xsd:string'/>
    <xsd:element name='name' type='xsd:string'/>
    <xsd:element name='email' type='xsd:string'/>
    <xsd:element name='address' type='xsd:string'/>
    <xsd:element name='city' type='xsd:string'/>
    <xsd:element name='state' type='xsd:string'/>
    <xsd:element name='zip' type='xsd:string'/>
    <xsd:element name='phone' type='xsd:string'/>
    <xsd:element name='url' type='xsd:string'/>
    <xsd:element name='date' type='xsd:string'/>
    <xsd:element name='misc' type='xsd:string'/>
    <xsd:element name='text' type='xsd:string'/>
    <xsd:element name='remove'/>

  </xsd:schema>
```

7.6.4 Examples

Node request for information required to register with a service:

```
<iq type="get" id="i_reg_001" to="service.denmark">
  <query xmlns="jabber:iq:register"/>
</iq>
```

Host response with registration fields required:

```
<iq type="result"
    from="service.denmark"
    to="hamlet@denmark"
    id="i_reg_001">
  <query xmlns="jabber:iq:register">
    <instructions>
      Choose a username and password to register with this service.
    </instructions>
    <name/>
    <email/>
    <password/>
  </query>
</iq>
```

Node request to register for an account:

```
<iq type="set"
    to="service.denmark"
    from="hamlet@denmark"
    id="i_reg_002">
  <query xmlns="jabber:iq:register">
    <name>hamlet</name>
    <email>hamlet@denmark</email>
    <password>gertrude</password>
  </query>
</iq>
```

Successful registration:

```
<iq
    type="result"
    from="service.denmark"
    to="hamlet@denmark"
    id="i_reg_002"/>
```

Failed registration:

```
<iq
    type="error"
    from="service.denmark"
    to="hamlet@denmark"
    id="i_reg_002"/>
  <error code="406">Not Acceptable</error>
</iq>
```

Node request to unregister:

```
<iq type="set"
    to="service.denmark"
    from="hamlet@denmark"
    id="i_reg_003">
  <query xmlns="jabber:iq:register">
    <remove/>
  </query>
</iq>
```

Successful unregistration:

```
<iq
    type="result"
    from="service.denmark"
    to="hamlet@denmark"
    id="i_reg_003"/>
```

7.7 jabber:iq:roster - Roster Management

The jabber:iq:roster namespace provides a mechanism for managing a node's roster (also known as a "contact list"). Upon connecting to the host, a node should request the roster using jabber:iq:roster. Since the roster may not be desirable for all resources (e.g., cellular phone client), the node's request for the roster is optional.

When a specific connected resource for a node updates the node's roster on the host, the host is responsible for pushing that change out to all connected resources for that node using an iq element of type "set" as seen in the final example within this section. This enables all connected resources to remain in sync with the host-based roster information.

7.7.1 Children

A <query/> element scoped by the jabber:iq:roster namespace may contain zero or more <item/> elements. An item element may contain the following attributes:

- jid - a required attribute that contains the complete Jabber Identifier of the contact that this item represents

- name - an optional attribute that contains a natural-language name for the contact

- subscription - the current status of the subscription related to this item. Should be one of the following (all other values are ignored):

    * none - no subscription.

    * from - this entity has a subscription to the contact.

    * to - the contact has a subscription to this entity.

    * both - subscription is both to and from.

    * remove - item is to be removed.

- ask - An optional attribute specifying the current status of a request to this contact. Should be one of the following (all other values are ignored):

    * subscribe - this entity is asking to subscribe to that contact's presence.

    * unsubscribe - this entity is asking unsubscribe from that contact's presence.

An <item/> element may contain zero or more instances of the following element:

- group - Natural-language name of a user-specified group for the purpose of categorizing contacts into groups.

7.7.2 DTD

```
   <!ELEMENT query ((item)*)>

   <!ELEMENT item ((group)*)>
   <!ATTLIST item
       jid CDATA #REQUIRED
       name CDATA #IMPLIED
       subscription ( to | from | both | none | remove ) #IMPLIED
       ask ( subscribe | unsubscribe ) #IMPLIED
       >
   <!ELEMENT group (#PCDATA)>
```

7.7.3 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='query'>
       <xsd:complexType>
         <xsd:sequence minOccurs='0' maxOccurs='unbounded'>
           <xsd:element ref='item'/>
         </xsd:sequence>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='item'>
       <xsd:complexType>
         <xsd:sequence minOccurs='0' maxOccurs='unbounded'>
           <xsd:element ref='group'/>
         </xsd:sequence>
         <xsd:attribute name='jid' type='xsd:string' use='required'/>
         <xsd:attribute name='name' type='xsd:string' use='optional'/>
         <xsd:attribute name='subscription' use='optional'>
           <xsd:simpleType>
             <xsd:restriction base='xsd:string'>
               <xsd:enumeration value='to'/>
               <xsd:enumeration value='from'/>
               <xsd:enumeration value='both'/>
               <xsd:enumeration value='none'/>
               <xsd:enumeration value='remove'/>
             </xsd:restriction>
           </xsd:simpleType>
```

```
         </xsd:attribute>
         <xsd:attribute name='ask' use='optional'>
           <xsd:simpleType>
             <xsd:restriction base='xsd:string'>
               <xsd:enumeration value='subscribe'/>
               <xsd:enumeration value='unsubscribe'/>
             </xsd:restriction>
           </xsd:simpleType>
         </xsd:attribute>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='group' type='xsd:string'/>

   </xsd:schema>
```

7.7.4 Examples

Node requests current roster from host:

```
   <iq id="i_roster_001" type="get">
     <query xmlns="jabber:iq:roster"/>
   </iq>
```

Node receives the roster from the host:

```
<iq id="i_roster_001" type="result">
  <query xmlns="jabber:iq:roster>
    <item
        jid="juliet@capulet.com"
        name="Juliet"
        subscription="both"/>
    <item
        jid="mercutio@montague.net"
        name="Mercutio"
        subscription="both">
      <group>Friends</group>
    </item>
    <item
        jid="benvolio@montague.net"
        name="Benvolio"
        subscription="both">
      <group>Friends</group>
    </item>
  </query>
</iq>
```

Node adds a new item:

```
<iq type="set" id="i_roster_002">
  <query xmlns="jabber:iq:roster">
    <item
        name="Nurse"
        jid="nurse@capulet.com">
      <group>Servants</group>
    </item>
</iq>
```

Host replies with the updated roster information, plus an IQ result:

```
<iq type="set" id="i_roster_003"/>
  <query xmlns="jabber:iq:roster">
    <item
        name="Nurse"
        jid="nurse@capulet.com">
      <group>Servants</group>
    </item>
</iq>
<iq type="result" id="i_roster_002"/>
```

7.8 jabber:iq:time - Entity Time

The jabber:iq:time namespace provides a standard way for Jabber Entities to exchange local time (e.g., to "ping" another entity or check network latency).

7.8.1 Children

- utc - the time at the responding entity in UTC (the format should be consistent with that defined in ISO 8601 [15])

- tz - the time zone in which the entity is located

- display - human-readable time format

7.8.2 DTD

```
<!ELEMENT query ((utc | tz? | display?)*)>

<!ELEMENT utc (#PCDATA)>
<!ELEMENT tz (#PCDATA)>
<!ELEMENT display (#PCDATA)>
```

7.8.3 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='query'>
       <xsd:complexType>
         <xsd:choice minOccurs='0' maxOccurs='unbounded'>
           <xsd:element ref='utc'/>
           <xsd:element ref='tz'/>
           <xsd:element ref='display'/>
         </xsd:choice>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='utc' type='xsd:string'/>
     <xsd:element name='tz' type='xsd:string'/>
     <xsd:element name='display' type='xsd:string'/>

   </xsd:schema>
```

7.8.4 Examples

Node requests time from another node:

```
   <iq
       type="get"
       to="juliet@capulet.com/balcony"
       from="romeo@montague.net/orchard"
       id="i_time_001">
     <query xmlns="jabber:iq:time"/>
   </iq>
```

Node replies to request:

```
<iq
    type="result"
    to="romeo@montague.net/orchard"
    from="juliet@capulet.com/balcony"
    id="i_time_001">
  <query xmlns="jabber:iq:time">
    <utc>20020214T23:55:06</utc>
    <tz>WET</tz>
    <display>14 Feb 2002 11:55:06 PM</display>
  </query>
</iq>
```

7.9 jabber:iq:version - Entity Version

The jabber:iq:version namespace provides a standard way for Jabber Entities to discover version information about other entities.

7.9.1 Children

- name - a natural-language name for the entity, resource, or application

- version - the specific version

- os - the operating system on which the entity is running

7.9.2 DTD

```
<!ELEMENT query ((name | version | os)?)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT os (#PCDATA)>
```

7.9.3 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='query'>
       <xsd:complexType>
         <xsd:choice minOccurs='0' maxOccurs='1'>
           <xsd:element ref='name'/>
           <xsd:element ref='version'/>
           <xsd:element ref='os'/>
         </xsd:choice>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='name' type='xsd:string'/>
     <xsd:element name='version' type='xsd:string'/>
     <xsd:element name='os' type='xsd:string'/>

   </xsd:schema>
```

7.9.4 Examples

Node requests version information from another node:

```
   <iq
       type="get"
       to="romeo@montague.net/orchard"
       from="juliet@capulet.com/balcony"
       id="i_version_001">
     <query xmlns="jabber:iq:version"/>
   </iq>
```

Node replies to request:

```
<iq
    type="result"
    to="juliet@capulet.com/balcony"
    from="romeo@montague.net/orchard"
    id="i_version_001">
  <query xmlns="jabber:iq:version">
    <name>Gabber</name>
    <version>0.8.6</version>
    <os>Linux i686</os>
  </query>
</iq>
```

7.10 jabber:x:delay - Delayed Delivery

The jabber:x:delay namespace is used to provide timestamp information about data stored for later delivery. The most common uses of this namespace are to stamp:

- a message sent to an offline entity and that is stored for later delivery

- the last presence update sent by a connected node to a host

- messages cached by a multi-user chat room for delivery to new entrants to the room

7.10.1 Attributes

- from - the Jabber Identifier of the location where the XML chunk has been delayed or held for later delivery (for example, the address of a multi-user chat room)

- stamp - a required attribute that contains information about the time when the chunk was originally sent (the format should be consistent with that defined in ISO 8601 [15])

7.10.2 DTD

```
<!ELEMENT x (#PCDATA)>

<!ATTLIST x
    from CDATA #IMPLIED
    stamp CDATA #REQUIRED
    >
```

7.10.3 Schema

```
<?xml version='1.0' encoding='UTF-8'?>
<xsd:schema
    xmlns:xsd='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://www.jabber.org/protocol'
    xmlns='http://www.jabber.org/protocol'
    elementFormDefault='qualified'>

  <xsd:element name='x'>
    <xsd:complexType mixed='true'>
      <xsd:attribute name='from' type='xsd:string' use='optional'/>
      <xsd:attribute name='stamp' type='xsd:string' use='required'/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

7.10.4 Examples

Message sent to an offline node:

```
<message
    to="node-a@host1"
    from="node-b@host2"
    type="chat"
>
  <body>Message me when you log in again.</body>
  <x
      xmlns="jabber:x:delay"
      from="node-a@host1"
      stamp="20020212T23:42:40">
    Offline Storage
  </x>
</message>
```

Last presence update sent by another node:

```
<presence
    to="node-a@host1"
    from="node-b@host2"
  <status>In a meeting for the next two hours.</status>
  <show>xa</show>
  <priority>1</priority>
  <x
      from='node-a@host1'
      stamp='20020212T23:57:03'
      xmlns='jabber:x:delay'/>
</presence>
```

Message sent in a conference room before the recipient arrived and cached for delivery to new entrants:

```
<message
    type="groupchat"
    from="cauldron@conference.witches.org/firstwitch"
    to="thirdwitch@kinglear.org"
>
  <body>Thrice the brinded cat hath mew'd.</body>
  <x
      xmlns="jabber:x:delay"
      from="cauldron@conference.witches.org"
      stamp="10541031T23:53:40">
    Cached In GC History
  </x>
</message>
```

7.11 jabber:x:oob - Out-of-Band Data

The jabber:x:oob namespace enables nodes to exchange special messages that contain URIs along along with a description. It is expected that a node will perform an HTTP HEAD request to determine the MIME type and size of any file before retrieving it from a URL.

7.11.1 Children

- url - a Uniform Resource Locator for the file

- desc - a natural language description of the file

7.11.2 DTD

```
<!ELEMENT x ((url? | desc?)*)>

<!ELEMENT url (#PCDATA)>
<!ELEMENT desc (#PCDATA)>
```

7.11.3 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='x'>
       <xsd:complexType>
         <xsd:choice minOccurs='0' maxOccurs='unbounded'>
           <xsd:element ref='url'/>
           <xsd:element ref='desc'/>
         </xsd:choice>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='url' type='xsd:string'/>
     <xsd:element name='desc' type='xsd:string'/>

   </xsd:schema>
```

7.11.4 Examples

A node sends a message to another node containing information about an out-of-band transfer:

```
   <message from="sailor@sea" to="horatio@denmark">
     <body>URL Attached.</body>
     <x xmlns="jabber:x:oob">
       <url>http://denmark/act4/letter-1.html</url>
       <desc>There's a letter for you sir</desc>
     </x>
   </message>
```

7.12 jabber:x:roster - Embedded Roster Items

The jabber:x:roster namespace is used to send roster items from one Jabber Entity to another.

7.12.1 Children

An <x/> element scoped by the jabber:x:roster namespace may contain zero or more <item/> elements. An item element may contain the following attributes:

- jid - the Jabber Identifier of the item

- name - a natural-language name or nickname for the item

An <item/> element may also contain one or more of the following children:

- group - Natural-language name of a user-specified group for the purpose of categorizing contacts into groups.

7.12.2 DTD

```
<!ELEMENT x ((item)*)>

<!ELEMENT item ((group)*)>
<!ATTLIST item
    jid CDATA #IMPLIED
    name CDATA #IMPLIED
    >
<!ELEMENT group (#PCDATA)>
```

7.12.3 Schema

```
   <?xml version='1.0' encoding='UTF-8'?>
   <xsd:schema
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       targetNamespace='http://www.jabber.org/protocol'
       xmlns='http://www.jabber.org/protocol'
       elementFormDefault='qualified'>

     <xsd:element name='x'>
       <xsd:complexType>
         <xsd:sequence minOccurs='0' maxOccurs='unbounded'>
           <xsd:element ref='item'/>
         </xsd:sequence>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='item'>
       <xsd:complexType>
         <xsd:sequence minOccurs='0' maxOccurs='unbounded'>
           <xsd:element ref='group'/>
         </xsd:sequence>
         <xsd:attribute name='jid' type='xsd:string' use='optional'/>
         <xsd:attribute name='name' type='xsd:string' use='optional'/>
       </xsd:complexType>
     </xsd:element>

     <xsd:element name='group' type='xsd:string'/>

   </xsd:schema>
```

7.12.4 Examples

Sending an embedded roster item to another node:

```
<message to="hamlet@denmark" from="horatio@denmark">
  <subject>Visitors</subject>
  <body>This message contains roster items.</body>
  <x xmlns="jabber:x:roster">
    <item
        jid="rosencrantz@denmark"
        name="Rosencrantz">
      <group>Visitors</group>
    </item>
    <item
        jid="guildenstern@denmark"
        name="Guildenstern">
      <group>Visitors</group>
    </item>
  </x>
</message>
```

8. Authentication Mechanisms

Authentication is any process of verifying that a Jabber Entity is who or what it claims it is. Because nodes, hosts, and services are fundamentally different kinds of entities, authentication is the only area of Jabber communications that has been perceived to necessitate differences at the protocol level (as opposed to implementation level) between the treatment of nodes, hosts, and services. The standard authentication mechanisms are described in this section.

8.1 Authentication of a Node by a Host

The process by which a node is authenticated by a host is defined by the jabber:iq:auth namespace (Section 7.4). This process is used only within XML streams that are declared under the "jabber:client" namespace. (Note: because a host never authenticates with a node, there is no defined protocol by which such authentication would take place.)

8.2 Authentication of a Host by Another Host

8.2.1 Overview

It became obvious to the developers of the Jabber protocol that they needed a method of verifying that a connection between two hosts could be trusted. Because the developers wished to avoid the overhead of building a network of trusted hosts, they sought a protocol-level system that would provide the necessary security. This method is called dialback and is used only within XML streams that are declared under the "jabber:server" namespace.

The dialback protocol is used to prevent spoofing of a particular hostname and sending false data from it. Dialback is made possible by the existence of DNS, since one host can verify that another host which is connecting to it is authorized to represent a given host on the Jabber network. All DNS host resolutions must first resolve the host using an SRV [16] record of _jabber._tcp.host. If the SRV lookup fails, the fallback is a normal A lookup using the jabber-server port of 5269 assigned by the Internet Assigned Numbers Authority [12].

Note that the method used to generate and verify the keys used in the dialback protocol must take into account the hostnames being used, along with a secret known only by the receiving host and the random id on the stream. Generating unique but verifiable keys is important to prevent common man-in-the-middle attacks and host spoofing.

In the description that follows we use the following terminology:

- Originating Host - the host that is attempting to establish a connection between the two hosts

- Receiving Host - the host that is trying to authenticate that the Originating Host represents the Jabber host which it claims to be

- Authoritative Host - the host which is given when a DNS lookup is performed on the name that the Originating Host initially gave; for simple environments this will be the Originating Host, but it could be a separate machine in the Originating Host's network

The following is a brief summary of the order of events in dialback:

1. Originating Host establishes a connection to Receiving Host.

2. Originating Host sends a 'key' value over the connection to Receiving Host.

3. Receiving Host establishes a connection to Authoritative Host.

4. Receiving Host sends the same 'key' value to Authoritative Host.

5. Authoritative Host replies that key is valid or invalid.

6. Receiving Host tells Originating Host whether it is authenticated or not.

We can represent this flow of events graphically as follows:

```
   Originating               Receiving
      Host                    Host
   -----------               ---------
       |                         |
       |  establish connection   |
       | ----------------------> |
       |                         |
       |   send stream header    |
       | ----------------------> |
       |                         |
       |  establish connection   |
       | <---------------------- |
       |                         |
       |   send stream header    |
       | <---------------------- |
       |                         |                   Authoritative
       |   send dialback key     |                       Host
       | ----------------------> |                   -------------
       |                         |                         |
                                 |  establish connection   |
                                 | ----------------------> |
                                 |                         |
                                 |   send stream header    |
                                 | ----------------------> |
                                 |                         |
                                 |   send stream header    |
                                 | <---------------------- |
                                 |                         |
                                 |   send dialback key     |
                                 | ----------------------> |
                                 |                         |
                                 |  validate dialback key  |
                                 | <---------------------- |
                                 |
       |  report dialback result |
       | <---------------------- |
       |                         |
```

8.2.2 Dialback Protocol

The traffic sent between the hosts is as follows:

1. Originating Host establishes connection to Receiving Host

2. Originating Host sends a stream header to Receiving Host (the 'to' and 'from' attributes are not required):

```
          <stream:stream xmlns:stream='http://etherx.jabber.org/
streams'
                         xmlns='jabber:server'
                         xmlns:db='jabber:server:dialback'>
```

Note: the value of the xmlns:db namespace declaration indicates to Receiving Host that Originating Host supports dialback.

3. Receiving Host sends a stream header back to Originating Host (the 'to' and 'from' attributes are not required):

```
          <stream:stream xmlns:stream='http://etherx.jabber.org/
streams'
                         xmlns='jabber:server'
                         xmlns:db='jabber:server:dialback'
                         id='457F9224A0...'>
```

4. Originating Host sends a dialback key to Receiving Host:

```
              <db:result
                  to='Receiving Host'
                  from='Originating Host'>98AF014EDC0...</db:result>
```

Note: this key is not examined by Receiving Host, since the Receiving Host does not keep information about Originating Host between sessions.

5. Receiving Host now establishes a connection back to Originating Host, getting the Authoritative Host.

6. Receiving Host sends Authoritative Host a stream header (the 'to' and 'from' attributes are not required):

```
          <stream:stream xmlns:stream='http://etherx.jabber.org/
streams'
                         xmlns='jabber:server'
                         xmlns:db='jabber:server:dialback'>
```

7. Authoritative Host sends Receiving Host a stream header:

```
          <stream:stream xmlns:stream='http://etherx.jabber.org/
streams'
                         xmlns='jabber:server'
                         xmlns:db='jabber:server:dialback'
                         id='1251A342B...'>
```

8. Receiving Host sends Authoritative Host a chunk indicating it wants Authoritative Host to verify a key:

```
<db:verify from='Receiving Host' to='Originating Host'
           id='457F9224A0...'>98AF014EDC0...</db:verify>
```

Note: passed here are the hostnames, the original identifier from Receiving Host's stream header to Originating Host in step 2, and the key Originating Host gave Receiving Host in step 3. Based on this information and shared secret information within the 'Originating Host' network, the key is verified. Any verifiable method can be used to generate the key.

9. Authoritative Host sends a chunk back to Receiving Host indicating whether the key was valid or invalid:

```
<db:result
    from='Originating Host'
    to='Receiving Host'
    type='valid'
    id='457F9224A0...'/>
```

or

```
<db:result
    from='Originating Host'
    to='Receiving Host'
    type='invalid'
    id='457F9224A0...'/>
```

10. Receiving Host informs Originating Host of the result:

```
<db:result
    from='Receiving Host'
    to='Originating Host'
    type='valid'/>
```

Note: At this point the connection has either been validated via a type='valid', or reported as invalid. Once the connection is validated, data can be sent by the Originating Host and read by the Receiving Host; before that, all data chunks sent to Receiving Host are dropped. As a final guard against domain spoofing, the Receiving Host must validate all XML chunk received from the Originating Host to verify that the from address of each chunk includes the validated domain.

8.3 Authentication of Services

As noted under Section 5.2, there are two ways that a service and a host can communicate:

1. The service initiates communications to the host. In this case the namespace declaration is "jabber:component:accept" (since the host "accepts" communications from the service).

2. The host initiates communications to the service. In this case the namespace declaration is "jabber:component:connect" (since the host "connects" to the service).

The authentication methods for these communication directions are defined in this section.

8.3.1 Authentication of a Service by a Host

When a service initiates a connection to a host, the host will want to verify the identity of the service so that it knows whether the service can be trusted. The first step is to negotiate the XML streams between the service and the host, scoped within the "jabber:component:accept" namespace:

Initiation of XML stream from service to host:

```
<stream:stream
    xmlns='jabber:component:accept'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='host'>
```

XML stream sent in reply from host to service:

```
<stream:stream
    xmlns='jabber:component:accept'
    xmlns:stream='http://etherx.jabber.org/streams'
    from='host'
    id='someid'>
```

(Note: the XML stream returned from the host to the service contains an 'id' attribute. This ID functions as a session key for the service's connection to the host.)

The next step is for the service to provide authentication credentials to the host. These credentials are sent in a <handshake/ > element. The information contained in the handshake element is a concatenation of the stream id and a secret known by the host and the service, encrypted according to the SHA1 Secure Hash Algorithm [14] and represented as all lowercase hex.

Handshake sent from service to host:

```
<handshake>aaee83c26aeeafcbabeabfcbcd50df997e0a2a1e</handshake>
```

If the host determines that the service's authentication credentials are valid, it will return an empty <handshake/> element to the service.

Handshake validation sent from host to service:

```
<handshake/>
```

If the host determines that the service's authentication credentials are not valid, it will return a stream error to the service and close the stream.

Host sends stream error to service:

```
<stream:error>Invalid handshake</stream:error>
</stream:stream>
```

8.3.2 Authentication of a Host by a Service

When a host initiates a connection to a service, the service will want to verify the identity of the host so that it knows whether the host can be trusted (e.g., if a service accepts connections from multiple hosts). The first step is to negotiate the XML streams between the service and the host, scoped within the "jabber:component:connect" namespace.

Initiation of XML stream from host to service:

```
<stream:stream
    xmlns='jabber:component:connect'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='service'>
```

XML stream sent in reply from service to host:

```
<stream:stream
    xmlns='jabber:component:connect'
    xmlns:stream='http://etherx.jabber.org/streams'
    from='service'
    id='someid'>
```

(Note: the XML stream returned from the service to the host contains an 'id' attribute. This ID functions as a session key for the host's

connection to the service.)

The next step is for the host to provide authentication credentials to the service. These credentials are sent in a <handshake/> element. The information contained in the handshake element is a concatenation of the stream id and a secret known by the host and the service, encrypted according to the SHA1 Secure Hash Algorithm [14] and represented as all lowercase hex.

Handshake sent from host to service:

```
<handshake>aaee83c26aeeafcbabeabfcbcd50df997e0a2a1e</handshake>
```

If the service determines that the host's authentication credentials are valid, it will return an empty <handshake/> element to the host.

Handshake validation sent from host to service:

```
<handshake/>
```

If the service determines that the host's authentication credentials are not valid, it will return a stream error to the host and close the stream.

Host sends stream error to service:

```
<stream:error>Invalid handshake</stream:error>
</stream:stream>
```

9. Routing, Delivery, and Presence Guidelines

9.1 Routing and Delivery of XML Chunks

XML chunks that are not handled directly by a host (e.g., for the purpose of data storage) are routed or delivered to the intended recipient of the chunk as represented by a Jabber Identifier in the 'to' attribute. The following rules apply:

- If the Jabber Identifier contains a resource identifier (to="node@host/resource"), the chunk is delivered first to the resource that exactly matches the resource identifier, or secondarily to a resource that matches partially (e.g., resource "foo" partially matches resource identifier "foobar").

- If the Jabber Identifier contains a resource identifier and there are no matching resources, but there are other connected resources associated with the node, then message chunks are further processed as if no resource is specified (see next item). For all other chunks, the host should return them to the sender with a type of "error" and an appropriate error code (503) and message.

- If the Jabber Identifier contains only a node@host and there is at least one connected resource available for the node, the host should deliver the chunk to an appropriate resource based on the availability state, priority, and connect time of the connected resource(s).

- If the Jabber Identifier contains only a node@host and there are no connected resources available for the node (e.g., an IM user is offline), the host may choose to store the chunk (usually only message and presence subscription chunks) on behalf of the node and deliver the chunk when a resource becomes available for that node.

9.2 Availability Tracking

A host is responsible for keeping track of who has been notified of availability for a resource, and for ensuring that all of those entities are notified when the resource becomes unavailable.

9.3 Presence Probe

Hosts may discover the presence of remote entities on behalf of a connected node by sending a presence chunk of type "probe". The remote host is responsible for responding to the presence probe only when (1) the probing entity has been allowed to access the probed

entity's presence (e.g., by server rules or user subscriptions) and (2) the probed entity is available. The probing entity's host then informs the probing entity of the probed entity's last known available presence (for all of the probed entity's resources if applicable).

9.4 Presence Broadcast

When a node first becomes available, the host sends presence probes to any remote entities that are subscribed to that node's presence. The host then sends the node's initial presence chunk and any future presence changes to any subscribed entities.

9.5 Supported Namespaces

If an entity receives an iq chunk in a namespace it does not understand, the entity should return an iq chunk of type "error" with an appropriate error element (code 400, bad request). If an entity receives a message chunk without a body and a namespace it does not understand, it must ignore that chunk. If an entity receives a message or presence chunk that contains XML data in an extended namespace it does not understand, the portion of the chunk that is in the unknown namespace should be ignored.

10. Security Considerations

10.1 SSL

Hosts can additionally support normal SSL [17] connections for added security on port 5223 for node-to-host communications and 5270 for host-to-host communications.

10.2 Secure Identity and Encryption

Nodes may optionally support signing and encrypting messages and presence by using the Public Key Infrastructure (e.g., PGP/GnuPG), with the encrypted or signed data sent in an <x/> element in the jabber:x:encrypted or jabber:x:signed namespace. (These are draft protocols and are not covered in this document.)

The Jabber model specifically does not require trust in the remote hosts. Although there may be benefits to a "trusted host" model, direct node-to-node trust is already in use in the SMTP protocol and allows those who desire a higher level of security to use it without requiring the significant increase in complexity throughout the architecture required to implement a trusted host model.

10.3 Node Connections

The IP address and method of access of nodes are never made available, nor are any connections other than the original host connection required. This protects the node's host from direct attack or identification by third parties via a gateway.

10.4 Presence Information

Presence subscriptions are enforced by the node's host. Only the approved entities are able to discover a node's availability.

10.5 Host-to-Host Communications

There is no necessity for any given Jabber host to communicate with other Jabber hosts, and host-to-host communications may be disabled by the administrator of any given Jabber deployment. This is especially valuable in non-public environments such as a company intranet.

For additional host-to-host security measures such as prevention of spoofing, see Section 8.2.

11. Multi-User Chat

In addition to one-to-one conversations between two people or entities, Jabber also enables multi-user chat environments similar to those of Internet Relay Chat. In Jabber these environments are variously called chat rooms or conference rooms, and utilize a special message type of "groupchat". Each room is identified as a node@host, specifically as room-name@conference-service, where "conference-service" is the hostname at which the conference service is running. Each participant in a room is identified as a node@host/resource, specifically room-name@conference-service/nickname, where "nickname" is the nickname of the participant (which may or may not be the participant's actual username).

Because multi-user chat is not usually considered part of the core functionality provided by an IM system, we have decided to describe it separately from the main body of this document, even though all XML data sent in order to provide multi-user chat functionality is fully compliant with the base protocols. The following descriptions are divided by "use case" to highlight how the Jabber protocols have been used to provide limited multi-user chat functionality.

11.1 Entering a Room

An IM user or other Jabber Entity becomes a participant in a room by entering the room. The user does this by sending presence to the room.

User enters room:

```
JABBER USER SENDS:
  <presence
      to='room-name@conference-service/nickname'/>

JABBER USER RECEIVES:
  <presence
      from='room-name@conference-service/nickname'
      to='node@host/resource'/>
```

11.2 Sending a Message to All Participants

A participant can send a message to all other participants in the room by sending a message of type "groupchat" to the room itself. The conference service is then responsible for reflecting that message out to all the participants with a type of "groupchat".

Participant sends message to all participants:

```
PARTICIPANT SENDS:
  <message
      to='room-name@conference-service'>
    <body>Hello world</body>
  </message>

EACH PARTICIPANT RECEIVES:
  <message
      from='room-name@conference-service/nickname'
      to='node@host/resource'
      type='groupchat'>
    <body>Hello room!</body>
  </message>
```

11.3 Sending a Message to A Selected Participant

A participant can send a message to a specific other participant in the room by sending a message of type other than "groupchat" to that participant.

Participant sends message to a selected participant:

```
PARTICIPANT SENDS:
  <message
      to='room-name@conference-service/nick2'
      type='chat'>
    <body>Hi</body>
  </message>

RECIPIENT RECEIVES:
  <message
      from='room-name@conference-service/nick1'
      to='node@host/resource'
      type='chat'>
    <body>Hi</body>
  </message>
```

11.4 Changing Nickname

A participant can change his or her nickname in a room by sending updated presence information to the room.

Participant sends nickname change:

```
PARTICIPANT SENDS:
<presence to='room-name@conference-service/newnick'/>

PARTICIPANT RECEIVES:
<presence
    from='room-name@conference-service/oldnick'
    to='node@host/resource'
    type='unavailable'/>

<presence
    from='room-name@conference-service/newnick'
    to='node@host/resource'/>
```

11.5 Exiting a Room

A participant exits a room by sending presence of type "unavailable" to the room.

User exits room:

```
PARTICIPANT SENDS:
<presence
      to='room-name@conference-service/nickname'
      type='unavailable'/>

PARTICIPANT RECEIVES:
  <presence
      from='room-name@conference-service/nickname'
      to='node@host/resource'
      type='unavailable'/>
```

12. IMPP and Interoperability Notes

12.1 Requirements Conformance

The Jabber protocols presented herein are in near conformance to RFC 2778 [18] and RFC 2779 [19]. Notable differences are:

- RFC 2779, section 2.5 - Complete conformance with these requirements can be obtained by using the public key infrastructure via applications such as PGP or GnuPG.

- RFC 2779, section 4.1, paragraph 10 - all MIME data is delivered via HTTP.

Note: the Jabber protocols have been in evolution for approximately four years as of the date of this memo, thus they have not been designed in response to RFCs 2778 and 2779.

12.2 Interoperability

Jabber provides interoperability with certain non-Jabber instant messaging networks, but at the cost of reverse engineering each non-Jabber instant messaging protocol and operating a host-based gateway to that protocol. The form of interoperability that Jabber offers also requires the Jabber user to have a valid account on each non-Jabber instant messaging network. It is recognized that this form of interoperability is sub-optimal, and the Jabber community looks forward to assisting in the development of standards-based interoperability.

13. Known Deficiencies

13.1 Further Definition of Transport Layer

The transport layer, currently implemented via XML streams, needs to be better defined and even further separated from the data to be transported. Ideally, any stateful, namespace-aware transport layer should be able to transport the common data types defined in the Jabber protocols.

Because Jabber was designed as a lightweight transport layer for routing instant messages, presence, and related information, quality of service (QoS) did not rank high in the priorities of its designers. In addition, features such as multi-hop routing and end-to-end store and forward of messages would be desirable in large-scale or mission-critical implementations of the Jabber protocols.

The Jabber protocols were built from the ground up to use XML, and the primary focus was on the exchange of small chunks of structured information. For this reason, the transport of binary payloads was not a priority and currently is supported only by sending them out of band (e.g., through HTTP PUTs to and GETs from a DAV server). Robust support for binary payloads would be desirable.

The current method of separating discrete semantic units from the stream in Jabber is elegant because all the necessary framing information is inherent in the XML; it makes framing entirely independent of the underlying transport layer. However, it has significant performance disadvantages, since it requires a Jabber Entity to parse the XML for the entire XML chunk in order to extract a subset of the information from it. A less resource-intensive framing mechanism may be desirable.

13.2 More Complete Namespace Support

At present the Jabber protocols comply only with a subset of the XML namespace specification and do not offer the full flexibility of XML namespaces. In addition it would be beneficial for the Jabber protocols to enable types of availability other than those defined for the <show/> element through a properly namespaced sub-element of the presence data type.

13.3 More Flexible Routing

Existing Jabber implementations contain some hardcoded rules (based on <priority/> and most recent connection time) for the routing of XML chunks to the resources associated with a node. A more flexible approach to routing would be desirable. In addition, full

conformance with RFC 2396 [7] would be valuable, perhaps by prepending the string "jabber:" to the Jabber Identifier, resulting in a URI of the form "jabber:node@host/resource".

13.4 More Robust Security

While the current Jabber protocols use Secure Sockets Layer to provide transport-level encryption and node-level encryption (PGP/GPG) for end-to-end message encryption, it would also be desirable to support network-wide authentication and trust based on the Public Key Infrastructure. This might be pursued through a Certificate Authority model, a Web of Trust model, or some combination of the two. In addition, XML encryption would be a valuable addition to the Jabber protocols.

13.5 Improved Subscriptions Model

The current specification overloads the presence element in order to provide a mechanism for subscription requests and responses. It is recognized that this solution is sub-optimal and a future protocol revision will address this deficiency by providing subscription functionality through the iq element and an appropriate namespace. Even further, a generic mechanism for publication and subscription (pub/sub) and the management of access control lists (ACLs) would be quite beneficial, perhaps on a model similar to the Presence and Availability Management [20] specfication.

14. Future Specifications and Submissions

Future specifications and submissions related to the Jabber protocols will most likely focus on a clear separation between the different protocol levels (e.g., routing, data transport, and messaging/ presence), as well as next-generation protocol enhancements that address the deficiencies described in the previous section.

References

[1] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, August 1982.

[2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997, <http://www.ietf.org/rfc/rfc2068.txt>.

[3] World Wide Web Consortium, "HyperText Markup Language", January 2000, <http://www.w3.org/TR/html/>.

[4] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C xml, October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>.

[5] XML-RPC.com, "XML-RPC", May 2001, <http://www.xmlrpc.com/spec>.

[6] World Wide Web Consortium, "SOAP", May 2000, <http://www.w3.org/TR/SOAP/>.

[7] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998, <http://www.ietf.org/rfc/rfc2396.txt>.

[8] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[9] Jeremie Miller, et al., "The jabberd Project", January 1998, <http://jabberd.jabberstudio.org/>.

[10] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[11] University of Southern California, "Transmission Control Protocol", RFC 793, September 1981, <http://www.ietf.org/rfc/rfc0793.txt>.

[12] Internet Assigned Numbers Authority, "Internet Assigned Numbers Authority", January 1998, <http://www.iana.org/>.

[13] World Wide Web Consortium, "Namespaces in XML", W3C xml-names, January 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>.

[14] World Wide Web Consortium, "Secure Hash Algorithm - Version 1.0", October 1997, <http://www.w3.org/PICS/DSig/SHA1_1_0.html>.

[15] International Organization for Standardization, "Data elements and interchange formats - Information interchange - Representation of dates and times", ISO Standard 8601, June 1988.

[16] Gulbrandsen, A. and P. Vixie, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2052, October 1996.

[17] Freier, A., Karlton, P. and P. Kocher, "The SSL Protocol - Version 3.0", November 1996, <http://home.netscape.com/eng/ssl3/draft302.txt>.

[18] Day, M., Rosenberg, J. and H. Sugano, "A Model for Presence and Instant Messaging", RFC 2778, February 2000, <http://www.ietf.org/rfc/rfc2778.txt>.

[19] Day, M., Aggarwal, S., Mohr, G. and J. Vincent, "A Model for Presence and Instant Messaging", RFC 2779, February 2000, <http://www.ietf.org/rfc/rfc2779.txt>.

[20] PAM Forum, "Presence and Availability Management", September 2001, <http://www.pamforum.org/>.

Authors' Addresses

Jeremie Miller
Jabber Software Foundation
1899 Wynkoop Street, Suite 600
Denver, CO 80202
US

EMail: jeremie@jabber.org
URI: http://www.jabber.org/

Peter Saint-Andre
Jabber Software Foundation
1899 Wynkoop Street, Suite 600
Denver, CO 80202
US

EMail: stpeter@jabber.org
URI: http://www.jabber.org/

James Barry
Jabber, Inc.
1899 Wynkoop Street, Suite 600
Denver, CO 80202
US

EMail: jmbarry@jabber.com
URI: http://www.jabber.com/

Appendix A. The <error/> element

A standard error element is used for failed processing of XML chunks. This element is a child of the failed element.

A.1 Attributes

- code - a numerical error code corresponding to a specific error description. The numerical codes used in Jabber are nearly synchronous with HTTP error codes:

  - 302 (Redirect) - Whereas the HTTP spec contains eight different codes for redirection, Jabber contains only one (which is intended to stand for any redirection error). However, Jabber code 302 is being reserved for future functionality and is not implemented at this time.

  - 400 (Bad Request) - Jabber code 400 is used to inform a sender that a request could not be understood by the recipient because it cannot be understood. This might be generated when, for example, a Jabber Entity sends a message that does not have a 'to' attribute, or when a node attempts to authenticate without sending a username.

  - 401 (Unauthorized) - Jabber code 401 is used to inform Jabber nodes that they have provided incorrect authorization information, e.g., an incorrect password or unknown username when attempting to authenticate with a Jabber host.

  - 402 (Payment Required) - Jabber code 402 is being reserved for future use and is not in use at this time.

  - 403 (Forbidden) - Jabber code 403 is used to inform a Jabber Entity that the its request was understood but that the recipient is refusing to fulfill it, e.g., if a node attempts to set information (e.g., preferences or profile information) associated with another node.

  - 404 (Not Found) - Jabber code 404 is used to inform a sender that no recipient was found matching the Jabber Identifier to which an XML chunk was sent, e.g., if a sender has attempted to send a message to a Jabber Identifier that does not exist. (Note: if the host of the intended recipient cannot be reached, an error code from the 500 series will be sent).

  - 405 (Not Allowed) - Jabber code 405 is used when the action requested is not allowed for the Jabber Identifier identified by the 'from' address, e.g., if a node attempts to set the time or version of a Jabber host.

- 406 (Not Acceptable) - Jabber code 406 is used when an XML chunk is for some reason not acceptable to a host or other Jabber Entity. This might be generated when, for example, a node attempts to register with a host using an empty password.

- 407 (Registration Required) - Jabber code 407 is used when a message or request is sent to a service that requires prior registration, e.g., if a node attempts to send a message through a gateway to a non-Jabber instant messaging system without having first registered with that gateway.

- 408 (Request Timeout) - Jabber code 408 is returned when a recipient does not produce a response within the time that the sender was prepared to wait.

- 500 (Internal Server Error) - Jabber code 500 is used when a Jabber host or service encounters an unexpected condition which prevents it from handling an XML chunk from a sender, e.g., if an authentication request is not handled by a host because the password could not be retrieved or if password storage fails when a node attempts to register with a host.

- 501 (Not Implemented) - Jabber code 501 is used when the recipient does not support the functionality being requested by a sender, e.g., if a node sends an authentication request that does not contain the elements defined by at least one of the accepted authentication methods or when a node attempts to register with a host that does not allow registration.

- 502 (Remote Server Error) - Jabber code 502 is used when delivery of an XML chunk fails because of an inability to reach the intended remote host or service. Specific examples of why this code is generated include a failure to connect to the remote host or resolve its hostname.

- 503 (Service Unavailable) - Jabber code 503 is used when a sender requests a service that a recipient is currently unable to handle, usually for temporary reasons, e.g., if a sender attempts to send a message to a recipient that is offline but the recipient's host is not running an offline message storage service.

- 504 (Remote Server Timeout) - Jabber code 504 is used when attempts to contact a remote host timeout, e.g., if an incorrect hostname is specified.

A.2 Examples

Message error:

```
<message
    to="juliet@montague.net"
    from="romeo@montague.net"
    type="error">
  <body>Sleep dwell upon thine eyes</body>
  <error code="404">Not Found</error>
</message>
```

IQ Error:

```
<iq
    type="error"
    from="service.shakespeare"
    to="juliet@capulet.com"
    id="i_002">
  <query xmlns="jabber:iq:register">
    <name>juliet</name>
    <email>juliet@somehost</email>
    <password>r0m30</password>
  </query>
  <error code="502">Remote Server Error</error>
</iq>
```

Appendix B. Acknowledgments

Thanks are due to all members of the Jabber Software Foundation. The following individuals have provided especially valuable assistance with the development of the Jabber protocols and/or comments on this document:

- John Hager
- Michael Lin
- Peter Millard
- Julian Missig
- Thomas Muldowney
- Iain Shigeoka
- Dave Smith
- Daniel Veillard
- David Waite

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.