Changes to UCR 2008, Change 2, made by Change 3 for Section 5.7, Near-Real-Time, Text-Based Messaging Products

SECTION	CORRECTION	EFFECTIVE DATE
5.7.3.3	Defined XMPP System Under Test (SUT) as an XMPP Server and Client.	Immediate
5.7.3.14.2.2; item number 2	Removed the requirement for the server to provide a temporary or permanent redirect address	Immediate
5.7.3.12.5; item number 1	Removed the requirement to change/delete "the handle."	Immediate
5.7.3.1.4	Removed the requirement mandating that clients reconnect after an unpredictable number of seconds between 0 and 60.	Immediate
5.7.3.17	Removed the requirement for XEP-0191: *Simple Communications Blocking*	Immediate
5.7.3.17	Removed the requirement for XEP-0138: *Stream Compression*	Immediate
5.7.3.17	Defined support for the following capabilities as Conditional requirements for XMPP Gateways: - XEP-0045: *Multi-User Chat* - XEP-0004: *Data Forms* - XEP-0077: *In-Band Registration* - XEP-0082: *XMPP Date and Time Profiles* - XEP-0068: *Field Standardization for Data Forms*	Immediate
Multiple	The active revision of RFC 3920 and 3921 were replaced with the published RFCs 6120, 6121, and 6122.	Immediate

TABLE OF CONTENTS

SECTION **PAGE**

LIST OF FIGURES

LIST OF TABLES

5.7 NEAR-REAL-TIME, TEXT-BASED MESSAGING PRODUCTS

5.7.1 Introduction

This section of the UCR defines functional requirements for Extensible Messaging and Presence Protocol (XMPP) clients and servers. These products fall within the data products category as shown in UCR, Figure 4.5.1-1, Overview of UC Product Categories within the DoD UC APL. The principal objective for this section is to address the essential capabilities needed to enable the following services:

- Exchange of presence
- One-to-one chat
- Multi-user chat

5.7.2 Overview

This section of the UCR addresses essential capabilities and features that enable the near-real-time exchange of relatively brief text-based messages in support of applications such as presence, one-to-one chat, and multi-user chat. The term "near-real-time" underscores the point that XMPP applications and services are generally used to enable the immediate interchange of information. The term "text-based" refers to the exchange of relatively brief text messages with particular contacts or services. The terms "messaging" or "instant messaging" are umbrella terms, which can refer to a wide variety of text-based applications, including, but not limited to the following:

- Sending messages in the context of a two-party text conversation (i.e., a one-to-one chat session)

- Sending messages in the context of a multiuser chat (i.e., text-based conferencing, also known as group chat)

- Sending messages in the context of a notification service (including content syndication, alerts, notifications, and other similar applications)

- Sending messages in the context of a structured request-response interaction (e.g., one entity requests information and another entity responds with the result)

- Sending messages to convey that an error occurred in relation to a previously sent message

5.7.3 XMPP Requirements

5.7.3.1 Introduction

In accordance with Joint Staff and DoD IT Standards Registry (DISR) mandates, this specification stipulates the use of the XMPP. The XMPP is an open, XML-based protocol specifically designed to enable the near-real-time exchange of text-based communication including applications such as presence, one-to-one chat, and multi-user chat. The XMPP is proven (i.e., has been widely deployed and rigorously tested), secure (i.e., offers inherent support for channel encryption and strong authentication), and highly scalable.

5.7.3.2 Scope and Acknowledgement

The principal intent for this section of the UCR is to address required functionality to enable:

- Multivendor interoperability
- Essential Information Assurance requirements

Additionally, a key objective for this section of the UCR is to create a well-defined and unambiguous set of requirements that vendors can "build to" and which will facilitate compliance and certification testing.

This section of the UCR defines an XMPP specification that is based upon commercial standards. This specification assumes that the reader is familiar with the general concepts and requirements defined in RFC 6120 and RFC 6121 (i.e., the XMPP baseline standards). For that reason, this specification does not attempt to cover all aspects exhaustively or all normative requirements addressed in these baseline documents. Concerning RFC 6120 and RFC 6121, compliant solutions are expected to implement all requirements defined as "MUST," "SHALL," "REQUIRED," "MUST NOT," and "SHALL NOT." It is also expected that vendors will likewise implement requirements defined as "SHOULD" or "SHOULD NOT" except where there may exist valid reasons in particular circumstances to ignore a particular requirement. To better enable multivendor interoperability and to address specific Information Assurance) requirements, some of the content defined as "SHOULD," "RECOMMENDED," "SHOULD NOT," "NOT RECOMMENDED," "MAY," or "OPTIONAL" in RFC 6120 and RFC 6121 has been redefined by this specification to reflect requirement levels associated with the following terminology: "MUST," "SHALL," "REQUIRED," "MUST NOT," or "SHALL NOT." In the event of a discrepancy between the commercial XMPP standards and this section of the UCR, the explicit requirements defined in this section of the UCR take precedence. A significant portion of the text of this specification was borrowed or derived from RFC 6120 and RFC 6121. For the sake of traceability, individual requirements are linked to a reference source by a bracketed section number and associated reference source identifier.

In addition to the core functionality specified in RFC 6120 and RFC 6121, this section of the UCR also defines a minimum XMPP feature set which will incorporate requirements from XMPP Extension Protocol (XEP) series documents plus a few additional IETF RFCs. For further detail, see Section 5.7.3.17, XMPP Extensions.

5.7.3.3 XMPP Solution Framework

The XMPP is implemented using a client-server design. Commonly, the XMPP network consists of a number of interconnected servers. Each server operates as the "home" server for some number of locally connected clients (see Figure 5.7.3-1, XMPP Requirements).

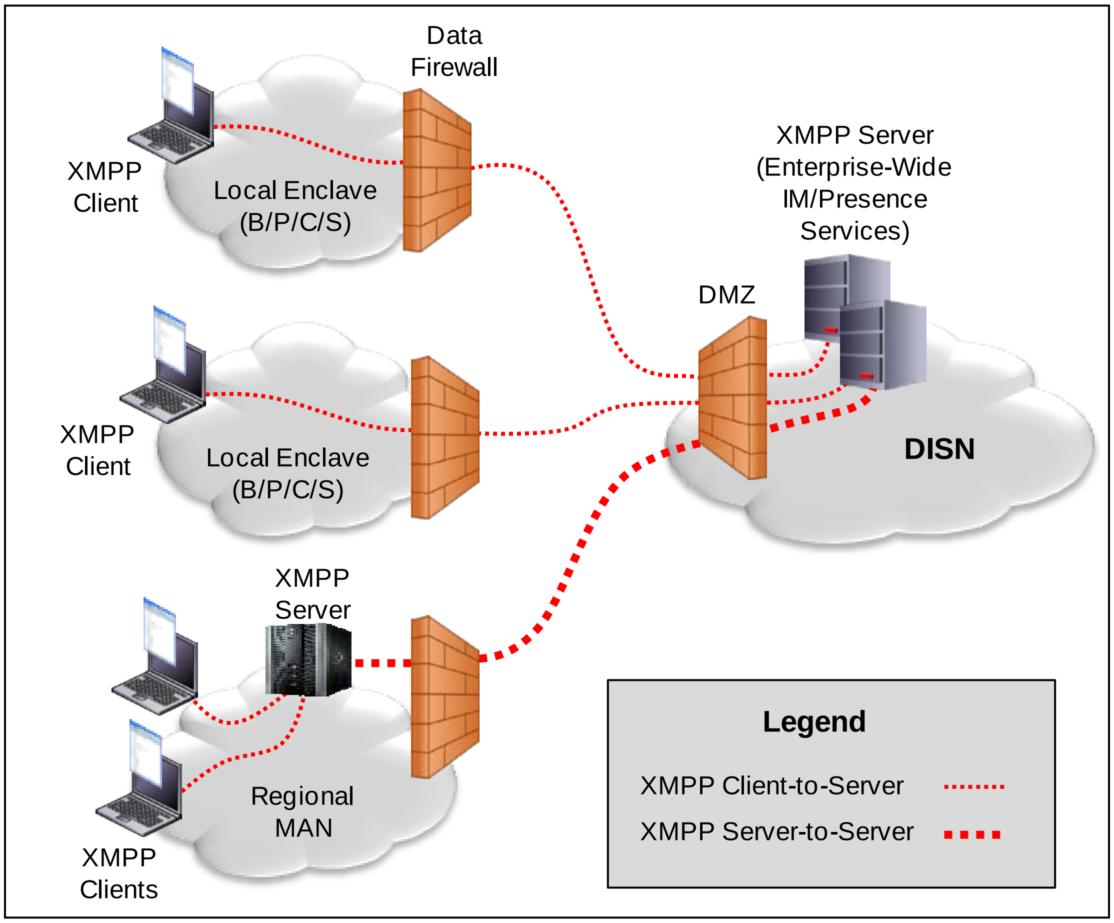

Figure 5.7.3-1. High-Level XMPP Solution Framework

- An **XMPP client** must connect to its "home" server in order to be granted access to the network and subsequently to be permitted to exchange instant messaging (IM) and presence information with other users/services. After the client successfully negotiates and establishes a connection with its home server, the client then uses XMPP to communicate with its server, other clients, and any other entities (e.g., a multiuser chat service) on the network. More than one client can connect concurrently to the same home server on behalf of the same local or user account. [Section 2.5, RFC 6120]

- An **XMPP server** manages XML streams with locally hosted clients and delivers XML stanzas to those clients over the negotiated streams. The server also manages XML streams with peer servers and routes XML stanzas to those servers over the negotiated streams. A server is responsible for the enforcement of security policies (e.g., user authentication and channel encryption), storing a user's roster, and maintaining presence information for all of its hosted users. A server may also host local services that use XMPP communication primitives (e.g., multiuser chat service). [Section 2.5, RFC 6120]

For APL certification purposes, an XMPP System Under Test (SUT) shall consist of both an XMPP server and XMPP client. The one exception is XMPP gateway implementations (see the note below for further clarification).

NOTE: Proprietary client-to-server protocols are permitted within the context of a MILDEP enclave. However, these proprietary implementations must be able to federate with native XMPP servers by means of an XMPP server-to-server stream enabled through the use of an XMPP gateway implementation. Likewise, an XMPP gateway must be able to federate with other XMPP gateways by means of an XMPP server-to-server stream. The XMPP gateway implementations are expected to comply with server-to-server stream-related requirements as defined in Section 5.7. From an applications perspective, XMPP gateways must support Presence and one-to-one chat. However, it is understood that the majority of XMPP Gateway implementations will not be compliant with the following XMPP Extension: XEP-0045: Multi-User Chat.

5.7.3.4 Terminology

- XML Stanza. An XML stanza is a discrete XML fragment that is sent over the transport provided by the negotiated XML stream. As defined in the XMPP baseline specification, an XML stanza is "the basic unit of meaning in XMPP." [Section 4.1, RFC 6120]

- Initiating Entity and Receiving Entity. When a client initiates a session with its home server, the client is designated as the "initiating entity" and the server is labeled the "receiving entity." Likewise, when a server initiates a session with a peer server, the server originating the connection is designated as the initiating entity and the targeted peer server is labeled as the receiving entity. [Section 1.4, RFC 6120]

- XML Stream. An XML stream provides the essential transport needed for all client-to-server and server-to-server communications. An XML stream acts as a logical envelope (i.e., container) for all the XML elements and XML stanzas exchanged between a client and server or between server peers. As discussed in RFC 6121, Section 4.3, an XML stream is always unidirectional, which means that XML stanzas can be sent in only one direction over the stream (either from the initiating entity to the receiving entity or from the receiving entity to the initiating entity). To enable communication between an initiating entity (i.e., a client or server) and a receiving entity (i.e., a server), the initiating entity will negotiate an XML stream to the receiving entity (the Initial Stream), and, in response, the receiving entity will negotiate an XML stream to the initiating entity (the Response Stream). [Section 4.1, RFC 6120]

- Contact. A contact is an entity that has a subscription to a user's presence or to which a user has a presence subscription. In this specification, the term "contact" is also used in a less strict sense to refer to a potential contact, an item in a user's roster, or the target of a particular message stanza or presence subscription request. [Section 3, RFC 6121]

- Entity. In the context of this specification, an entity typically refers to a client or server implementation. However, in XMPP, an entity also could be a reference to a gateway, a service, or a chat room.

- Originating Entity. The entity (e.g., a client or server) that generates a stanza is referred to as the originating entity.

- Mandatory-to-Negotiate Stream Features. Mandatory-to-negotiate stream features refer to a set of particular protocol interactions that are mandatory for the initiating entity to complete before the receiving entity will accept XML stanzas from the initiating entity (e.g., authentication and channel encryption). [Section 4.2.1, RFC 6120]

- Connected Resource. After successfully binding a resource to the XML stream, the client is referred to as a Connected Resource.

- Available Resource. After a connected resource sends initial presence, it is referred to as an Available Resource.

- Interested Resource. If a connected resource or available resource requests the roster, it is referred to as an Interested Resource.

- User. The term "user" commonly refers to the owner of an XMPP account. It is worth noting that a user may not necessarily be a natural person (e.g., it could be an automated process).

- Related Abbreviations:
  - C = client
  - CC = contact's client
  - CS = contact's server
  - I = an initiating entity
  - R = a receiving entity
  - S = server
  - UC = user's client
  - US = user's server

5.7.3.5 Functional Summary

5.7.3.5.1 Client-to-Server Connections

As discussed previously, a client needs to connect to a server in order to be granted access to the network. The process used by a client to open, secure, and close an XML stream is as follows [Section 1.3, RFC 6120]:

1. Determine the hostname and port at which to connect.

2. Open a Transmission Control Protocol (TCP) connection.

3. Open an XML stream over TCP.

4. Negotiate Transport Layer Security (TLS) for channel encryption.

5. Authenticate using a Simple Authentication and Security Layer (SASL) mechanism.

6. Bind a resource to the stream (see Section 5.7.3.10, Resource Binding).

7. Exchange an unbounded number of XML stanzas with other entities on the network.

8. Close the XML stream.

9. Close the TCP connection.

5.7.3.5.2 Server-to-Server Connections

For server-to-server communications (also known as "federation"), an XMPP server must establish an XML stream with a peer server. This type of connection is also commonly abbreviated as (s2s). The process for establishing and terminating server-to-server connections is as follows [Section 1.3, RFC 6120]:

1. Determine the hostname and port at which to connect.

2. Open a TCP connection.

3. Open an XML stream over TCP.

4. Negotiate TLS for channel encryption.

5. Authenticate using a SASL mechanism.

6. Exchange an unbounded number of XML stanzas both directly for the servers and indirectly on behalf of entities associated with each server (e.g., connected clients).

7. Close the XML stream.

8. Close the TCP connection.

5.7.3.6 XMPP Addressing

All the basic elements (i.e., XMPP clients, servers, and associated services) of XMPP are addressable using a globally unique address. Generally, XMPP addresses are referred to as Jabber IDs or JIDs. Typically, a JID is made up of three parts within the following structure: [localpart@domainpart/resourcepart].

- Domainpart. The domainpart of a JID is that portion after the "@" character (if any) and before the "/" character (if any); it is the primary identifier and is the only required element of a JID (a mere domainpart is a valid JID). Typically, a domainpart identifies the "home" server to which clients connect for XML routing and data management functionality. However, it is not necessary for an XMPP domainpart to identify an entity that provides core XMPP server functionality (e.g., a domainpart can identify an entity such as a multiuser chat service or a user directory). [Section 2.2, RFC 6122]

- Localpart. The localpart of a JID is an optional identifier placed before the domainpart and separated from the latter by the "@" character. Typically, a localpart uniquely identifies the entity requesting and using network access provided by a server (i.e., a local account). However, the localpart of a JID can also represent other kinds of entities (e.g., a chat room associated with a multiuser chat service). The entity represented by an XMPP localpart is addressed within the context of a specific domain. [Section 2.3, RFC 6122]

- Resourcepart. The resourcepart of a JID is an optional identifier placed after the domainpart and separated from the latter by the "/" character. A resourcepart can modify either a <localpart@domainpart> address or a mere <domainpart> address. Typically a resourcepart uniquely identifies a specific connection (e.g., a device or location) or object (e.g., an occupant in a multiuser chat room) belonging to the entity associated with an XMPP localpart at a local domain. [Section 2.4, RFC 6122]

An address of the form [localpart@domainpart] is referred to as a bare JID. An address of the form [localpart@domainpart/resourcepart] is referred to as a full JID. Table 5.7.3-1, XMPP Addressing Examples, provides a few examples.

Table 5.7.3-1. XMPP Addressing Examples

XMPP ENTITY	FORMAT	EXAMPLE
Server	Consisting of a single domainpart identifier.	"chat.dod.mil"
User Account	Consisting of a localpart and domainpart separated by the "@" character.	"john.smith@chat.dod.mil"
Specific Client Connection	Consisting of a localpart, domainpart and resourcepart, where the localpart is separated from the domainpart by the "@" character and the domainpart is separated from the resourcepart by the "/" character.	"john.smith@chat.dod.mil/XMPP Desktop Client"

5.7.3.7 XML Streams

As mentioned previously, an XML stream provides the fundamental transport needed for all client-to-server and server-to-server communications. The ability to establish and maintain an XML stream is an essential capability of XMPP.

5.7.3.7.1 TCP Binding

[Required] As XMPP is defined in this specification, an initiating entity SHALL open a TCP connection to the receiving entity before it negotiates XML streams with the receiving entity. The parties then maintain that TCP connection for as long as the XML streams are in use. [Section 3.1, RFC 6120]

5.7.3.7.1.1 Hostname Resolution

Because XML streams are sent over TCP, the initiating entity needs to determine the IPv4 or IPv6 address (and port) of the receiving entity's "origin domain" before it can attempt to connect to the XMPP network. [Section 3.2, RFC 6120]

1. **[Required]** When a server receives a stanza and the JID contained in the "to" attribute does not match one of the configured hostnames of the server itself, the server SHALL attempt to route the stanza to the remote domain. If no server-to-server stream exists between the two domains, the sender's server SHALL attempt to resolve the remote hostname using a DNS service location record service (DNS SRV record) of "xmpp-server" (for server-to-server connections). [Sections 10.4 of RFC 6120],

2. **[Required]** To discover the hostname of the XMPP service in a given domain, XMPP clients SHALL use the same hostname resolution process. However, the DNS service location record service identified in the DNS SRV query will be "xmpp-client" (for client-to-server connections).

   NOTE: It is not necessary to resolve the DNS domain name before each connection attempt, because DNS resolution results can be cached temporarily in accordance with time-to-live values. [Section 13.9.2, RFC 6120]

3. **[Required]** All server and client implementations SHALL support this hostname resolution process as follows [Section 3.2.1, RFC 6120]:

   a. The initiating entity SHALL construct a DNS SRV query (see RFC 2782) where inputs are:

      (1) A service of "xmpp-server" for server-to-server connections (or alternatively, "xmpp-client" for client-to-server connections)

      (2) A proto of "tcp"

      (3) A name corresponding to the "origin domain" of the XMPP service to which the initiating entity wishes to connect (e.g., "example.disn.mil")

   b. The result is a query such as "_xmpp-server._tcp.example.disn.mil." (or alternatively, "_xmpp-client._tcp.exmple.disn.mil." for client-to-server connections).

   c. If a response is received, it will contain one or more combinations of a port and hostname, each of which is weighted and prioritized as described in RFC 2782.

d. The initiating entity SHALL choose one of the returned hostnames to resolve (following the rules in RFC 2782), which it SHALL do by using a DNS “A” or “AAAA” lookup on the hostname; this will result in an IPv4 or IPv6 address.

e. The initiating entity SHALL use the IP address from the first successfully resolved hostname (with the corresponding port number returned by the SRV lookup) as the connection address for the receiving entity.

f. If the initiating entity fails to connect using that IP address, but the “A” or “AAAA” lookup returned more than one IP address, then the initiating entity SHALL use the next resolved IP address for that hostname as the connection address.

g. If the initiating entity fails to connect using all resolved IP addresses for a given hostname, then it repeats the process of resolution and connection for the next hostname returned by the SRV lookup.

h. If the initiating entity fails to connect using any hostname returned by the SRV lookup, then it either SHALL abort the connection attempt or SHALL use the fallback process described in the following section.

5.7.3.7.1.2 Standard, Default Port Values

The standard default XMPP port for client-to-server connections is 5222. The standard default XMPP port for server-to-server connections is 5269.

5.7.3.7.1.3 Fallback Process

[Required] The fallback process SHALL be a normal “A” or “AAAA” address record resolution to determine the IPv4 or IPv6 address of the origin domain, where the port used is the “xmpp-client” port of 5222 for client-to-server connections or the “xmpp-server” port 5269 for server-to-server connections. [Section 3.2.2, RFC 6120]

NOTE: If the initiating entity has been explicitly configured to associate a particular hostname (and potentially a port value) with the origin domain of the receiving entity, the initiating entity SHOULD use the configured name instead of performing the DNS SRV resolution process on the origin name. Naturally, if the initiating entity has knowledge (e.g., through the configuration process) of the IP address and port of the receiving entity, then there is no reason to perform hostname resolution. [Section 3.2.3 RFC 6120]

5.7.3.7.2 Stream Negotiation Overview

To establish an XML stream, the initiating entity (e.g., client or server) and the receiving entity (e.g., a server) shall agree on a set of preconditions for connecting as a client or as a peer server. The entities involved will begin the process of stream negotiation. In this process, the receiving entity for a stream will impose certain conditions upon the connection. For example, when a client attempts to establish an XML stream with its home server, it will first open a persistent TCP connection and then begin the process of stream negotiation. Through an exchange of XML elements with the client, the server will inform the client regarding what stream features it supports. The server will specify whether a particular stream feature is required or optional. As a result, the stream negotiation process permits the server to enforce important preconditions (e.g., user authentication and channel encryption) upon the connection. Stream negotiation is a multistage process. [Section 4 of RFC 6120]

5.7.3.7.3 Stream Features

1. **[Required]** The initiating entity SHALL initiate an XML stream by sending an initial stream header to the receiving entity.

```
C:  <stream:stream
        from='john@im.example1.dod.mil'
        to='im.example1.dod.mil'
        version='1.0'
        xml:lang='en'
        xmlns='jabber:client'
        xmlns:stream='http://etherx.jabber.org/streams'>
```

2. **[Required]** In response, the receiving entity SHALL send a response stream header to the initiating entity.

```
S:  <stream:stream
        from='im.example1.dod.mil'
        id='t7AMCin9zjMNwQKDnplntZPIDEI='
        to='john@im.example1.dod.mil'
        version='1.0'
        xml:lang='en'
        xmlns='jabber:client'
        xmlns:stream='http://etherx.jabber.org/streams'
```

3. **[Required]** After the receiving entity has sent a response stream header to the initiating entity, the receiving entity SHALL send a <features/> child element (prefixed by the streams namespace prefix) to the initiating entity in order to announce any conditions for

continuation of the stream negotiation process. Each condition takes the form of a child element of the <features/> element, qualified by a namespace that is different from the streams namespace and the content namespace. The <features/> element can contain one child, contain multiple children, or be empty. [Section 4.2.2, RFC 6120]

4. **[Required]** For stream features that are mandatory-to-negotiate, the definition of that feature SHALL declare that the feature is always mandatory-to-negotiate (e.g., this is true of resource binding for XMPP clients) or the receiving entity SHALL explicitly flag the feature as mandatory-to-negotiate (e.g., this is done for TLS by including an empty <required/> element in the advertisement for the STARTTLS feature). [Section 4.2.2, RFC 6120]

```
R:   <stream:features>
        <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
              <required/>
        </starttls>
     </stream:features>
```

5. **[Required]** If the <features/> element contains at least one mandatory feature, then the initiating entity SHALL continue with the stream negotiation process. An empty <features/> element indicates that the stream negotiation is complete and that the initiating entity is cleared to send XML stanzas. [Section 4.2.2, RFC 6120]

```
R:   <stream:features/>
```

NOTE: A <features/> element that contains only voluntary features indicates that the stream negotiation is complete and that the initiating entity is cleared to send XML stanzas. However, the initiating entity MAY negotiate further features if desired. [Section 4.2.2, RFC 6120]

5.7.3.7.4 Stream Restarts

1. **[Required]** On successful negotiation of a feature that necessitates a stream restart, both the initiating entity and the receiving entity SHALL consider the previous stream to be replaced, but SHALL NOT terminate the underlying TCP connection; instead, the initiating entity and the receiving entity SHALL reuse the existing connection. [Section 4.2.3, RFC 6120]

2. **[Required]** The initiating entity then SHALL send a new initial stream header to the receiving entity. [Section 4.2.3, RFC 6120]

3. **[Required]** When the receiving entity receives the new initial stream header, it SHALL generate a new stream ID (instead of reusing the old stream ID) and SHALL then send a new response stream header to the initiating entity. [Section 4.2.3, RFC 6120]

5.7.3.7.5 Continuation and Completion of Stream Negotiation

1. **[Required]** The receiving entity SHALL send an updated list of stream features to the initiating entity after a stream restart. [Section 4.2.4, RFC 6120]

   NOTE: The list of updated features MAY be empty if there are no further features to be advertised. [Section 4.2.4, RFC 6120]

2. **[Required]** The receiving entity SHALL indicate completion of the stream negotiation process by sending to the initiating entity either an empty <features/> element or a <features/> element that contains only voluntary features. Once stream negotiation is complete, the initiating entity is cleared to send XML stanzas over the stream for as long as the stream is maintained by both parties. [Section 4.2.5, RFC 6120]

   ```
   R:   <stream:features/>
   ```

   NOTE: A <features/> element that contains only voluntary features indicates that the stream negotiation is complete and that the initiating entity is cleared to send XML stanzas, but that the initiating entity MAY negotiate further features if desired. [Section 4.2.5, RFC 6120]

5.7.3.7.6 Directionality

An XML stream is always unidirectional, by which is meant that XML stanzas can be sent in only one direction over the stream (either from the initiating entity to the receiving entity or from the receiving entity to the initiating entity). [Section 4.3, RFC 6120]

1. **[Required]** For client-to-server sessions, a server SHALL allow a client to use “two streams over a single TCP connection.”

2. **[Required]** For server-to-server sessions, the two server peers SHALL use two streams over two TCP connections, where one TCP connection is used for the stream in which stanzas are sent from the initiating entity to the receiving entity and the other TCP connection is used for the stream in which stanzas are sent from the receiving entity to the initiating entity. [Section 4.3, RFC 6120]

   NOTE: This concept of directionality applies only to stanzas and explicitly does not apply to other first-level children of the stream root, such as elements used for TLS negotiation, SASL negotiation. In particular, during establishment of a server-to-server session, while completing STARTTLS negotiation and SASL negotiation, the two servers would use one TCP connection, but after the stream negotiation process is finished, that original TCP connection would be used only for the initiating server to send XML stanzas to the receiving server. In order for the receiving server to send XML stanzas to the initiating server, the receiving server would need to reverse the roles and negotiate an XML stream from the receiving server to the initiating server over a separate TCP connection. [Section 4.3, RFC 6120]

5.7.3.7.7 Closing a Stream

5.7.3.7.7.1 Closing a Stream without a Stream Error

1. **[Required]** Client and server implementations SHALL be capable of closing an XML stream by sending a closing </stream> tag. [Section 4.4, RFC 6120]

   S: </stream:stream>

   NOTE: The entity that sends the closing stream tag SHOULD behave as follows [Section 4.4, RFC 6120]:

   a. Wait for the other party to close also its stream before terminating the underlying TCP connection (this gives the other party an opportunity to finish transmitting any data in the opposite direction before the TCP connection is terminated).

   b. Refrain from initiating the sending of further data over that stream but continue to process data sent by the other entity (and, if necessary, react to such data).

   c. Consider both streams to be void if the other party does not send its closing stream tag within a configurable amount of time.

   d. After receiving a reciprocal closing stream tag from the other party or waiting a configurable amount of time with no response, the entity SHALL terminate the underlying TCP connection.

2. **[Required]** After the entity that sent the first closing stream tag receives a reciprocal closing stream tag from the other party, it SHALL terminate the underlying TCP connection or connections. [Section 4.4, RFC 6120]

5.7.3.7.8 Stream Attributes

5.7.3.7.8.1 Initial Streams

1. **[Required]** For client-to-server connections, it is assumed that the client knows the associated XMPP account name of the form <localpart@domain>. The client SHALL include the "from" attribute in the initial stream header it sends to the server and SHALL set the value to the associated XMPP account name of the form <localpart@domain>. [Section 4.6.1, RFC 6120]

2. **[Required]** For server-to-server connections, the initiating entity SHALL include the "from" attribute in the initial stream header it sends to the receiving entity and SHALL set its value to a hostname serviced by the initiating entity. [Section 4.6.1, RFC 6120]

3. **[Required]** For both client-to-server and server-to-server connections, the initiating entity SHALL include the "to" attribute in the initial stream header that it sends to the receiving entity and SHALL set its value to a hostname that the initiating entity knows or expects the receiving entity to service. [Section 4.6.2, RFC 6120]

   NOTE: For both client-to-server and server-to-server connections, the initiating entity SHOULD include an "xml:lang" attribute in the initial stream headers that it generates. [Section 4.6.4, RFC 6120]

4. **[Required]** For both client-to-server and server-to-server connections, the initiating entity SHALL include a "version" attribute whose value is "1.0" (or higher) in the initial stream headers it generates. [Section 4.6.5, RFC 6120]

   Example:

```
C:      <stream:stream
            from='john@im.example1.dod.mil'
            to='im.example1.dod.mil'
            version='1.0'
            xml:lang='en'
            xmlns='jabber:client'
            xmlns:stream='http://etherx.jabber.org/streams'>
```

5.7.3.7.8.2 Response Streams

1. **[Required]** For both client-to-server and server-to-server connections, the receiving entity SHALL include the "from" attribute in the response stream header that it sends to the

initiating entity and SHALL set its value to a hostname serviced by the receiving entity. [Section 4.6.1, RFC 6120]

2. **[Required]** For response stream headers in client-to-server communication, if the client included a “from” attribute in the initial stream header then the server SHALL include a “to” attribute in the response stream header and SHALL set its value to the bare JID specified in the “from” attribute of the initial stream header. If the client did not include a “from” attribute in the initial stream header then the server SHALL NOT include a “to” attribute in the response stream header. [Section 4.6.2, RFC 6120]

3. **[Required]** For server-to-server connections, the receiving entity SHALL include the “to” attribute in the response stream header that it sends to the initiating entity and SHALL set its value to the hostname specified in the “from” attribute of the initial stream header. [Section 4.6.2, RFC 6120]

4. **[Required]** For both client-to-server and server-to-server connections, the receiving entity SHALL include an “id” attribute in the response stream header that it sends to the initiating entity. The “id” attribute communicates a unique identifier for the stream, called a STREAM ID. The stream “id” shall have the property of randomness. [Section 4.6.3, RFC 6120]

5. **[Required]** For both client-to-server and server-to-server connections, the receiving entity SHALL include a “version” attribute where the value is 1.0 (or higher) in the response stream headers it sends to the initiating entity. [Section 4.6.5, RFC 6120]

Example:

```
S:     <stream:stream
           from='im.example1.dod.mil'
           id='t7AMCin9zjMNwQKDnplntZPIDEI='
           to='john@im.example1.dod.mil'
           version='1.0'
           xml:lang='en'
           xmlns='jabber:client'
           xmlns:stream='http://etherx.jabber.org/streams'
```

5.7.3.7.9 Namespaces

5.7.3.7.9.1 Streams Namespace

[Required] Client and server implementations SHALL qualify the root <stream/> element (“stream header”) by the namespace “http://etherx.jabber.org/streams” (the “streams namespace”). If this rule is violated, the entity that receives the offending stream header SHALL return a stream error to the sending entity, which SHALL be either <invalid-namespace/> or <bad-format/>. [Section 4.7.1, RFC 6120]

5.7.3.7.9.2 Content Namespace

1. **[Required]** An entity (client or server) SHALL declare a content namespace for data sent over the stream. The content namespace SHALL be the same for the initial stream and the response stream so that both streams are qualified consistently. The content namespace applies to all first-level child elements sent over the stream unless explicitly qualified by another namespace. [Section 4.7.2, RFC 6120]

2. **[Required]** The XMPP defines two content namespaces: “jabber:client” and “jabber:server.” Client implementations SHALL support the jabber:client content namespace. Server implementations SHALL support both the jabber:client content namespace (when the stream is used for communication between a client and a server) and the jabber:server content namespace (when the stream is used for communication between two servers). [Section 4.7.5, RFC 6120]

   Example:

```
C:    <stream:stream
          from='john@im.example1.dod.mil'
          to='im.example1.dod.mil'
          version='1.0'
          xml:lang='en'
          xmlns='jabber:client'
          xmlns:stream='http://etherx.jabber.org/streams'>
```

3. **[Required]** If an entity receives a first-level child element qualified by a content namespace it does not support, it SHALL return an <invalid-namespace/> stream error. [Section 4.7.5, RFC 6120]

5.7.3.7.10 Stream Errors

1. **[Required]** The error child SHALL be sent by an entity (client or server) if it perceives that a stream-level error has occurred. [Section 4.8, RFC 6120]

2. **[Required]** Stream-level errors are unrecoverable. Therefore, if an error occurs at the level of the stream, the entity (client or server) that detects the error SHALL send an <error/> element with an appropriate child element that specifies the error condition and at the same time send a closing </stream> tag. [Section 4.8.1.1, RFC 6120]

```
S:    <stream:error>
        <xml-not-well-formed
          xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
      </stream:error>
      </stream:stream>
```

3. **[Required]** The entity that generates the stream error then SHALL close the stream as explained under Section 4.4 of RFC 6120). [Section 4.8.1.1, RFC 6120]

```
C:    </stream:stream>
```

4. **[Required]** If the error is triggered by the initial stream header, the receiving entity SHALL still send the opening <stream> tag, include the <error/> element as a child of the stream element, and then send the closing </stream> tag (preferably all at the same time). [Section 4.8.1.2, RFC 6120]

5.7.3.7.10.1 Stream Error Syntax and Defined Stream Error Conditions

For guidance and associated requirements related to stream error syntax and defined stream error conditions, see Section 4.8, RFC 6120.

5.7.3.8 TLS and STARTTLS Negotiation

[Required] All XML streams (i.e., including both client-to-server and server-to-server connections) SHALL be secured with the use of the TLS protocol.

NOTE: On extremely low-bandwidth, high-latency connections, the use of TLS is not recommended.

5.7.3.8.1 STARTTLS Process

1. **[Required]** This specification mandates the use of the STARTTLS command to initiate TLS negotiation. All client and server implementations SHALL support and use the "STARTTLS" extension.

2. **[Required]** Immediately after the opening of the response stream, the receiving entity SHALL initiate the process of stream negotiation. [Section 5.4.1, RFC 6120]

3. **[Required]** In the stream feature announcement provided by the receiving entity during the initial stage of the stream negotiation process, the receiving entity SHALL advertize ONLY the STARTTLS feature (qualified by the XML namespace:

"urn:ietf:params:xml:ns:xmpp-tls") and SHALL also include an empty <required/> child element. [Section 5.4.1, RFC 6120] See the following example:

```
R:      <stream:features>
            <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
                <required/>
            </starttls>
        </stream:features>
```

5.7.3.8.2 Initiation of STARTTLS Negotiation

1. **[Required]** In order to begin the STARTTLS negotiation, the initiating entity SHALL issue the STARTTLS command (i.e., a <starttls/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace) to instruct the receiving entity that it wishes to begin a STARTTLS negotiation to secure the stream. [Section 5.4.2.1, RFC 6120]

```
I:      <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

2. **[Required]** The receiving entity SHALL reply with a <proceed/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace. [Section 5.4.2.1, RFC 6120]

```
R:      <proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

5.7.3.8.3 STARTTLS Negotiation Fails

[Required] If there is a failure of STARTTLS negotiations, the receiving entity SHALL return a <failure/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace and SHALL close the XML stream. [Section 5.4.2.2, RFC 6120]

```
R:      <failure xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>

R:      </stream:stream>
```

NOTE: A STARTTLS failure is not triggered by TLS errors such as bad_certificate or handshake failure, which are generated and handled during the TLS negotiation itself.

NOTE: If the failure case occurs, the initiating entity MAY attempt to reconnect.

5.7.3.8.4 TLS Negotiation

[Required] After the receiving entity has sent and the initiating entity has received the <proceed/> element, the initiating and receiving entities SHALL proceed to TLS negotiation.

The TLS negotiation and implementation SHALL be in accordance with the requirements defined in Section 5.4, Information Assurance Requirements. Section 5.4 provides detailed guidance and requirements regarding the use of TLS with DoD PKI certificates.

5.7.3.8.5 TLS Success

[Required] If the TLS negotiation is successful, then the initiating and receiving entities SHALL proceed as follows. [Section 5.4.3.3, RFC 6120]

- The initiating entity SHALL send a new initial stream header to the receiving entity over the encrypted connection. The initiating entity SHALL NOT send a closing </stream> tag before sending the new initial stream header, since the receiving entity and initiating entity MUST consider the original stream to be replaced upon success of the TLS negotiation.

- The receiving entity SHALL respond with a new response stream header over the encrypted connection. In this new response stream header, the receiving entity SHALL generate a new stream ID instead of reusing the old stream ID.

- The receiving entity also SHALL send stream features to the initiating entity, which SHALL NOT include the STARTTLS feature, but which SHALL advertise support of SASL negotiation as described in Section 5.7.3.9, Authentication and SASL Negotiation.

5.7.3.8.6 TLS Failure

[Required] If the TLS negotiation results in failure, the receiving entity SHALL terminate the TCP connection. [Section 5.4.3.2, RFC 6120]

5.7.3.8.7 Order of TLS and SASL Negotiation

[Required] Client and server implementations SHALL complete STARTTLS negotiation before proceeding to SASL protocol negotiation; this order of negotiation is necessary to help safeguard authentication information sent during SASL negotiation, as well as to make it possible to base the use of the SASL EXTERNAL mechanism on a certificate provided during prior TLS negotiation (for entities who authenticate using a DoD PKI certificate). [Section 5.3.4, RFC 6120]

5.7.3.8.8 STARTTLS Failure Case

[Required] If the STARTTLS negotiation fails, the receiving entity SHALL return a <failure/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace, terminate the XML stream, and terminate the underlying TCP connection. [Section 5.4.2.2, RFC 6120]

5.7.3.9 Authentication and SASL Negotiation

The XMPP includes a method for adding authentication support to an XML stream by means of an XMPP-specific profile of the SASL protocol. As described in RFC 4422, SASL is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms. [Section 6 of RFC 6120 and RFC 4422]

1. **[Required]** All client and server implementations SHALL support SASL negotiations. [Section 6.2, RFC 6120]

2. **[Required]** The entities involved in an XML stream SHALL consider SASL as mandatory-to-negotiate. [Section 6.3.1, RFC 6120]

3. **[Required]** Anonymous login capability is prohibited. [Instant Messaging STIG, Version 1, Release 2]

NOTE: SASL negotiation follows successful STARTTLS negotiation. The SASL negotiation occurs over the encrypted stream that has already been negotiated.

5.7.3.9.1 Client-to-Server Streams

1. **[Required]** During the prior TLS negotiation, the server SHALL authenticate using a DoD PKI certificate. The client SHALL validate the certificate presented by the server (i.e., shall verify that the certificate is unexpired, unrevoked, and anchored to a trusted DoD CA in accordance with the policies and requirements defined in Section 5.4).

2. **[Required]** The client SHALL authenticate using name and password using the SASL PLAIN mechanism [RFC 4616] as defined below.

   NOTE: As defined by this specification, the SASL PLAIN mechanism SHALL only be used when the underlying XML stream is protected using Transport Layer Security (TLS).

   NOTE: Client authentication using name and password is a minimum requirement. Client authentication using a DoD PKI certificate is preferred. The client in this scenario would comply with the behavior defined for the "initiating entity" in Section 5.7.3.9.2, Server-to-Server Streams.

3. **[Required]** After successful STARTTLS negotiation, the server SHALL offer the SASL PLAIN mechanism to the client during SASL negotiation. The <mechanisms/> element SHALL be qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace. The <mechanisms/> element SHALL contain one <mechanism/> child element including the appropriate value for the PLAIN mechanism. [Section 6.4.1, RFC 6120]

```
S:  <stream:features>
      <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
        <mechanism>PLAIN</mechanism>
        <required/>
        </mechanisms>
    </stream:features>
```

4. **[Required]** The client SHALL select the PLAIN authentication mechanism by sending an <auth/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace and which SHALL include the appropriate value for the PLAIN 'mechanism' attribute. See the following example:

```
C:  <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
        mechanism='PLAIN'>AGp1bGlldAByMG0zMG15cjBtMzA=</auth>
```

   As discussed in RFC 4616, the PLAIN SASL mechanism consists of a single message, a string of [UTF-8] encoded [Unicode] characters, from the client to the server. The client presents a NUL (U+0000) character, followed by the authentication identity (i.e., name), followed by a NUL (U+0000) character, followed by the clear-text password. For additional details, see RFC 4616. [Section 2, RFC 4616]

5. **[Required]** Upon receipt of the message, the server will verify the presented authentication identity and password by performing a directory lookup to a directory service linked to the XMPP server for authenticating the user. [Instant Messaging STIG, Version 1, Release 2]

6. **[Required]** All users SHALL be linked to a directory service, which is linked to the user's home XMPP server. [Instant Messaging STIG, Version 1, Release 2]

7. **[Required]** The server SHALL report the success of the handshake by sending a <success/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace [Section 6.4.6. RFC 6120]:

```
S:        <success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

8. **[Required]** After successful SASL negotiation, the client and server SHALL restart the stream. Upon receiving the <success/> element, the client SHALL initiate a new stream over the existing TLS connection by sending a new initial stream header to the server. The client SHALL NOT send a closing </stream> tag before sending the new initial stream header, since the server and client MUST consider the original stream to be replaced upon sending or receiving the <success/> element. [Section 6.4.6. RFC 6120]

9. **[Required]** Upon receiving the new initial stream header from the client, the server SHALL respond by sending a new response stream header to the client (for which it SHALL generate a new stream ID instead of re-using the old stream ID). [Section 6.4.6, RFC 6120]

10. **[Required]** The server SHALL also send stream features, containing any further available features or containing no features (via an empty <features/> element). [Section 6.4.6, RFC 6120]

```
S:    <stream:features>
          <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
      </stream:features>
```

5.7.3.9.2 Server-to-Server Streams

1. **[Required]** During the prior TLS negotiation, the initiating entity and the receiving entity SHALL mutually authenticate using DoD PKI certificates.

2. **[Required]** After the successful mutual authentication of the receiving entity and the initiating entity during the prior TLS negotiation, the receiving entity SHALL offer the SASL EXTERNAL mechanism (as defined in Appendix A of RFC 4422) to the initiating entity during SASL negotiation. [Section 6.3.4, RFC 6120]

3. **[Required]** The receiving entity SHALL include an empty <required/> element in its advertisement of the SASL feature.

   NOTE: The SASL EXTERNAL mechanism allows the initiating entity to request that the receiving entity use the credentials exchanged during the TLS Handshake process (See RFC 4422, Appendix A and XEP 0178: Best Practices for Use of SASL EXTERNAL with Certificates).

```
R: <stream:features>
       <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
       <mechanism>EXTERNAL</mechanism>
       <required/>
```

```
    </mechanisms>
</stream:features>
```

4. **[Required]** In response to the receiving entity offering the SASL EXTERNAL mechanism, the initiating entity SHALL select the EXTERNAL authentication mechanism by sending an <auth/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace and which SHALL include the appropriate value for the EXTERNAL 'mechanism' attribute and which also includes an empty response of "=". [Section 6.4, RFC 6120 and Section 3, XEP-178]:

```
I:  <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
        mechanism='EXTERNAL'/>=</auth>
```

   NOTE: For the sake of backwards compatibility, the initiating entity MAY alternatively include an authorization identity (base64-encoded as described in RFC 6120) as the XML character data of the <auth/> element, which SHOULD be the same as the 'from' address in the stream header it sent to the initiating entity as defined in XEP-0178.

```
I:   <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
     mechanism='EXTERNAL'>Y29uZmVyZW5jZS5leGFtcGxlLm9yZwo=</auth>
```

5. **[Required]** The receiving entity SHALL report the success of the handshake by sending a <success/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace [Section 6.4.6, RFC 6120]:

```
R:  <success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

6. **[Required]** After successful SASL negotiation, the initiating entity and the receiving entity SHALL restart the stream. Upon receiving the <success/> element, the initiating entity SHALL initiate a new stream over the existing TLS connection by sending a new initial stream header to the receiving entity. The initiating entity SHALL NOT send a closing </stream> tag before sending the new initial stream header, since the receiving entity and initiating entity MUST consider the original stream to be replaced upon sending or receiving the <success/> element. [Section 6.4.6, RFC 6120]

```
I:  <stream:stream
        from='im.example.dod.mil'
        to='chat.example2.dod.mil'
        version='1.0'
        xmlns='jabber:server'
        xmlns:stream='http://etherx.jabber.org/streams'>
```

7. **[Required]** Upon receiving the new initial stream header from the initiating entity, the receiving entity SHALL respond by sending a new response stream header to the initiating entity (for which it SHALL generate a new stream ID instead of reusing the old stream ID). [Section 6.3.2, and Section 6.4.6, RFC 6120]

```
R:  <stream
      from='im.example.dod.mil'
      id='MbbV2FeojySpUIP6J91qaa+TWHM='
      to='chat.example2.dod.mil'
      version='1.0'
      xmlns='jabber:server'
      xmlns='http://etherx.jabber.org/streams'>
```

8. **[Required]** The receiving entity SHALL also send stream features, containing any further available features or containing no features (via an empty <features/> element). [Section 6.4.6, RFC 6120]

5.7.3.9.3 SASL Failure

1. **[Required]** The receiving entity SHALL report failure of the handshake by sending a <failure/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace. [Section 6.4.5, RFC 6120]

2. **[Required]** The particular cause of failure SHALL be communicated in an appropriate child element of the <failure/> element as defined under Section 6.4 (SASL Errors) of RFC 6120. [Section 6.4.5, RFC 6120]

```
R:    <failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
          not-authorized/>
      </failure>
```

3. **[Required]** The receiving entity SHALL allow a configurable number of retries (at least two and no more than three per IM STIG policy).

4. **[Required]** If the initiating entity exceeds the maximum number of retries, the server SHALL return a stream error (which SHALL be either <policy-violation/> or <not-authorized/>). [Section 6.4.5, RFC 6120]

5.7.3.9.4 SASL Errors

For guidance and associated requirements related to SASL errors and defined conditions, see Section 6.5, RFC 6120.

5.7.3.10 Resource Binding

5.7.3.10.1 Overview

The baseline standard, RFC 6120, defines the concept of binding a resource (e.g., a particular client implementation) to an XML stream. After a client authenticates with its home server, the client will bind a specific resource to the stream so that the server can properly address the client. In this process, the server will associate an XMPP resource with the client's bare JID (<localpart@domain>). As described in Section 5.7.3.6, XMPP Addressing, the resourcepart identifier is used for routing purposes to ensure that XMPP traffic is routed to the appropriate client connection. The combination of the resourcepart identifier and the client's bare JID constitute the client's full JID of the form <localpart@domain/resourcepart>. [Section 7.1, RFC 6120]

After a client has successfully bound a resource to the XML stream, it is referred to as a Connected Resource. A compliant server implementation SHALL allow a user to maintain multiple connected resources simultaneously. [Section 7.1, RFC 6120]

5.7.3.10.2 Resource Binding Process

5.7.3.10.2.1 Mandatory-to-Negotiate

1. **[Required]** All client and server implementations SHALL support resource binding. [Section 7.2, RFC 6120]

2. **[Required]** For client-to-server connections, both the client and server SHALL consider resource binding as mandatory-to-negotiate. [Section 7.3.1, RFC 6120]

5.7.3.10.2.2 Advertising Support

[Required] Upon sending a new response stream header to the client after successful SASL negotiation, the server SHALL include a <bind/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-bind' namespace in the stream features it presents to the client. [Section 7.4, RFC 6120]

```
S:   <stream:features>
         <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
     </stream:features>
```

5.7.3.10.2.1 Server-Generated Resource Identifier

1. **[Required]** A server implementation SHALL be able to generate an XMPP resourcepart on behalf of a client. [Section 7.6, RFC 6120]

2. **[Required]** A resourcepart SHALL at a minimum, be unique among the connected resources for a specific local account in the form of <localpart@domain>. Enforcement of this policy is the responsibility of the server.

3. **[Required]** A client SHALL request a server-generated resourcepart by sending an Info/Query (IQ) stanza of type “set” (see Section 5.7.3.12.2, Roster-Related Methods) containing an empty <bind/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-bind' namespace. [Section 7.6.1, RFC 6120]

```
C:  <iq id='tn281v37' type='set'>
        <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
    </iq>
```

4. **[Required]** Once the server has generated an XMPP resourcepart for the client, it SHALL return an IQ stanza of type "result" to the client, which SHALL include a <jid/> child element that specifies the full JID for the connected resource as determined by the server. [Section 7.6.1, RFC 6120]

```
S:   <iq id='tn281v37' type='result'>
       <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
         <jid>
            juliet@im.example.com/4db06f06-1ea4-11dc-aca3-000bcd821bfb
         </jid>
       </bind>
     </iq>
```

5.7.3.10.3 Error Cases Associated with Server-Generated Resource Identifiers

For guidance and associated requirements related to Server-Generated Resource Identifiers, see Section 7.6.2, RFC 6120.

5.7.3.11 XML Stanzas

After a client and a server (or two servers) have completed stream negotiation, either party can send XML stanzas. For the ‘jabber:client’ and ‘jabber:server’ content namespaces, three XML stanza are defined: <message/>, <presence/>, and <iq/>. There are five common attributes associated with these three stanza types. These common attributes and the basic semantics of these three stanza types are defined below.

[Required] Client and server implementations SHALL support the syntax and semantics associated with the message, presence, and IQ stanzas. [See the following sections: 5.7.3.11.1 through 5.7.3.11.3]

5.7.3.11.1 Common Attributes

5.7.3.11.1.1 'to' Attribute

The 'to' attribute specifies the JID of the intended recipient of a stanza. [Section 8.1.1, RFC 6120]

```
<message to='robert@example1.dod.mil'>
   <body>Hello</body>
</message>
```

1. **[Required]** The following rules SHALL be followed regarding the use of the 'to' attribute in the context of XML streams qualified by the 'jabber:client' namespace (i.e., client-to-server streams) [Section 8.1.1.1, RFC 6120]:

   a. A stanza with a specific intended recipient SHALL possess a 'to' attribute whose value is an XMPP address.

   b. A stanza sent from a client to a server for direct processing by the server on behalf of the client (e.g., presence sent to the server for broadcasting to other entities) SHALL NOT possess a 'to' attribute.

2. **[Required]** The following rules SHALL be followed regarding the use of the 'to' attribute in the context of XML streams qualified by the 'jabber:server' namespace (i.e., server-to-server streams) [Section 8.1.1.2, RFC 6120]:

   a. A stanza SHALL possess a 'to' attribute whose value is an XMPP address; if a server receives a stanza that does not meet this restriction, it SHALL generate an <improper-addressing/> stream error.

   b. The domain identifier portion of the JID in the 'to' attribute SHALL match a hostname serviced by the receiving server; if a server receives a stanza that does not meet this restriction, it SHALL generate a <host-unknown/> or <host-gone/> stream error.

5.7.3.11.1.2 ‘from’ Attribute

The ‘from’ attribute specifies the JID of the sender. [Section 8.1.2, RFC 6120]

```
<message from='john@im.example1.dod.mil/office'
         to='robert@example1.dod.mil'>
   <body>Hello</body>
</message>
```

1. **[Required]** The following rules SHALL be followed regarding the use of the ‘from’ attribute in the context of XML streams qualified by the ‘jabber:client’ namespace (i.e., client-to-server streams) [Section 8.1.2.1, RFC 6120]:

   a. When the server receives an XML stanza from a client, the server SHALL add a ‘from’ attribute to the stanza or override the ‘from’ attribute specified by the client, where the value of the ‘from’ attribute is the full JID (<localpart@domainpart/resource>) determined by the server for the connected resource that generated the stanza or the bare JID (<localpart@domainpart>) in the case of subscription-related presence stanzas.

   b. When the server generates a stanza from the server itself for delivery to the client, the stanza SHALL include a ‘from’ attribute whose value is the bare JID (i.e., <domain>) of the server as agreed upon during stream negotiation (e.g., based on the ‘to’ attribute of the initial stream header).

   c. When the server generates a stanza from the server for delivery to the client on behalf of the account of the connected client (e.g., in the context of data storage services provided by the server on behalf of the client), the stanza SHALL either (a) not include a ‘from attribute or (b) include a 'from' attribute whose value is the account's bare JID (<localpart@domainpart>).

   d. A server SHALL NOT send to the client a stanza without a ‘from’ attribute if the stanza was not generated by the server (e.g., if it was generated by another client or another server).

   e. When a client receives a stanza that does not include a ‘from’ attribute, it SHALL assume that the stanza is from the user’s account on the server.

2. **[Required]** The following rules SHALL be followed regarding the use of the ‘from’ attribute in the context of XML streams qualified by the ‘jabber:server’ namespace (i.e., server-to-server streams) [Section 8.1.2.2, RFC 6120]:

a. A stanza SHALL possess a 'from' attribute whose value is an XMPP address; if a server receives a stanza that does not meet this restriction, it SHALL generate an <improper-addressing/> stream error.

b. The domain identifier portion of the JID contained in the 'from' attribute SHALL match the hostname of the sending server (or any validated domain thereof) as communicated in the SASL negotiation; if a server receives a stanza that does not meet this restriction, it SHALL generate an <invalid-from/> stream error.

Enforcement of these rules helps to prevent certain denial of service attacks.

5.7.3.11.1.3 'id' Attribute

As discussed in Section 8.1.3 of RFC 6120, the 'id' attribute is used by the entity that generates a stanza ("the originating entity") to track any response or error stanza that it might receive in relation to the generated stanza from another entity (such as an intermediate server or the intended recipient). It is up to the originating entity whether the value of the 'id' attribute will be unique only within its current stream or unique globally.

1. **[Required]** For <iq/> stanzas, the originating entity SHALL include an 'id' attribute. [Section 8.1.3, RFC 6120]

   NOTE: For <message/> and <presence/> stanzas, it is recommended for the originating entity to include an 'id' attribute. [Section 8.1.3, RFC 6120]

2. **[Required]** If the generated stanza includes an 'id' attribute, then it is required for the associated response or error stanza to also include an 'id' attribute, where the value of the 'id' attribute SHALL match that of the generated stanza. [Section 8.1.3, RFC 6120]

5.7.3.11.1.4 'type' Attribute

As discussed in Section 8.1.4 of RFC 6120, the 'type' attribute specifies the purpose or context of the message, presence, or IQ stanza. The particular allowable values for the 'type' attribute vary depending on whether the stanza is a message, presence, or IQ stanza. The defined values for message and presence stanzas are specific to instant messaging and presence applications and therefore are defined in subsequent sections of this specification (e.g., 5.7.3.13, 5.7.3.14, 5.7.3.15, 5.7.3.17), whereas the values for IQ stanzas specify the role of an IQ stanza in a structured request-response exchange and therefore are specified under Section 5.7.3.11.2.3, IQ Semantics. The only 'type' value common to all three stanzas is "error"; see Section 5.7.3.11.3, Stanza Errors. [Section 8.1.4, RFC 6120]

5.7.3.11.1.5 ‘xml:lang’ Attribute

NOTE: A stanza SHOULD possess an ‘xml:lang’ attribute if the stanza contains XML character data that is intended to be presented to a human user. The value of the ‘xml:lang’ attribute specifies the default language of any such human-readable XML character data. [Section 8.1.5, RFC 6120]

```
<presence from='robert@example1.dod.mil/office' xml:lang='en'>
  <show>dnd</show>
  <status>Hello</status>
</presence>
```

NOTE: If an outbound stanza generated by a client does not possess an ‘xml:lang’ attribute, the client’s server SHOULD add an ‘xml:lang’ attribute whose value is that which is specified for the stream. [Section 8.1.5, RFC 6120]

1. **[Required]** If an inbound stanza received by a client or server does not possess an ‘xml:lang’ attribute, an implementation SHALL assume that the default language is that which is specified for the stream. [Section 8.1.5, RFC 6120]

2. **[Required]** A server SHALL NOT modify or delete the ‘xml:lang’ attribute of stanzas it receives from other entities. [Section 8.1.5, RFC 6120]

5.7.3.11.2 Basic Semantics

5.7.3.11.2.1 Message Semantics

As discussed in Section 8.2.1 of RFC 6120, the <message/> stanza can be seen as a “push” mechanism whereby one entity pushes information to another entity. For additional clarification and requirements associated with the use of the <message/> stanza in the context of one-to-one chat sessions and multi-user chat sessions, see Sections 5.7.3.15 and 5.7.3.17 respectively.

5.7.3.11.2.2 Presence Semantics

As discussed in Section 8.2.2 of RFC 6120, the <presence/> stanza can be seen as a specialized broadcast or “publish-subscribe” mechanism, whereby multiple entities receive information (in this case, network availability information) about an entity to which they have subscribed. For additional clarification and requirements associated with the use of the <presence/> stanza to enable the exchange of presence information, see Sections 5.7.3.13, Presence Subscription Management, and 5.7.3.14, Exchanging Presence Information.

5.7.3.11.2.3 IQ Semantics

As discussed in Section 8.2.3 of RFC 6120, the Info/Query (IQ) stanza provides a request-response mechanism. The semantics of the IQ stanza enables an entity to make a request of, and receive a response from, another entity. The data content of the request and response is defined by the schema or other structural definition associated with the XML namespace that qualifies the direct child element of the IQ element and the interaction is tracked by the requesting entity through use of the ‘id’ attribute. [Section 8.2.3, RFC 6120]

1. **[Required]** When a client or server implementation generates or processes an IQ stanza, the following rules apply [Section 8.2.3, RFC 6120]:

    a. An IQ stanza SHALL include the ‘id’ attribute.

    b. An IQ stanza SHALL include the ‘type’ attribute.

    c. The value of the ‘type’ attribute for IQ stanzas SHALL be one of the following (if the value is other than one of the following strings, the recipient or an intermediate server SHALL return a stanza error of <bad-request/>):

        (1) get – The stanza requests information (e.g., the stanza inquires about data which is needed in order to complete further operations)

        (2) set – The stanza provides data that is needed for an operation to be completed (e.g., it sets new values, replaces existing values)

        (3) result – The stanza is a response to a successful “get” or “set” request

        (4) error – The stanza reports an error that has occurred regarding the processing or delivery of a previously sent “get” or “set” request

    d. An entity that receives an IQ request of type “get” or “set” SHALL reply with an IQ response of type “result” or “error”. The response SHALL preserve the 'id' attribute of the request.

    e. An entity that receives a stanza of type “result” or “error” SHALL NOT respond to the stanza by sending a further IQ response of type “result” or “error”.

    f. An IQ stanza of type “get” or “set” SHALL contain exactly one child element, which specifies the semantics of the particular request.

    g. An IQ stanza of type “result” SHALL include zero or one child element.

h. An IQ stanza of type "error" SHALL include an <error/> child.

5.7.3.11.3 Stanza Errors

[Required] Client and server implementations SHALL comply with the mandatory requirements defined in Section 8.3 of RFC 6120.

5.7.3.11.4 Server Rules for Processing XML Stanzas

5.7.3.11.4.1 Rules for Processing XML Stanzas to Remote Domains

[Required] If the domainpart of the JID contained in the 'to' attribute does not match one of the configured hostnames of the server itself, the server SHALL attempt to route the stanza to the remote domain. [Section 10.4, RFC 6120]

NOTE: These rules apply only to client-to-server streams. As described under Section 5.7.3.9.2, Server-to-Server Streams, a server SHALL NOT accept a stanza over a server-to-server stream if the domainpart of the JID in the 'to' attribute does not match a hostname serviced by the receiving server. [Section 10.4, RFC 6120]

5.7.3.11.4.1.1 Server-to-Server Stream Already Exists

[Required] If a server-to-server stream already exists between the two domains, the sender's server SHALL attempt to route the stanza to the authoritative server for the remote domain over the existing stream. [Section 10.4.1, RFC 6120]

5.7.3.11.4.1.2 No Server-to-Server Stream Currently Exists

[Required] If no server-to-server stream exists between the two domains, the sender's server SHALL proceed as follows [Section 10.4.2, RFC 6120]:

- Resolve the hostname of the remote domain, as described in Section 5.7.3.7.1.1, Hostname Resolution.
- Negotiate a server-to-server stream between the two domains (as defined in Section 5.7.3.8, TLS and STARTTLS Negotiation, and Section 5.7.3.9, Authentication and SASL Negotiation.
- Route the stanza to the authoritative server for the remote domain over the newly-established stream.

5.7.3.11.4.1.3 Error Handling

1. **[Required]** If the routing of a stanza to the intended recipient's server is unsuccessful, the sender's server SHALL return an error to the sender. If resolution of the remote domain is unsuccessful, the stanza error SHALL be <remote-server-not-found/>. If the resolution succeeds, but the XML streams cannot be negotiated, the stanza error SHALL be <remote-server-timeout/>. [Section 10.4.3, RFC 6120]

2. **[Required]** If stream negotiation with the intended recipient's server is successful but the remote server cannot deliver the stanza to the recipient, the remote server SHALL return an appropriate error to the sender by way of the sender's server. [Section 10.4.3, RFC 6120]

5.7.3.11.4. 2 Rules for Processing XML Stanzas to Local Domain

[Required] If the hostname of the domainpart of the JID contained in the 'to' attribute matches one of the configured hostnames of the server, the server SHALL first determine if the hostname is serviced by the server itself or by a specialized local service. If the latter, the server SHALL route the stanza to that service. If the former, the server SHALL proceed as follows [Section 10.5.3, RFC 6120]:

5.7.3.11.4.2.1 No Such User

[Required] If there is no local account associated with the <localpart@domainpart>, how the stanza is processed depends on the stanza type. [Section 10.5.3.1, RFC 6120]

- For a message stanza, the server SHALL return a <service-unavailable/> stanza error to the sender.
- For a presence stanza, the server SHALL ignore the stanza.
- For an IQ stanza, the server SHALL return a <service-unavailable/> stanza error to the sender.

5.7.3.11.4.2.2 Bare JID

[Required] If the JID contained in the 'to' attribute is of the form <localpart@domainpart>, how the stanza is processed depends on the stanza type. [Section 10.5.3.2, RFC 6120]

- For a message stanza, if at least one connected resource for the account exists, the server SHALL deliver it to at least one of the connected resources. If there exists no connected resource, the server SHALL either return a <service-unavailable/> stanza error or store the message offline for delivery when the account next has a connected resource.

- For a presence stanza, if at least one connected resource that has sent initial presence exists (i.e., has a “presence session”), the server SHALL deliver it to such resources. If no connected resource exists, the server SHALL ignore the stanza.

- For an IQ stanza, the server SHALL handle it directly on behalf of the intended recipient.

5.7.3.11.4.2.3 Full JID

1. **[Required]** If the JID contained in the ‘to’ attribute is of the form <localpart@domainpart/resource> and there is no connected resource that exactly matches the full JID, the stanza SHALL be processed as if the JID were of the form <localpart@domainpart>. [Section 10.5.4, RFC 6120]

2. **[Required]** If the JID contained in the ‘to’ attribute is of the form <localpart@domainpart/resource> and there is a connected resource that exactly matches the full JID, the server SHALL deliver the stanza to that connected resource. [Section 10.5.4, RFC 6120]

5.7.3.12 Roster Management

In XMPP, a user’s contact list is referred to as a roster. As defined in RFC 6121, a user’s roster is stored by the user’s server on the user’s behalf so that the user can access roster information from any device. This section addresses the protocol mechanics that permit a client to retrieve a roster from its home server and to add, delete, and modify items within the roster.

5.7.3.12.1 Roster-Related Elements and Attributes

1. **[Required]** Client and server implementations SHALL use IQ stanzas containing a <query/> child element qualified by the ‘jabber:iq:roster’ namespace to manage elements in a roster. [Section 2.1, RFC 6121]

   NOTE: As discussed in Section 2.1.1 of RFC 6121, the ‘ver’ attribute is a string that identifies a particular version of the roster information. The ‘ver’ attribute is only generated by the server. An implementation treats the ‘ver’ attribute of the <query/> element qualified by the ‘jabber:iq:roster’ namespace as an identifier of the particular version of roster information being sent or received. Inclusion of the ‘ver’ attribute is recommended. [Section 2.1.1, RFC 6121]

2. **[Required]** Client and server implementations SHALL support the ‘subscription’ attribute and the allowable subscription-related values for this attribute. The state of the presence subscription in relation to a roster item is captured in the ‘subscription’ attribute of the <item/> element. The allowable subscription-related values for this attribute are [Section 2.1.2.5, RFC 6121]:

   a. “none” – the user does not have a subscription to the contact’s presence, and the contact does not have a subscription to the user’s presence; this is the default value, so if the subscription attribute is not included, then the state is to be understood as “none”

   b. “to” – the user has a subscription to the contact’s presence, but the contact does not have a subscription to the user’s presence

   c. “from – the contact has a subscription to the user’s presence, but the user does not have a subscription to the contact’s presence

   d. “both” – both the user and the contact have subscriptions to each other’s presence (also called a “mutual subscription”)

3. **[Required]** In a roster result, the client SHALL ignore values of the ‘subscription’ attribute other than “none”, “to”, “from”, or “both”. [Section 2.1.2.5, RFC 6121]

4. **[Required]** In a roster push, the client SHALL ignore values of the ‘subscription’ attribute other than “none”, “to”, “from”, “both”, or “remove”. [Section 2.1.2.5, RFC 6121]

5. **[Required]** In a roster set, the value of the ‘subscription’ can have a value of “remove”, which indicates that the item is to be removed from the roster; the server SHALL ignore all values of the ‘subscription’ attribute other than “remove”. [Section 2.1.2.5, RFC 6121]

6. **[Required]** Client implementations SHALL support the ‘name’ attribute, which is used to specify the “handle” to be associated with the JID, as determined by the user (not the contact). It is optional for a client to include the ‘name’ attribute when adding or updating a roster item. [Section 2.1.2.4, RFC 6121]

7. **[Required]** Client and server implementations SHALL support the ‘ask’ attribute, which is used to specify presence subscriptions sub-state. [Section 2.1.2.2, RFC 6121]

8. **[Required]** A value of “subscribe” in the ‘ask’ attribute is used to signal a “Pending Out” sub-state as described under Section 3.1.2 of RFC 6121. A server SHALL include the ‘ask’ attribute to inform the client of “Pending Out” sub-state. [Section 2.1.2.2, RFC 6121]

9. **[Required]** Client and server implementations SHALL support the <group/> child element which is used to specify a category or "bucket" into which the roster item is to be grouped by a client. It is optional for a client to include the <group/> element when adding or updating a roster item. If a roster set (Roster Set) includes no <group/> element, then the item is to be interpreted as being affiliated with no group. [Section 2.1.2.6, RFC 6121]

   NOTE: An <item/> element MAY contain more than one <group/> element, which means that roster groups are not exclusive. [Section 2.1.2.6, RFC 6121]

5.7.3.12.2 Roster-Related Methods

1. **[Required]** A client implementation SHALL have the ability to generate a Roster Get. A Roster Get is a client's request for the server to return the roster; syntactically it is an IQ stanza of type "get" sent from client to server and containing a <query/> element qualified by the 'jabber:iq:roster' namespace, where the <query/> element SHALL NOT contain any <item/> child elements. Likewise, a compliant server implementation SHALL be able to process this request. The expected outcome of sending a roster get is for the server to return a roster result. [Section 2.1.3, RFC 6121]

```
C:      <iq from='john.smith@chat.dod.mil/desktop client'
            id='bv1bs71f'
            type='get'>
          <query xmlns='jabber:iq:roster'/>
        </iq>
```

2. **[Required]** A server implementation SHALL be able to process a Roster Get.

3. **[Required]** A server implementation SHALL have the ability to generate a Roster Result. A Roster Result is the server's response to a roster get; syntactically it is an IQ stanza of type "result" sent from server to client and containing a <query/> element qualified by the 'jabber:iq:roster' namespace. The <query/> element in a roster result contains one <item/> element for each contact and therefore can contain more than one <item/> element. The ability to generate this response is required for server implementations. Likewise, a compliant client implementation SHALL be able to process this response. [Section 2.1.4, RFC 6121]

```
S:  <iq id='bv1bs71f'
             to='robert.jones@chat.dod.mil/desktop client'
             type='result'>
          <query xmlns='jabber:iq:roster' ver='ver7'>
             <item jid='mike@example2.dod.mil'/>
             <item jid='bob@example1.dod.mil'/>
          </query>
       </iq>
```

4. **[Required]** A client implementation SHALL be able to process a Roster Result.

5. **[Required]** A client implementation SHALL have the ability to generate a Roster Set. A Roster Set is a client's request for the server to modify (i.e., create, update, or delete) a roster item; syntactically it is an IQ stanza of type "set" sent from client to server and containing a <query/> element qualified by the 'jabber:iq:roster' namespace. [Section 2.1.5, RFC 6121]

   The following rules apply to roster sets:

   a. The <query/> element SHALL contain one and only one <item/> element.

   b. The server SHALL ignore any value of the 'subscription' attribute other than "remove".

```
C:  <iq from='robert@example2.dod.mil'
        id='rs1'
        type='set'>
      <query xmlns='jabber:iq:roster'>
       <item jid='bob@chat.dod.mil'/>
      </query>
    </iq>
```

6. **[Required]** A server implementation SHALL be able to process a Roster Set.

7. **[Required]** A server implementation SHALL have the ability to generate a Roster Push. A Roster Push is a newly created, updated, or deleted roster item that is sent from the server to the client; syntactically it is an IQ stanza of type "set" sent from server to client and containing a <query/> element qualified by the 'jabber:iq:roster' namespace. [Section 2.1.6, RFC 6121]

   The following rules apply to roster pushes:

a. The <query/> element in a roster push SHALL contain one and only one <item/> element.

b. A receiving client SHALL ignore the stanza unless it has no 'from' attribute (i.e., implicitly from the user's bare JID) or it has a 'from' attribute whose value matches the user's bare JID <user@domain>.

```
S:    <iq id='a78b4q6ha463'
           to='john@example1.dod.mil/desktop client'
           type='set'>
       <query xmlns='jabber:iq:roster'>
         <item jid='robert@example2.dod.mil'/>
       </query>
      </iq>
```

8. **[Required]** A client implementation SHALL be able to process a Roster Push.

9. **[Required]** As mandated by the semantics of the IQ stanza as defined in [RFC 6120] each resource that receives a roster push SHALL reply with an IQ stanza of type 'result' (or 'error').

```
C:  <iq from='john@example1.dod.mil/desktop client'
        id='a78b4q6ha463'
        type='result'/>
```

5.7.3.12.3 Retrieving the Roster on Login

1. **[Required]** Upon authenticating with a server and binding a resource (thus becoming a connected resource), a client SHALL request the roster before sending initial presence. A client requests the roster by sending a roster get over its stream to the server. [Section 2.2, RFC 6121]

   NOTE: Because receiving the roster is not necessarily desirable for all resources, e.g., a connection with limited bandwidth, the client's request for the roster in bandwidth-limited environments is not mandatory. [Section 2.2, RFC 6121]

   NOTE: If a connected resource or available resource requests the roster, it is referred to as an interested resource. [Section 2.2, RFC 6121]

2. **[Required]** The server SHALL process the roster get and SHALL return a roster result containing a <query/> element qualified by the 'jabber:iq:roster' namespace. The <query/>

element in a roster result SHALL contain one <item/> element for each contact and therefore can contain more than one <item/> element. [Section 2.1.3 and Section 2.2, RFC 6121]

```
C: <iq from='john@example1.dod.mil'
       id='hu2bac18'
       type='get'>
     <query xmlns='jabber:iq:roster'/>
    </iq>

S:  <iq id='hu2bac18'
       to='john@example1.dod.mil/desktop client'
       type='result'>
      <query xmlns='jabber:iq:roster' ver='ver11'>
         <item jid='robert@example2.dod.mil'
               name='Robert'
               subscription='both'>
           <group>Friends</group>
         </item>
         <item jid='mike@example2.dod.mil'
               name='Mike'
               subscription='from'/>
         <item jid='bob@example1.dod.mil'
               name='Bob'
               subscription='both'/>
      </query>
    </iq>
```

3. **[Required]** If the server cannot process the roster get, it SHALL return an appropriate stanza error as described in RFC 6120.

5.7.3.12.4 Adding a Roster Item

1. **[Required]** A client SHALL support the ability to add an item to the roster by sending a roster set containing a new item. [Section 2.3.1, RFC 6121]

```
C: <iq from='john@example1.dod.mil/desktop client'
        id='ph1xaz53'
        type='set'>
     <query xmlns='jabber:iq:roster'>
        <item jid='robert@example2.dod.mil'
            name='Robert'>
        <group>Friends</group>
       </item>
     </query>
   </iq>
```

2. **[Required]** If the server can successfully process the roster set for the new item (i.e., if no error occurs), it SHALL create the roster item in persistent storage. The server SHALL then return an IQ stanza of type "result" to the connected resource that sent the roster set. [Section 2.3.2, RFC 6121]

3. **[Required]** The server SHALL also send a roster push containing the new roster item to all of the user's interested resources, including the resource that generated the roster set. [Section 2.3.2, RFC 6121]

4. **[Required]** If the server cannot successfully process the roster set, it SHALL return a stanza error. For additional details, see Section 2.3.3 of RFC 6121.

5.7.3.12.5 Updating a Roster Item

1. **[Required]** A client SHALL support the ability to update a roster item by sending a roster set to the server. Because a roster item is atomic, the item SHALL be updated exactly as provided in the roster set. [Section 2.4.1, RFC 6121]

   NOTE: There are several reasons why a client might update a roster item [Section 2.4.1, RFC 6121]:

   a. Adding a group
   b. Deleting a group

2. **[Required]** As with adding a roster item, if the roster item can be successfully processed, then the server SHALL update the roster information in persistent storage, send a roster push to the entire user's interested resources, and send an IQ result to the initiating resource. [Section 2.4.2, RFC 6121]

5.7.3.12.6 Deleting a Roster Item

1. **[Required]** A client SHALL support the ability to delete a roster item by sending a roster set and specifying the value of the 'subscription' attribute to "remove". [Section 2.5.1, RFC 6121]

```
C: <iq from='john@example1.dod.mil/desktop client'
        id='hm4hs97y'
        type='set'>
      <query xmlns='jabber:iq:roster'>
        <item jid='robert@example2.dod.mil'
              subscription='remove'/>
             </query>
   </iq>
```

2. **[Required]** As with adding a roster item, if the server can successfully process the roster set then it SHALL update the roster information in persistent storage, send a roster push to all of the user's interested resources (with the 'subscription' attribute set to a value of 'remove'), and send an IQ result to the initiating resource. [Section 2.5.2, RFC 6121]

3. **[Required]** The user's server SHALL generate one or more subscription-related presence stanzas, as per the following use cases [Section 2.5.2, RFC 6121]:

   a. If the user has a presence subscription to the contact, then the user's server SHALL send a presence stanza of type "unsubscribe" to the contact (to unsubscribe from the contact's presence).

   b. If the contact has a presence subscription to the user, then the user's server SHALL send a presence stanza of type "unsubscribed" to the contact (to cancel the contact's subscription to the user), or both.

   c. If the presence subscription is mutual, then the user's server SHALL send both a presence stanza of type "unsubscribe" and a presence stanza of type "unsubscribed" to the contact.

```
S:  <presence from='john@example1.dod.mil'
              id='lm3ba81g'
              to='robert@example2.dod.mil'
              type='unsubscribe'/>
```

4. **[Required]** If the value of the 'jid' attribute specifies an item that is not in the roster, then the server SHALL return an <item-not-found/> stanza error. [Section 2.5.3, RFC 6121]

5.7.3.13 Presence Subscription Management

As discussed in RFC 2778, presence technology allows a user to subscribe to another user's availability status and to be notified when that state changes. Before a particular user is permitted to receive information/updates regarding another user's presence, that exchange SHALL first be authorized using a basic subscription request and approval process. When an entity receives a presence subscription request, the entity can either accept or deny the request. An entity that has a subscription to a user's presence or to which a user has a presence subscription is called a "contact". In XMPP, a subscription lasts across presence sessions; indeed, it lasts until the contact unsubscribes or the user cancels the previously-granted subscription. In XMPP, presence subscription management is accomplished through the use of presence stanzas with specially defined attributes ("subscribe", "unsubscribe", "subscribed", and "unsubscribed").

5.7.3.13.1 Subscription Requests

A Subscription Request is a request from a user for authorization to permanently subscribe to a contact's presence information; syntactically it is a presence stanza whose 'type' attribute has a value of "subscribe".

5.7.3.13.1.1 Rules for Client Generation of Outbound Subscription Requests

1. **[Required]** A client implementation SHALL be capable of generating a subscription request by sending a presence stanza of type "subscribe". [Section 3.1.1, RFC 6121]

```
UC: <presence id='xk3h1v69'
              to='john@example1.dod.mil'
              type='subscribe'/>
```

2. **[Required]** When the client sends a presence subscription request to a potential instant messaging and presence contact, the value of the 'to' attribute SHALL be a bare JID <contact@domain> rather a full JID <contact@domain/resource>. [Section 3.1.1, RFC 6121]

   NOTE: For tracking purposes, a client SHOULD include an 'id' attribute in a presence subscription request.

5.7.3.13.1.2 Rules for Server Processing of Outbound Subscription Requests

1. **[Required]** Upon receiving the outbound presence subscription request, the user's server SHALL comply with the following rules for Server Processing of Outbound Subscription Requests as defined below [Section 3.1.2, RFC 6121]:

a. Before processing the request, the user's server SHALL check the syntax of the JID contained in the 'to' attribute. If the JID is of the form <localpart@domain/resourcepart> instead of <localpart@domain>, the user's server SHALL treat it as if the request had been directed to the contact's bare JID and modify the 'to' address accordingly.

b. If the potential contact is hosted on the same server as the user, then the server SHALL adhere to the Rules for Server Processing of Inbound Subscription Requests (see below) and SHALL deliver it to the local contact.

c. If the potential contact is hosted on a remote server, the user's server SHALL then route the stanza to that remote domain in accordance with the Server Rules for Processing XML Stanzas (e.g., see Section 5.7.3.11.4.1, Rules for Processing XML Stanzas to Remote Domains).

2. **[Required]** When a server processes or generates an outbound presence stanza of type "subscribe", "subscribed", "unsubscribe", or "unsubscribed", the server SHALL stamp the outgoing presence stanza with the bare JID <localpart@domain> of the sending entity. Enforcement of this rule simplifies the presence subscription model and helps to prevent presence leaks. [Section 3.1.2, RFC 6121]

3. **[Required]** If the presence subscription request cannot be locally delivered or remotely routed (e.g., because the request is malformed, the local contact does not exist, the remote server does not exist, an attempt to contact the remote server times out, or any other error determined or experienced by the user's server), then the user's server SHALL return an appropriate error stanza to the user. [Section 3.1.2, RFC 6121]

4. **[Required]** After locally delivering or remotely routing the presence subscription request, the user's server SHALL then send a roster push to all of the user's interested resources, containing the potential contact with a subscription state of "none" and with notation that the subscription is pending (via an 'ask' attribute whose value is "subscribe"). [Section 3.1.2, RFC 6121]:

```
US: <iq id='b89c5r7ib574'
       to='john.smith@chat.dod.mil/desktop client'
       type='set'>
     <query xmlns='jabber:iq:roster'>
       <item ask='subscribe'
               jid='robert.jones@example2.dod.mil/desktop client'
               subscription='none'/>
     </query>
    </iq>
```

NOTE: If a remote contact does not approve or deny the subscription request within a configurable amount of time, the user's server SHOULD resend the subscription request to the contact based on an implementation-specific algorithm (e.g., whenever a new resource becomes available for the user, or after a certain amount of time has elapsed); this helps to recover from transient, silent errors that might have occurred when the original subscription request was routed to the remote domain. When doing so, it is recommended for the server to include an 'id' attribute so that it can track responses to the resent subscription request. [Section 3.1.2, RFC 6121]

5.7.3.13.1.3 Rules for Server Processing of Inbound Subscription Requests

1. **[Required]** Before processing the inbound presence subscription request, the contact's server SHALL check the syntax of the JID contained in the 'to' attribute. If the JID is of the form <contact@domain/resource> instead of <contact@domain>, the contact's server SHALL treat it as if the request had been directed to the contact's bare JID and modify the 'to' address accordingly. [Section 3.1.3, RFC 6121]

2. **[Required]** When processing the inbound presence subscription request, the user's server SHALL comply with the following rules for Server Processing of Inbound Subscription Requests as defined below [Section 3.1.3, RFC 6121]:

   a. Above all, the contact's server SHALL NOT automatically approve subscription requests on the contact's behalf (unless the contact has configured its account to automatically approve subscription requests). Instead, the contact's server SHALL deliver that request to the contact's available resource(s) for approval or denial by the contact.

   b. If the contact exists and the user already has a subscription to the contact's presence, then the contact's server SHALL auto-reply on behalf of the contact by sending a presence stanza of type "subscribed" from the contact's bare JID to the user's bare JID.

   c. If the contact does not exist, then the contact's server SHALL automatically return a presence stanza of type "unsubscribed" to the user.

   d. Otherwise, if there is at least one available resource associated with the contact when the subscription request is received by the contact's server, then the contact's server SHALL broadcast that subscription request to all of the contact's available resources.

   e. Otherwise, if the contact exists, the user does not already have a subscription to the contact's presence, and the contact has no available resources when the subscription request is received by the contact’s server, then the contact’s server SHALL keep a record of the complete presence stanza comprising the subscription request, including any extended content contained therein, and deliver the request when the contact next has an available resource. The contact’s server SHALL continue to deliver the subscription request whenever the contact creates an available resource, until the contact either approves or denies the request.

5.7.3.13.1.4 Rules for Client Processing of Inbound Subscription Requests

1. **[Required]** When the contact’s client receives a subscription request from the user, it SHALL present the request to the contact for approval (unless the contact has explicitly configured the client to automatically approve or deny some or all subscription requests). [Section 3.1.4, RFC 6121]

2. **[Required]** A client implementation SHALL be capable of generating a subscription approval by sending a presence stanza of type “subscribed”.

```
CC: <presence id='h4v1c4kj'
              to='robert@example2.dod.mil'
              type='subscribed'/>
```

3. **[Required]** A client implementation SHALL be capable of denying a subscription request by sending a presence stanza of type “unsubscribed”. [Section 3.1.4, RFC 6121]

```
CC: <presence id='h4v1c4kj'
              to='robert@example2.dod.mil'
              type='unsubscribed'/>
```

NOTE: If the subscription request is approved by the contact, the contact’s client SHOULD send a subscription request to the user automatically. This assumes that the desired end state is a mutual subscription. [Section 3.1.5, RFC 6121]

5.7.3.13.1.5 Rules for Server Processing of Outbound Subscription Approval

1. **[Required]** When the contact’s client sends the subscription approval, the contact’s server SHALL stamp the outbound stanza with the bare JID <localpart@domain> of the contact and locally deliver or remotely route the stanza to the user. [Section 3.1.5, RFC 6121]

```
CS: <presence from='john@example1.dod.mil'
              id='h4v1c4kj'
              to='robert@example2.dod.mil'
              type='subscribed'/>
```

2. **[Required]** The contact's server then SHALL send an updated roster push to all of the contact's interested resources, with the 'subscription' attribute set to a value of "from". [Section 3.1.5, RFC 6121]

3. **[Required]** The contact's server SHALL then also send current presence to the user from each of the contact's available resources. [Section 3.1.5, RFC 6121]

   NOTE: In order to subscribe to the user's presence, the contact's client should then send a subscription request to the user. It is assumed that the normal, desired end state is a mutual subscription.

5.7.3.13.1.6 Rules for Server Processing of Inbound Subscription Approval

1. **[Required]** When the user's server receives the subscription approval, it SHALL first check if the contact is in the user's roster with subscription='none' or subscription='from' and the 'ask' flag set to "subscribe" (see Appendix A of RFC 6121). If this check is successful, then the user's server SHALL proceed as follows [Section 3.1.6, RFC 6121]:

   a. Deliver the inbound subscription approval to all of the user's interested resources. This SHALL occur before sending the roster push described in the next step. [Section 3.1.6, RFC 6121]

```
US: <presence from='john@example1.dod.mil'
              id='h4v1c4kj'
              to='robert@example2.dod.mil'
              type='subscribed'/>
```

   b. Initiate a roster push to all of the user's interested resources, containing an updated roster item for the contact with the 'subscription' attribute set to a value of "to" (if the subscription state was "None + Pending Out" or "None + Pending Out+In") or "both" (if the subscription state was "From + Pending Out"). See Table 5 of Appendix A of RFC 6121. [Section 3.1.6, RFC 6121]

```
US: <iq id='b89c5r7ib576'
        to='robert@example2.dod.mil/desktop client'
        type='set'>
      <query xmlns='jabber:iq:roster'>
        <item jid='john@example1.dod.mil'
              subscription='to'/>
      </query>
    </iq>
```

c. The user's server SHALL also deliver the available presence stanza received from each of the contact's available resources to each of the user's available resources.

2. **[Required]** Otherwise – that is, if the user does not exist, if the contact is not in the user's roster, or if the contact is in the user's roster with a subscription state other than those described in the foregoing check – then the user's server SHALL silently ignore the subscription approval stanza by not delivering it to the user, not modifying the user's roster, and not generating a roster push to the user's interested resources. [Section 3.1.6, RFC 6121]

   NOTE: If the account has no available resources when the inbound subscribed notification is received, a server MAY keep a record of the notification (ideally the complete presence stanza) and then deliver the notification when the account next has an available resource. This behavior provides more complete signaling to the user regarding the reasons for the roster change that occurred while the user was offline. [Section 3.1.6, RFC 6121]

5.7.3.13.2 Cancelling a Subscription

5.7.3.13.2.1 Rules for Client Generation of Subscription Cancellation

[Required] A client implementation SHALL be capable of sending a presence stanza of type "unsubscribed" in order to cancel a subscription that it has previously granted to a user. [Section 3.2.1, RFC 6121]

```
CC: <presence id='ij5b1v7g'
              to='robert@example2.dod.mil'
              type='unsubscribed'/>
```

5.7.3.13.2.2 Rules for Server Processing of Outbound Subscription Cancellation

[Required] Upon receiving the outbound subscription cancellation, the contact's server SHALL proceed as follows [Section 3.2.2, RFC 6121]:

1. If the user is hosted on the same server as the contact, then the server SHALL adhere to the rules specified in the next section in processing the subscription cancellation.

2. If the user is hosted on a remote server, the contact's server SHALL then route the stanza to that remote domain.

3. As mentioned, before locally delivering or remotely routing the stanza, the contact's server SHALL stamp the outbound subscription cancellation with the bare JID <localpart@domain> of the contact.

```
CS: <presence from='john@example1.dod.mil'
              id='ij5b1v7g'
              to='robert@example2.dod.mil'
              type='unsubscribed'/>
```

4. The contact's server then SHALL send a roster push with the updated roster item to all of the contact's interested resources, where the subscription state is now either "none" or "to". For added clarification, see Appendix A of RFC 6121.

5. The contact's server then SHALL send a presence stanza of type "unavailable" from all of the contact's online resources to the user.

```
CS: <presence from='john@example1.dod.mil/desktop client'
              id='i8bsg3h3'
              type='unavailable'/>
```

5.7.3.13.2.3 Rules for Server Processing of Inbound Subscription Cancellation

[Required] When the user's server receives the inbound subscription cancellation, it SHALL first check if the contact is in the user's roster with subscription='to' or subscription='both' (see Appendix A of RFC 6121).

1. If this check is successful, the user's server SHALL [Section 3.2.3, RFC 6121]:

   a. Deliver the inbound subscription cancellation to all of the user's interested resources. This SHALL occur before sending the roster push described in the next step.

   ```
   US: <presence from='john@example1.dod.mil'
                 id='ij5b1v7g'
                 to='robert@example2.dod.mil'
                 type='unsubscribed'/>
   ```

   b. Initiate a roster push to all of the user's interested resources, containing an updated roster item for the contact with the 'subscription' attribute set to a value of "none" (if the subscription state was "To" or "To + Pending In") or "from" (if the subscription state was "Both").

2. If the check (above) is not successful, that is, if the user does not exist, if the contact is not in the user's roster, or if the contact is in the user's roster with a subscription state other than those described in the foregoing check, then the user's server SHALL silently ignore the stanza by not delivering it to the user, not modifying the user's roster, and not generating a roster push to the user's interested resources. [Section 3.2.3, RFC 6121]

5.7.3.13.3 Unsubscribing

5.7.3.13.3.1 Rules for Client Unsubscribing

[Required] To unsubscribe from a contact's presence, the client SHALL send a presence stanza of type "unsubscribe". [Section 3.3.1, RFC 6121]

```
UC: <presence id='ul4bs71n'
                to='john@example.dod.mil'
                type='unsubscribe'/>
```

5.7.3.13.3.2 Rules for Server Processing of Outbound Unsubscribe

[Required] Upon receiving the outbound unsubscribe, the user's server SHALL proceed as follows [Section 3.3.2, RFC 6121]:

1. If the contact is hosted on the same server as the user, then the server SHALL adhere to the rules specified for Server Processing of Inbound Unsubscribe (see below).

2. If the contact is hosted on a remote server, the user's server SHALL then route the stanza to that remote domain.

3. The user's server then SHALL send a roster push with the updated roster item to all the user's interested resources, where the subscription state is now either "none" or "from" (see Appendix A of RFC 6121).

```
US: <iq id='h37h3u1bv402'
        to='robert@example2.dod.mil/desktop client'
        type='set'>
      <query xmlns='jabber:iq:roster'>
        <item jid='john@example1.dod.mil'
              subscription='none'/>
     </query>
    </iq>
```

5.7.3.13.3.3 Rules for Server Processing of Inbound Unsubscribe

[Required] When the contact's server receives the unsubscribe notification, it SHALL first check if the user is in the contact's roster with subscription='from' or subscription='both' (i.e., a subscription state of "From", "From + Pending Out", or "Both"; see Appendix A of RFC 6121).

1. If this check is successful, the contact's server SHALL [Section 3.3.3, RFC 6121]:

   a. Deliver the inbound unsubscribe to all of the contact's interested resources. This SHALL occur before sending the roster push described in the next step.

   b. Initiate a roster push to all of the contact's interested resources, containing an updated roster item for the contact with the 'subscription' attribute set to a value of "none" (if the subscription state was "From" or "From + Pending Out") or "to" (if the subscription state was "Both").

   c. Generate an outbound presence stanza of type "unavailable" from each of the contact's available resources to the user.

2. If the check (above) is not successful, that is, if the contact does not exist, if the user is not in the contact's roster, or if the user is in the contact's roster with a subscription state other than those described in the foregoing check, then the contact's server SHALL silently ignore the stanza by not delivering it to the contact, not modifying the contact's roster, and not generating a roster push to the contact's interested resources. [Section 3.3.3, RFC 6121]

5.7.3.14 Exchanging Presence Information

In XMPP, presence information is exchanged using <presence/> stanzas as defined in RFC 6121. A client controlled by a user sends presence information to its home server and the home server in turn propagates that information to all of the user's contacts who have a subscription to that user's presence. [Section 4.1, RFC 6121]

5.7.3.14.1 Initial Presence

5.7.3.14.1.1 Client Generation of Initial Presence

[Required] After completing the mandatory-to-negotiate stream features and retrieving a roster, a client implementation SHALL signal its availability for communication by sending initial presence to its server, i.e., a presence stanza with no 'to' address and no 'type' attribute. [Section 4.2.1, RFC 6121]

UC: <presence/>

NOTE: The initial presence stanza may contain the <priority/> element, the <show/> element, and one or more instances of the <status/> element. [Section 4.2, RFC 6121]

5.7.3.14.1.2 Server Processing of Outbound Initial Presence

1. **[Required]** Upon receiving initial presence from a client, the user's server SHALL send the initial presence stanza from the full JID <user@domain/resource> of the user to all contacts that are subscribed to the user's presence. [Section 4.2.2, RFC 6121]

   ```
   US: <presence from='user@domain/resourcepart'
                 to='contact@domain'/>
   ```

2. **[Required]** The user's server SHALL also broadcast initial presence from the user's newly available resource to all of the user's available resources (including the resource that generated the presence notification in the first place). [Section 4.2.2, RFC 6121]

3. **[Required]** In the absence of presence information about the user's contacts, the user's server SHALL also send presence probes to the user's contacts on behalf of the user (see Section 5.7.3.14.2, Presence Probes). [Section 4.2.2, RFC 6121]

5.7.3.14.1.3 Server Processing of Inbound Initial Presence

[Required] Upon receiving presence from the user, the contact's server SHALL deliver the user's presence stanza to all of the contact's available resources. [Section 4.2.3, RFC 6121]

5.7.3.14.1.4 Client Processing of Inbound Initial Presence

[Required] When the contact's client receives presence from the user, it SHALL proceed as follows [Section 4.2.4, RFC 6121]:

1. If the user is in the contact's roster, the client SHALL display the presence information in an appropriate roster interface.

2. If the user is not in the contact's roster, the client SHALL ignore the presence information and not display it to the contact.

5.7.3.14.2 Presence Probes

A presence probe is a request for a contact's current presence information, sent on behalf of a user by the user's server; syntactically it is a presence stanza whose 'type' attribute has a value of “probe”. In the context of presence subscriptions, the value of the ‘from’ address SHALL be the bare JID of the subscribed user and the value of the ‘to’ address SHALL be the bare JID of the contact to which the user is subscribed, since presence subscriptions are based on the bare JID. [Section 4.3, RFC 6121]

5.7.3.14.2.1 Server Generation of Outbound Presence Probe

1. **[Required]** To discover the availability of a user’s contact, the user’s server SHALL be capable of sending a presence probe from the bare JID <user@domain> of the user to the bare JID <contact@domain> of the contact. [Section 4.3.1, RFC 6121]

```
US:  <presence from='john@example1.dod.mil'
               id='ign291v5'
               to='robert@example2.dod.mil'
               type='probe'/>
```

2. **[Required]** The server SHALL NOT send a probe to a contact if the user is not subscribed to the contact's presence (i.e., if the contact is not in the user’s roster with the ‘subscription’ attribute set to a value of “to” or “both”). [Section 4.3.1, RFC 6121]

   NOTE: The user’s server SHOULD send a presence probe whenever the user starts a new presence session by sending initial presence. However, the server MAY choose not to send the probe at that point if it has what it deems to be reliable and up-to-date presence information about the user’s contacts (e.g., because the user has another available resource or because the user briefly logged off and on before the new presence session began). In addition, a server MAY periodically send a presence probe to a contact if it has not received presence information or other traffic from the contact in some configurable amount of time; this can help to prevent “ghost” contacts who appear to be online but in fact are not. [Section 4.3.1, RFC 6121]

   NOTE: Naturally, the user’s server does not need to send a presence probe to a contact if the contact’s account resides on the same server as the user, since the server possesses the contact’s information locally. [Section 4.3.1, RFC 6121]

5.7.3.14.2.2 Server Processing of Inbound Presence Probe

[Required] Upon receiving a presence probe to the contact’s bare JID from the user’s server on behalf of the user, the contact’s server SHALL reply as follows [Section 4.3.2, RFC 6121]:

1. If the contact account does not exist or the user is in the contact’s roster with a subscription state other than “From”, “From + Pending Out”, or “Both” (as defined under Appendix A of RFC 6121), then the contact’s server SHALL return a presence stanza of type

"unsubscribed" in response to the presence probe. Here the 'from' address SHALL be the bare JID of the contact, since specifying a full JID would constitute a presence leak as described in RFC 6120.

```
CS:  <presence from='mike@example2.dod.mil'
               id='xv291f38'
               to='john@example1.dod.mil'
          type='unsubscribed'/>2.
```

2. Else, if the contact has no available resources, then the server SHALL reply to the presence probe by sending to the user a presence stanza of type "unavailable".

3. Else, if the contact has at least one available resource, then the server SHALL reply to the presence probe by sending to the user the full XML of the last presence stanza with no 'to' attribute received by the server from each of the contact's available resources. Here the 'from' addresses are the full JIDs of each available resource.

```
CS:  <presence from='robert@example2.dod.mil/foo'
               id='hzf1v27k'
               to='john@example1.dod.mil'/>
```

5.7.3.14.3 Subsequent Presence Broadcasts

[Required] After sending initial presence, a client implementation SHALL be capable of updating its availability by sending a presence stanza with no 'to' address and no 'type' attribute. [Section 4.4.1, RFC 6121]

```
UC:  <presence>
       <show>away</show>
     </presence>
```

NOTE: This presence update MAY contain the <priority/> element, the <show/> element, and one or more instances of the <status/> element.

5.7.3.14.3.1 Server Processing of Outbound Presence

1. **[Required]** Upon receiving a presence stanza expressing updated availability, the user's server SHALL broadcast the full XML of that presence stanza to the contacts who meet all of the following criteria [Section 4.4.2, RFC 6121]:

   a. The contact is in the user's roster with a subscription type of "from" or "both".

b. The last presence stanza received from the contact during the user's presence session was NOT of type "unsubscribe".

NOTE: As an optimization, if the subscription type is "both", then the server SHOULD send subsequent presence notifications to a contact only if the contact is online according to the user's server. [Section 4.4.2, RFC 6121]

2. **[Required]** The user's server SHALL also send the presence stanza to all of the user's available resources (including the resource that generated the presence notification in the first place). [Section 4.4.2, RFC 6121]

5.7.3.14.3.2 Server Processing of Inbound Presence

[Required] Upon receiving presence from the user, the contact's server SHALL deliver the user's presence stanza to all of the contact's available resources. [Section 4.4.3, RFC 6121]

5.7.3.14.3.3 Client Processing of Inbound Presence

[Required] From the perspective of the contact's client, there is no significant difference between initial presence broadcast and subsequent presence broadcast, so the contact's client SHALL follow the rules for processing of inbound presence defined under Section 5.7.3.14.1.4, Client Processing of Inbound Initial Presence. [Section 4.4.4, RFC 6121]

5.7.3.14.4 Unavailable Presence

5.7.3.14.4.1 Client Generation of Unavailable Presence

[Required] Before ending its presence session with a server, the user's client SHALL gracefully become unavailable by sending unavailable presence, i.e., a presence stanza that possesses no 'to' attribute and that possesses a 'type' attribute whose value is "unavailable". The unavailable presence stanza SHALL NOT contain the <priority/> element or the <show/> element, since these elements apply only to available resources. [Section 4.5.1, RFC 6121]

UC: <presence type='unavailable'/>

NOTE: Optionally, the unavailable presence stanza MAY contain one or more <status/> elements specifying the reason why the user is no longer available.

5.7.3.14.4.2 Server Processing of Outbound Unavailable Presence

1. **[Required]** The user's server SHALL NOT depend on receiving unavailable presence from an available resource, since the resource can become unavailable ungracefully (e.g.,

the resource can be timed out by the server because of inactivity). [Section 4.5.2, RFC 6121]

2. **[Required]** If an available resource becomes unavailable for any reason (either gracefully or ungracefully), the user's server SHALL broadcast unavailable presence to all contacts that meet all of the following criteria [Section 4.5.2, RFC 6121]:

    a. The contact is in the user's roster with a subscription type of "from" or "both".

    b. The last presence stanza received from the contact during the user's presence session was not of type "error" or "unsubscribe".

3. **[Required]** If the unavailable notification was gracefully received from the client, then the server SHALL broadcast the full XML of the presence stanza. [Section 4.5.2, RFC 6121]

4. **[Required]** The user's server SHALL also send the unavailable notification to all of the user's available resources (including the resource that generated the presence notification in the first place). [Section 4.5.2, RFC 6121]

5. **[Required]** If the server detects that the user has gone offline ungracefully, then the server SHALL generate the unavailable presence broadcast on the user's behalf. [Section 4.5.2, RFC 6121]

5.7.3.14.4.3 Server Processing of Inbound Unavailable Presence

[Required] Upon receiving an unavailable notification from the user, the contact's server SHALL deliver the user's presence stanza to all of the contact's available resources. [Section 4.5.3, RFC 6121]

5.7.3.14.4.4 Client Processing of Inbound Unavailable Presence

[Required] From the perspective of the contact's client, there is no significant difference between initial presence broadcast and unavailable presence broadcast, so the contact's client SHALL follow the rules for processing of inbound presence defined under Section 5.7.3.14.1.4, Client Processing of Inbound Initial Presence. [Section 4.5.4, RFC 6121]

5.7.3.14.5 Presence Syntax

5.7.3.14.5.1 Show Element

[Required] To specify a particular availability sub-state, a client implementation SHALL support the <show/> element within a presence stanza. A presence stanza SHALL NOT contain more than one <show/> element. The XML character data of the <show/> element is not human-readable. The XML character data SHALL be one of the following [Section 4.7.2.1, RFC 6121]:

- away – The entity or resource is temporarily away.
- chat – The entity or resource is actively interested in chatting.
- dnd – The entity or resource is busy (dnd = "Do Not Disturb").
- xa – The entity or resource is away for an extended period (xa = "eXtended Away").

NOTE: If no <show/> element is provided, the entity is assumed to be online and available. [Section 4.7.2.1, RFC 6121]

NOTE: While support for this feature is required, the use of this feature is optional.

5.7.3.14.5.2 Status Element

To convey human-readable XML character data specifying a natural-language description of an entity's availability, the client SHALL support the <status/> element within a presence stanza. It is normally used in conjunction with the show element to provide a detailed description of an availability state (e.g., "In a meeting") when the presence stanza has no 'type' attribute. There are no attributes defined for the <status/> element, with the exception of the 'xml:lang' attribute. [Section 4.7.2.2, RFC 6121]

```
<presence from='john.smith@chat1.dod.mil/office'
            xml:lang='en'>
     <show>dnd</show>
     <status>In a meeting</status>
</presence>
```

NOTE: A presence stanza of type "unavailable" MAY also include a <status/> element to provide detailed information about why the entity is going offline.

NOTE: While support for this feature is required, the use of this feature is optional.

5.7.3.14.5.3 Priority Element

NOTE: The OPTIONAL <priority/> element contains non-human-readable XML character data that specifies the priority level of the resource. The value SHALL be an integer between -128 and +127. [Section 4.7.2.3, RFC 6121]

```
<presence xml:lang='en'>
    <show>dnd</show>
    <status>In Meeting</status>
   <priority>1</priority>
</presence>
```

If no priority is provided, the processing server or client SHOULD consider the priority to be zero (“0”).

5.7.3.15 Exchanging Messages

After a client has established and secured a stream with its home server, the next step, as discussed above, is to bind a specific resource to the stream. Once the client has completed the resource binding step, the client may generate and exchange an unlimited number of stanzas. One such stanza that can be exchanged is <message/>. As discussed in RFC 6121, a <message/> stanza is used to “push” information to another entity.

5.7.3.15.1 One-to-One Chat Sessions

One-to-One Chat permits a user to engage in a near real-time, text-based conversation with another user. In XMPP, this text-based conversation is enabled through the exchange of <message/> stanzas. As discussed in Section 5 of RFC 6121, the two parties will typically exchange a number of messages in relatively rapid succession within a relatively brief period. [Section 5.1, RFC 6121]

1. **[Required]** When a user’s client is engaged in a chat session with a contact, the user's client SHALL send a message of type “chat” and the contact’s client SHALL preserve that message type in subsequent replies. [Section 5.1, RFC 6121]

2. **[Required]** The user’s client SHALL be capable of including a <thread/> element with its initial message, which the contact's client SHALL also preserve during the life of the chat session. The primary use of the XMPP <thread/> element is to uniquely identify a conversation thread or “chat session” between two entities instantiated by <message/> stanzas of type ‘chat’. [Section 5.1, RFC 6121]

3. **[Required]** The user's client SHALL address the initial message in a chat session to the bare JID of the contact (i.e., <contact@domain>). Until and unless the user's client receives a reply from the contact, it SHALL continue sending any further messages to the contact's bare JID. Once the user's client receives a reply from the contact's full JID, it SHALL address its subsequent messages to the contact's full JID as provided in the 'from' address of the contact's replies. [Section 5.1, RFC 6121]

4. **[Required]** The contact's client SHALL address its subsequent replies to the user's full JID <user@domain/resource> as provided in the 'from' address of the initial message. [Section 5.1, RFC 6121]

5.7.3.15.2 Message Stanza Syntax

5.7.3.15.2.1 To Attribute

[Required] An instant messaging client SHALL specify the intended recipient for a message stanza by providing the JID of the intended recipient in the 'to' attribute of the <message/> stanza. [Section 5.2.1, RFC 6121]

5.7.3.15.2.2 Type Attribute

1. **[Required]** An instant messaging client SHALL support all of the following message types [Section 5.2.2, RFC 6121]:

   a. chat – The value "chat" indicates that the message is sent in the context of a one-to-one chat session. Typically, a receiving client will present/display messages of type "chat" in an interface that enables one-to-one chat between the two parties, including an appropriate conversation history.

   b. error – The value "error" indicates that the message is generated by an entity that experienced an error in processing a message received from another entity.
   NOTE: A client that receives a message of type "error" SHOULD present an appropriate interface informing the sender of the nature of the error.

   c. groupchat – The value "groupchat" indicates that the message is sent in the context of a multiuser chat environment. Typically, a receiving client will present a message of type "groupchat" in an interface that enables many-to-many chat between the parties.

   d. normal – The value "normal" indicates that the message is a standalone message that is sent outside the context of a one-to-one conversation or groupchat, and to which it is expected that the recipient will reply. Typically, a receiving client will present a message of type “normal” in an interface that enables the recipient to reply, but without a conversation history. The default value of the ‘type’ attribute is "normal".

NOTE: Support for the following message type is defined as recommended.

e. headline – The value “headline” indicates that the message provides an alert, a notification, or other information to which no reply is expected (e.g., news headlines, sports updates, near-real-time market data, and syndicated content). Because no reply to the message is expected, typically a receiving client will present a message of type “headline” in an interface that appropriately differentiates the message from standalone messages, chat messages, or groupchat messages (e.g., by not providing the recipient with the ability to reply).

2. **[Required]** If an application receives a message with no ‘type’ attribute or the application does not understand the value of the ‘type’ attribute provided, it SHALL consider the message to be of type “normal” (i.e., “normal” is the default). [Section 5.2.2, RFC 6121]

5.7.3.15.2.3 Body Element

[Required] A client SHALL be capable of populating a <message/> stanza with the <body/> element. The <body/> element contains human-readable XML character data that specifies the textual content of the message.

NOTE: While support for this feature is required, the use of this feature is optional. This child element is normally included in a message stanza. [Section 5.2.3, RFC 6121]

NOTE: There are no attributes defined for the <body/> element, with the exception of the ‘xml:lang’ attribute. Multiple instances of the <body/> element MAY be included in a message stanza, but only if each instance possesses an ‘xml:lang’ attribute with a distinct language value. [Section 5.2.3, RFC 6121]

5.7.3.16 Conformance Requirements in RFC 6120 and RFC 6121

Section 15 of RFC 6120 and Section 13 of RFC 6121 describe a protocol feature set that summarizes the conformance requirements associated with these two specifications. In the event of a discrepancy between Section 15 of RFC 6121 or Section 13 of RFC 6121 and this section of the UCR, the explicit requirements defined in this section of the UCR take precedence.

5.7.3.17 XMPP Extensions

The documents referenced in this section represent extensions to the XMPP baseline specifications (i.e., RFC 6120 and RFC 6121). Through an open standards process, the XMPP Standards Foundation (XSF) develops extensions to XMPP. These extensions are published by the XSF as XMPP Extension Protocols (XEPs) series documents at http://xmpp.org/. While the majority of XMPP extensions are defined in the XEP series documents, other important related specifications/extensions are defined by the XMPP Working Group at the IETF. These XMPP extensions address functionality or enable innovative features that are not addressed in the core XMPP specifications.

The protocol specifications referenced within Table 5.7.3-2, DoD XMPP Protocol Suite, constitute a mandatory protocol suite (i.e., for the purpose of compliance testing and certification; support for these extensions is defined as REQUIRED). Regarding the specifications defined in Table 5.7.3-2, DoD XMPP Protocol Suite, client and server implementations SHALL comply with all requirements defined as “MUST”, “SHALL”, “REQUIRED”, “MUST NOT”, “SHALL NOT”. It is also expected that vendors will likewise implement requirements defined as “SHOULD” or “SHOULD NOT” except where there may exist valid reasons in particular circumstances to ignore a particular requirement.

NOTE: Some of the protocol specifications referenced in Table 5.7.3-2, DoD XMPP Protocol Suite, have their own dependencies.

Table 5.7.3-2. DoD XMPP Protocol Suite

REFERENCE	XMPP SERVER	XMPP CLIENT	XMPP GATEWAYS
XEP-0045: *Multi-User Chat*	✓	✓	Conditional
XEP-0030: *Service Discovery*	✓	✓	✓
XEP-0085: *Chat State Notifications*	N/A	✓	✓
RFC 4422 – Appendix A *SASL EXTERNAL Mechanism**	✓		✓
XEP-0004: *Data Forms*	✓	✓	Conditional
XEP-0077: *In-Band Registration***	✓	✓	Conditional
XEP-0082: *XMPP Date and Time Profiles*	✓	✓	Conditional
XEP-0068: *Field Standardization for Data Forms*	✓	✓	Conditional
* See XEP-0178: Best Practices for Use of SASL EXTERNAL with Certificates ** The use of In-Band Registration is restricted to the use case where a user is attempting to register with a moderated room in the context of a Multi-User Chat service.			

5.7.3.17.1 Elevated/Clarified Requirements

To better enable multivendor interoperability, to facilitate full feature functionality, and to address specific security requirements, some of the requirements defined as "SHOULD", "RECOMMENDED", "SHOULD NOT", "NOT RECOMMENDED", "MAY", or "OPTIONAL" in the above XMPP extensions have been redefined in this specification to reflect requirement levels associated with the following terminology: "MUST", "SHALL", "REQUIRED", "MUST NOT", or "SHALL NOT". These elevated requirements are explicitly defined in Table 5.7.3-3. Also, where there may be some degree of ambiguity in a commercial standard regarding whether or not support for a particular capability or feature is REQUIRED, Table 5.7.3-3, Elevated/Clarified Requirements, adds explicit clarification.

Table 5.7.3-3. Elevated/Clarified Requirements

REFERENCE DOCUMENT	REFERENCE DOCUMENT SECTION	REQUIREMENT*
XEP-0045 *Multi-User Chat*	5.1	Implementations SHALL provide support for the 'Visitor' role.
XEP-0045 *Multi-User Chat*	5.2	Implementations SHALL provide support for the 'Admin', 'Member', and 'Outcast' affiliation.
XEP-0045 *Multi-User Chat*	6.1, 6.2, and 6.3	Implementations SHALL support the following capabilities (as defined in Sections 6.1, 6.2, and 6.3): 1. Discovering Component Support for MUC 2. Discovering Rooms 3. Querying for Room Information
XEP-0045 *Multi-User Chat*	3, 4.2, 7.1.5, 7.1.6, 7.1.7, and 7.1.8	Implementations SHALL support the following room types: 1. Both Persistent or Temporary 2. Public 3. Non-Anonymous 4. Password-Protected and Unsecured 5. Both Members-Only and Open 6. Moderated and Un-moderated
XEP-0045 *Multi-User Chat*	7.1.15	Implementations SHALL support the sending of Discussion History to a new occupant (as defined in Sections 7.1.15). NOTE: "Whether such history is sent, and how many messages comprise the history, shall be determined by the chat service implementation or specific deployment."

REFERENCE DOCUMENT	**REFERENCE DOCUMENT SECTION**	**REQUIREMENT***
XEP-0045 *Multi-User Chat*	7.1, 7.2, 7.4, 7.5, 7.6, 7.8, 7.9, 7.10, and 7.13	Implementations SHALL support a user's ability to: 1. Enter a Room 2. Exit a Room 3. Change Availability Status 4. Invite Another User to a Room 5. Convert a One-to-One Chat into a Multi-User Conference 6. Send a Private Message 7. Send a Message to All Occupants 8. Register with a Room 9. Request Voice
XEP-0045 *Multi-User Chat*	8.1 through 8.6	Implementations SHALL support the ability of a Moderator to perform the following privileges: 1. Modify the subject 2. Kick a participant or visitor from the room 3. Grant or revoke voice in a moderated room 4. Modify the list of occupants who have voice in a moderated room
XEP-0045 *Multi-User Chat*	9.1 through 9.9	Implementations SHALL support the ability of an Admin to perform the following privileges: 1. Ban a user from the room 2. Modify the list of users who are banned from the room 3. Grant or revoke membership 4. Modify the member list 5. Grant or revoke moderator privileges 6. Modify the list of moderators 7. Approve Registration Requests
XEP-0045 *Multi-User Chat*	10.1 and 10.2	Implementations SHALL support the ability of an Owner to create a room and to change defining room configuration settings (as defined in Section 10.1 and 10.2)
XEP-0045 *Multi-User Chat*	10.3 through 10.9	Implementations SHALL support the ability of an Owner to perform the following privileges (as defined in Section 10): 1. Grant or revoke ownership privileges 2. Modify the owner list 3. Grant or revoke administrative privileges 4. Modify the Admin list 5. Destroy a room

REFERENCE DOCUMENT	REFERENCE DOCUMENT SECTION	REQUIREMENT*
XEP-0030 *Service Discovery*	4 and 3	Implementation SHALL provide support for: 1. Discovering information about an entity as defined in Section 3 [XEP-030] 2. Discovering the items associated with an entity as defined in Section 4 [XEP-030]

NOTE: Table 5.7.3-3, Elevated/Clarified Requirements, ONLY addresses functionality where the associated requirement level has been elevated (e.g., from a “SHOULD” to a “SHALL”) or where there was a need to explicitly clarify whether support for a particular capability or feature is REQUIRED.

5.7.3.18 XML Usage

[Required] XMPP client and server implementations SHALL comply with the mandatory requirements defined in Section 11 of RFC 6120.

5.7.3.19 DiffServ Code Point (DSCP) Requirements

[Required] XMPP client and server implementations shall class mark XMPP traffic consistent with the code point values defined for ROUTINE Low-Latency Data as per Table 5.3.3-1 (DSCP Assignments).