

Network Working Group	P. Saint-Andre	
Internet-Draft	Cisco	
Obsoletes: 3920 (if approved)	March 09, 2009	
Intended status: Standards Track		
Expires: September 10, 2009		

[TOC](#)

Extensible Messaging and Presence Protocol (XMPP): Core draft-saintandre-rfc3920bis-09

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on September 10, 2009.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (<http://trustee.ietf.org/license-info>). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This document defines the core features of the Extensible Messaging and Presence Protocol (XMPP), a technology for streaming Extensible Markup Language (XML) elements for the purpose of exchanging structured information in close to real time between any two or more network-aware

entities. XMPP provides a generalized, extensible framework for incrementally exchanging XML data, upon which a variety of applications can be built. The framework includes methods for stream setup and teardown, channel encryption, authentication of a client to a server and of one server to another server, and primitives for push-style messages, publication of network availability information ("presence"), and request-response interactions. This document also specifies the format for XMPP addresses, which are fully internationalizable. This document obsoletes RFC 3920.

Table of Contents

- [1. Introduction](#)
 - [1.1. Overview](#)
 - [1.2. Functional Summary](#)
 - [1.3. Conventions](#)
 - [1.4. Acknowledgements](#)
 - [1.5. Discussion Venue](#)
- [2. Architecture](#)
 - [2.1. Overview](#)
 - [2.2. Server](#)
 - [2.3. Client](#)
 - [2.4. Network](#)
- [3. Addresses](#)
 - [3.1. Overview](#)
 - [3.2. Domain Identifier](#)
 - [3.3. Node Identifier](#)
 - [3.4. Resource Identifier](#)
 - [3.5. Determination of Addresses](#)
- [4. TCP Binding](#)
 - [4.1. Scope](#)
 - [4.2. Hostname Resolution](#)
 - [4.3. Client-to-Server Communication](#)
 - [4.4. Server-to-Server Communication](#)
 - [4.5. Reconnection](#)
 - [4.6. Other Bindings](#)
- [5. XML Streams](#)
 - [5.1. Overview](#)
 - [5.2. Stream Security](#)
 - [5.3. Stream Attributes](#)
 - [5.3.1. from](#)
 - [5.3.2. to](#)
 - [5.3.3. id](#)
 - [5.3.4. xml:lang](#)
 - [5.3.5. version](#)
 - [5.3.6. Summary of Stream Attributes](#)
 - [5.4. Namespace Declarations](#)

- [5.5.](#) Stream Features
- [5.6.](#) Restarts During Stream Negotiation
- [5.7.](#) Closing a Stream
 - [5.7.1.](#) With Stream Error
 - [5.7.2.](#) Without Stream Error
 - [5.7.3.](#) Handling of Idle Streams
- [5.8.](#) Stream Errors
 - [5.8.1.](#) Rules
 - [5.8.1.1.](#) Stream Errors Are Unrecoverable
 - [5.8.1.2.](#) Stream Errors Can Occur During Setup
 - [5.8.1.3.](#) Stream Errors When the Host is Unspecified or

Unknown

- [5.8.2.](#) Syntax
 - [5.8.3.](#) Defined Stream Error Conditions
 - [5.8.3.1.](#) bad-format
 - [5.8.3.2.](#) bad-namespace-prefix
 - [5.8.3.3.](#) conflict
 - [5.8.3.4.](#) connection-timeout
 - [5.8.3.5.](#) host-gone
 - [5.8.3.6.](#) host-unknown
 - [5.8.3.7.](#) improper-addressing
 - [5.8.3.8.](#) internal-server-error
 - [5.8.3.9.](#) invalid-from
 - [5.8.3.10.](#) invalid-id
 - [5.8.3.11.](#) invalid-namespace
 - [5.8.3.12.](#) invalid-xml
 - [5.8.3.13.](#) not-authorized
 - [5.8.3.14.](#) policy-violation
 - [5.8.3.15.](#) remote-connection-failed
 - [5.8.3.16.](#) resource-constraint
 - [5.8.3.17.](#) restricted-xml
 - [5.8.3.18.](#) see-other-host
 - [5.8.3.19.](#) system-shutdown
 - [5.8.3.20.](#) undefined-condition
 - [5.8.3.21.](#) unsupported-encoding
 - [5.8.3.22.](#) unsupported-stanza-type
 - [5.8.3.23.](#) unsupported-version
 - [5.8.3.24.](#) xml-not-well-formed
 - [5.8.4.](#) Application-Specific Conditions
- [5.9.](#) Simplified Stream Examples
- [6.](#) STARTTLS Negotiation
- [6.1.](#) Overview
 - [6.2.](#) Rules
 - [6.2.1.](#) Data Formatting
 - [6.2.2.](#) Order of Negotiation
 - [6.3.](#) Process
 - [6.3.1.](#) Exchange of Stream Headers and Stream Features
 - [6.3.2.](#) Initiation of STARTTLS Negotiation
 - [6.3.2.1.](#) STARTTLS Command

- [6.3.2.2.](#) Failure Case
 - [6.3.2.3.](#) Proceed Case
 - [6.3.3.](#) TLS Negotiation
 - [6.3.3.1.](#) Rules
 - [6.3.3.2.](#) TLS Failure
 - [6.3.3.3.](#) TLS Success
- [7.](#) SASL Negotiation
 - [7.1.](#) Overview
 - [7.2.](#) Rules
 - [7.2.1.](#) Mechanism Preferences
 - [7.2.2.](#) Mechanism Offers
 - [7.2.3.](#) Data Formatting
 - [7.2.4.](#) Security Layers
 - [7.2.5.](#) Simple Usernames
 - [7.2.6.](#) Authorization Identities
 - [7.2.7.](#) Realms
 - [7.2.8.](#) Round Trips
 - [7.3.](#) Process
 - [7.3.1.](#) Exchange of Stream Headers and Stream Features
 - [7.3.2.](#) Initiation
 - [7.3.3.](#) Challenge-Response Sequence
 - [7.3.4.](#) Abort
 - [7.3.5.](#) Failure
 - [7.3.6.](#) Success
 - [7.4.](#) SASL Errors
 - [7.4.1.](#) aborted
 - [7.4.2.](#) account-disabled
 - [7.4.3.](#) credentials-expired
 - [7.4.4.](#) encryption-required
 - [7.4.5.](#) incorrect-encoding
 - [7.4.6.](#) invalid-authzid
 - [7.4.7.](#) invalid-mechanism
 - [7.4.8.](#) malformed-request
 - [7.4.9.](#) mechanism-too-weak
 - [7.4.10.](#) not-authorized
 - [7.4.11.](#) temporary-auth-failure
 - [7.4.12.](#) transition-needed
 - [7.5.](#) SASL Definition
- [8.](#) Resource Binding
 - [8.1.](#) Overview
 - [8.2.](#) Advertising Support
 - [8.3.](#) Generation of Resource Identifiers
 - [8.4.](#) Server-Generated Resource Identifier
 - [8.4.1.](#) Success Case
 - [8.4.2.](#) Error Cases
 - [8.4.2.1.](#) Resource Constraint
 - [8.4.2.2.](#) Not Allowed
 - [8.5.](#) Client-Submitted Resource Identifier
 - [8.5.1.](#) Success Case

- [8.5.2. Error Cases](#)
 - [8.5.2.1. Bad Request](#)
 - [8.5.2.2. Conflict](#)
 - [8.5.3. Retries](#)
 - [8.6. Binding Multiple Resources](#)
 - [8.6.1. Support](#)
 - [8.6.2. Binding an Additional Resource](#)
 - [8.6.3. Unbinding a Resource](#)
 - [8.6.3.1. Success Case](#)
 - [8.6.3.2. Error Cases](#)
 - [8.6.4. From Addresses](#)
- [9. XML Stanzas](#)
 - [9.1. Common Attributes](#)
 - [9.1.1. to](#)
 - [9.1.1.1. Client-to-Server Streams](#)
 - [9.1.1.2. Server-to-Server Streams](#)
 - [9.1.2. from](#)
 - [9.1.2.1. Client-to-Server Streams](#)
 - [9.1.2.2. Server-to-Server Streams](#)
 - [9.1.3. id](#)
 - [9.1.4. type](#)
 - [9.1.5. xml:lang](#)
 - [9.2. Basic Semantics](#)
 - [9.2.1. Message Semantics](#)
 - [9.2.2. Presence Semantics](#)
 - [9.2.3. IQ Semantics](#)
 - [9.3. Stanza Errors](#)
 - [9.3.1. Rules](#)
 - [9.3.2. Syntax](#)
 - [9.3.3. Defined Conditions](#)
 - [9.3.3.1. bad-request](#)
 - [9.3.3.2. conflict](#)
 - [9.3.3.3. feature-not-implemented](#)
 - [9.3.3.4. forbidden](#)
 - [9.3.3.5. gone](#)
 - [9.3.3.6. internal-server-error](#)
 - [9.3.3.7. item-not-found](#)
 - [9.3.3.8. jid-malformed](#)
 - [9.3.3.9. not-acceptable](#)
 - [9.3.3.10. not-allowed](#)
 - [9.3.3.11. not-authorized](#)
 - [9.3.3.12. not-modified](#)
 - [9.3.3.13. payment-required](#)
 - [9.3.3.14. recipient-unavailable](#)
 - [9.3.3.15. redirect](#)
 - [9.3.3.16. registration-required](#)
 - [9.3.3.17. remote-server-not-found](#)
 - [9.3.3.18. remote-server-timeout](#)
 - [9.3.3.19. resource-constraint](#)

- [9.3.3.20.](#) service-unavailable
 - [9.3.3.21.](#) subscription-required
 - [9.3.3.22.](#) undefined-condition
 - [9.3.3.23.](#) unexpected-request
 - [9.3.3.24.](#) unknown-sender
 - [9.3.4.](#) Application-Specific Conditions
- [9.4.](#) Extended Content
- [9.5.](#) Stanza Size
- [10.](#) Examples
 - [10.1.](#) Client-to-Server
 - [10.1.1.](#) TLS
 - [10.1.2.](#) SASL
 - [10.1.3.](#) Resource Binding
 - [10.1.4.](#) Stanza Exchange
 - [10.1.5.](#) Close
 - [10.2.](#) Server-to-Server Examples
 - [10.2.1.](#) TLS
 - [10.2.2.](#) SASL
 - [10.2.3.](#) Stanza Exchange
 - [10.2.4.](#) Close
- [11.](#) Server Rules for Processing XML Stanzas
 - [11.1.](#) No 'to' Address
 - [11.1.1.](#) Overview
 - [11.1.2.](#) Message
 - [11.1.3.](#) Presence
 - [11.1.4.](#) IQ
 - [11.2.](#) Local Domain
 - [11.2.1.](#) Mere Domain
 - [11.2.2.](#) Domain with Resource
 - [11.2.3.](#) Node at Domain
 - [11.2.3.1.](#) No Such User
 - [11.2.3.2.](#) Bare JID
 - [11.2.3.3.](#) Full JID
 - [11.3.](#) Foreign Domain
 - [11.3.1.](#) Existing Stream
 - [11.3.2.](#) No Existing Stream
 - [11.3.3.](#) Error Handling
- [12.](#) XML Usage
 - [12.1.](#) Restrictions
 - [12.2.](#) XML Namespace Names and Prefixes
 - [12.2.1.](#) Streams Namespace
 - [12.2.2.](#) Default Namespace
 - [12.2.3.](#) Extended Namespaces
 - [12.3.](#) Well-Formedness
 - [12.4.](#) Validation
 - [12.5.](#) Inclusion of Text Declaration
 - [12.6.](#) Character Encoding
 - [12.7.](#) Whitespace
 - [12.8.](#) XML Versions

- [13.](#) Compliance Requirements
 - [13.1.](#) Servers
 - [13.2.](#) Clients
- [14.](#) Internationalization Considerations
- [15.](#) Security Considerations
 - [15.1.](#) High Security
 - [15.2.](#) Certificates
 - [15.2.1.](#) Certificate Generation
 - [15.2.1.1.](#) Server Certificates
 - [15.2.1.2.](#) Client Certificates
 - [15.2.1.3.](#) ASN.1 Object Identifier
 - [15.2.2.](#) Certificate Validation
 - [15.2.2.1.](#) Server-to-Server Streams
 - [15.2.2.2.](#) Client-to-Server Streams
 - [15.2.2.3.](#) Use of Certificates in XMPP Extensions
 - [15.3.](#) Client-to-Server Communication
 - [15.4.](#) Server-to-Server Communication
 - [15.5.](#) Order of Layers
 - [15.6.](#) Mandatory-to-Implement Technologies
 - [15.7.](#) SASL Downgrade Attacks
 - [15.8.](#) Lack of SASL Channel Binding to TLS
 - [15.9.](#) Use of base64 in SASL
 - [15.10.](#) Stringprep Profiles
 - [15.11.](#) Address Spoofing
 - [15.11.1.](#) Address Forging
 - [15.11.2.](#) Address Mimicking
 - [15.12.](#) Firewalls
 - [15.13.](#) Denial of Service
 - [15.14.](#) Presence Leaks
 - [15.15.](#) Directory Harvesting
- [16.](#) IANA Considerations
 - [16.1.](#) XML Namespace Name for TLS Data
 - [16.2.](#) XML Namespace Name for SASL Data
 - [16.3.](#) XML Namespace Name for Stream Errors
 - [16.4.](#) XML Namespace Name for Resource Binding
 - [16.5.](#) XML Namespace Name for Stanza Errors
 - [16.6.](#) Nodeprep Profile of Stringprep
 - [16.7.](#) Resourceprep Profile of Stringprep
 - [16.8.](#) GSSAPI Service Name
 - [16.9.](#) Port Numbers
- [17.](#) References
 - [17.1.](#) Normative References
 - [17.2.](#) Informative References
- [Appendix A.](#) Nodeprep
 - [A.1.](#) Introduction
 - [A.2.](#) Character Repertoire
 - [A.3.](#) Mapping
 - [A.4.](#) Normalization
 - [A.5.](#) Prohibited Output

A.6.	Bidirectional Characters
A.7.	Notes
Appendix B.	Resourceprep
B.1.	Introduction
B.2.	Character Repertoire
B.3.	Mapping
B.4.	Normalization
B.5.	Prohibited Output
B.6.	Bidirectional Characters
Appendix C.	XML Schemas
C.1.	Streams Namespace
C.2.	Stream Error Namespace
C.3.	STARTTLS Namespace
C.4.	SASL Namespace
C.5.	Resource Binding Namespace
C.6.	Stanza Error Namespace
Appendix D.	Contact Addresses
Appendix E.	Account Provisioning
Appendix F.	Differences From RFC 3920
Appendix G.	Copying Conditions
§	Index
§	Author's Address

1. Introduction

[TOC](#)

1.1. Overview

[TOC](#)

The Extensible Messaging and Presence Protocol (XMPP) is an application profile of the Extensible Markup Language [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#) for streaming XML data in close to real time between any two (or more) network-aware entities. XMPP is typically used to exchange messages, share presence information, and engage in structured request-response interactions. The basic syntax and semantics of XMPP were developed originally within the Jabber open-source community, mainly in 1999. In late 2002, the XMPP Working Group was chartered with developing an adaptation of the core Jabber protocol that would be suitable as an IETF instant messaging (IM) and presence technology. As a result of work by the XMPP WG, [\[RFC3920\] \(Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol \(XMPP\): Core," October 2004.\)](#) and [\[RFC3921\] \(Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol \(XMPP\): Instant](#)

[Messaging and Presence," October 2004.](#)) were published in October 2004, representing the most complete definition of XMPP at that time. As a result of extensive implementation and deployment experience with XMPP since 2004, as well as more formal interoperability testing carried out under the auspices of the XMPP Standards Foundation (XSF), this document reflects consensus from the XMPP developer community regarding XMPP's core XML streaming technology. In particular, this document incorporates the following backward-compatible changes from RFC 3920:

- *Incorporated corrections and errata
- *Added examples throughout
- *Clarified and more completely specified matters that were underspecified
- *Modified text to reflect updated technologies for which XMPP is a using protocol, e.g., Transport Layer Security (TLS) and the Simple Authentication and Security Layer (SASL)
- *Defined several additional stream, stanza, and SASL error conditions
- *Removed the deprecated DIGEST-MD5 SASL mechanism [\[DIGEST-MD5\]](#) (Leach, P. and C. Newman, "Using Digest Authentication as a SASL Mechanism," May 2000.) as a mandatory-to-implement technology
- *Added the TLS plus the SASL PLAIN mechanism [\[PLAIN\]](#) (Zeilenga, K., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism," August 2006.) as a mandatory-to-implement technology
- *Defined of optional support for multiple resources over the same connection
- *Transferred historical documentation for the server dialback protocol from this specification to a separate specification

Therefore, this document defines the core features of XMPP 1.0, thus obsoleting RFC 3920.

Note: [\[rfc3921bis\]](#) (Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," March 2009.) defines the XMPP features needed to provide the basic instant messaging and presence functionality that is described in [\[IMP-REQS\]](#) (Day, M., Aggarwal, S., and J. Vincent, "Instant Messaging / Presence Protocol Requirements," February 2000.).

1.2. Functional Summary

[TOC](#)

This non-normative section provides a developer-friendly, functional summary of XMPP; refer to the sections that follow for a normative definition of XMPP.

The purpose of XMPP is to enable the exchange of relatively small pieces of structured data (called "XML stanzas") over a network between any two (or more) entities. XMPP is implemented using a client-server architecture, wherein a client needs to connect to a server in order to gain access to the network and thus be allowed to exchange XML stanzas with other entities (which can be associated with other servers). The process whereby a client connects to a server, exchanges XML stanzas, and ends the connection is:

1. Determine the hostname and port at which to connect
2. Open a TCP connection
3. Open an XML stream
4. Complete TLS negotiation for channel encryption (recommended)
5. Complete SASL negotiation for authentication
6. Bind a resource to the stream
7. Exchange an unbounded number of XML stanzas with other entities on the network
8. Close the XML stream
9. Close the TCP connection

Within XMPP, one server can optionally connect to another server to enable inter-domain or inter-server communication. For this to happen, the two servers need to negotiate a connection between themselves and then exchange XML stanzas; the process for doing so is:

1. Determine the hostname and port at which to connect
2. Open a TCP connection
3. Open an XML stream
4. Complete TLS negotiation for channel encryption (recommended)
5. Complete SASL negotiation for authentication *

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

* Note: Depending on local service policies, it is possible that a deployed server will use the older server dialback protocol to provide weak identity verification in cases where SASL negotiation would not result in strong authentication (e.g., because TLS negotiation was not mandated by the peer server, or because the certificate presented by the peer server during TLS negotiation is self-signed and thus provides only weak identity); for details, see [\[XEP-0220\] \(Saint-Andre, P. and J. Miller, "Server Dialback," October 2008.\)](#).

In the sections following discussion of XMPP architecture and XMPP addresses, this document specifies how clients connect to servers and specifies the basic semantics of XML stanzas. However, this document does not define the "payloads" of the XML stanzas that might be exchanged once a connection is successfully established; instead, those payloads are defined by various XMPP extensions. For example, [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#) defines extensions for basic instant messaging and presence functionality. In addition, various specifications produced in the XSF's XEP series [\[XEP-0001\] \(Saint-Andre, P., "XMPP Extension Protocols," January 2008.\)](#) define extensions for a wide range of more advanced functionality.

1.3. Conventions

[TOC](#)

The following capitalized keywords are to be interpreted as described in [\[TERMS\] \(Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.\)](#): "MUST", "SHALL", "REQUIRED"; "MUST NOT", "SHALL NOT"; "SHOULD", "RECOMMENDED"; "SHOULD NOT", "NOT RECOMMENDED"; "MAY", "OPTIONAL".

The term "whitespace" is used to refer to any character that matches production [3] content of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#), i.e., any instance of SP, HT, CR, and LF.

Following the "XML Notation" used in [\[IRI\] \(Duerst, M. and M. Suignard, "Internationalized Resource Identifiers \(IRIs\)," January 2005.\)](#) to represent characters that cannot be rendered in ASCII-only documents, some examples in this document use the form "&#x...." as a notational

device to represent Unicode characters (e.g., the string "ř" stands for the Unicode character LATIN SMALL LETTER R WITH CARON). In examples, lines have been wrapped for improved readability, "[...]" means elision, and the following prepended strings are used (these prepended strings are not to be sent over the wire):

*C: = a client

*E: = any XMPP entity

*I: = an initiating entity

*P: = a peer server

*R: = a receiving entity

*S: = a server

*S1: = server1

*S2: = server2

1.4. Acknowledgements

[TOC](#)

The editor of this document finds it impossible to appropriately acknowledge the many individuals who have provided comments regarding the protocols defined herein. However, thanks are due to those who have who have provided implementation feedback, bug reports, requests for clarification, and suggestions for improvement since the publication of the RFC this document supersedes. The editor has endeavored to address all such feedback, but is solely responsible for any remaining errors and ambiguities.

1.5. Discussion Venue

[TOC](#)

The document editor and the broader XMPP developer community welcome discussion and comments related to the topics presented in this document. The preferred forum is the <standards@xmpp.org> mailing list, for which archives and subscription information are available at <http://mail.jabber.org/mailman/listinfo/standards>.

[TOC](#)

2. Architecture

2.1. Overview

[TOC](#)

XMPP assumes a client-server architecture, wherein a client utilizing XMPP accesses a server (normally over a [\[TCP\] \(Postel, J., "Transmission Control Protocol," September 1981.\)](#) connection) and servers can also communicate with each other over TCP connections. A simplified architectural diagram for a typical deployment is shown here, where the entities have the following significance:

```
*romeo@example.net -- an XMPP user.  
  
*example.net -- an XMPP server.  
  
*im.example.com -- an XMPP server.  
  
*juliet@im.example.com -- an XMPP user.
```

```
example.net ----- im.example.com  
  |                   |  
  |                   |  
romeo@example.net    juliet@im.example.com
```

Note: Architectures that employ [XML streams \(XML Streams\)](#) and [XML stanzas \(XML Stanzas\)](#) but that establish peer-to-peer connections directly between clients using technologies based on [\[LINKLOCAL\] \(Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses," May 2005.\)](#) have been deployed, but such architectures are not XMPP and are best described as "XMPP-like"; for details, see [\[XEP-0174\] \(Saint-Andre, P., "Link-Local Messaging," September 2007.\)](#). In addition, XML streams can be established end-to-end over any reliable transport, including extensions to XMPP itself; for details, see [\[XEP-0246\] \(Saint-Andre, P., "End-to-End XML Streams," June 2008.\)](#).

2.2. Server

[TOC](#)

A SERVER is an entity whose primary responsibilities are to:

*Manage [XML streams \(XML Streams\)](#) with local clients and deliver [XML stanzas \(XML Stanzas\)](#) to those clients over the negotiated XML streams.

*Subject to local service policies on server-to-server communication, manage [XML streams \(XML Streams\)](#) with foreign servers and route [XML stanzas \(XML Stanzas\)](#) to those servers over the negotiated XML streams.

Depending on the application, the secondary responsibilities of an XMPP server can include:

*Storing XML data that is used by clients (e.g., contact lists for users of XMPP-based instant messaging and presence applications as defined in [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#)); in this case, the relevant XML stanza is handled directly by the server itself on behalf of the client and is not routed to a foreign server or delivered to a local entity.

*Hosting local services that also use XMPP as the basis for communication but that provide additional functionality beyond that defined in this document or in [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#); examples include multi-user conferencing services as specified in [\[XEP-0045\] \(Saint-Andre, P., "Multi-User Chat," July 2007.\)](#) and publish-subscribe services as specified in [\[XEP-0060\] \(Millard, P., Saint-Andre, P., and R. Meijer, "Publish-Subscribe," September 2007.\)](#).

2.3. Client

[TOC](#)

A CLIENT is an entity that establishes an XML stream with a server by authenticating using the credentials of a local account and that then completes [resource binding \(Resource Binding\)](#) in order to enable delivery of XML stanzas via the server to the client. A client then uses XMPP to communicate with its server, other clients, and any other accessible entities on a network. Multiple clients can connect simultaneously to a server on behalf of a local account, where each client is differentiated by the resource identifier portion of an XMPP address (e.g., <node@domain/home> vs. <node@domain/work>), as defined under [Section 3 \(Addresses\)](#) and [Section 8 \(Resource Binding\)](#). The RECOMMENDED port for TCP connections between a client and a server is 5222, as registered with the IANA (see [Section 16.9 \(Port Numbers\)](#)).

2.4. Network

[TOC](#)

Because each server is identified by a network address and because server-to-server communication is a straightforward extension of the client-to-server protocol, in practice the system consists of a network of servers that inter-communicate. Thus, for example, <juliet@im.example.com> is able to exchange messages, presence, and other information with <romeo@example.net>. This pattern is familiar from messaging protocols (such as [\[SMTP\] \(Klensin, J., "Simple Mail Transfer Protocol," April 2001.\)](#)) that make use of network addressing standards. Communication between any two servers is OPTIONAL. The RECOMMENDED port for TCP connections between servers is 5269, as registered with the IANA (see [Section 16.9 \(Port Numbers\)](#)).

3. Addresses

[TOC](#)

3.1. Overview

[TOC](#)

An ENTITY is anything that is network-addressable and that can communicate using XMPP. For historical reasons, the native address of an XMPP entity is called a JABBER IDENTIFIER or JID. A valid JID contains a set of ordered elements formed of an XMPP node identifier, domain identifier, and resource identifier.

The syntax for a JID is defined as follows using the Augmented Backus-Naur Form as specified in [\[ABNF\] \(Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," January 2008.\)](#).

```

jid          = [ node "@" ] domain [ "/" resource ]
node         = 1*(nodepoint)
              ; a "nodepoint" is a UTF-8 encoded Unicode code
              ; point that satisfies the Nodeprep profile of
              ; stringprep
domain       = fqdn / address-literal
fqdn         = *(ldhlabel ".") toplabel
ldhlabel     = letdig [*61(ldh) letdig]
toplabel     = ALPHA *61(ldh) letdig
letdig       = ALPHA / DIGIT
ldh          = ALPHA / DIGIT / "-"
address-literal = IPv4address / IPv6address
              ; the "IPv4address" and "IPv6address" rules are
              ; defined in RFC 3986
resource     = 1*(resourcepoint)
              ; a "resourcepoint" is a UTF-8 encoded Unicode
              ; code point that satisfies the Resourceprep
              ; profile of stringprep

```

All JIDs are based on the foregoing structure. One common use of this structure is to identify a messaging and presence account, the server that hosts the account, and a connected resource (e.g., a specific device) in the form of <node@domain/resource>. However, node types other than clients are possible; for example, a specific chat room offered by a multi-user conference service (see [\[XEP-0045\] \(Saint-Andre, P., "Multi-User Chat," July 2007.\)](#)) could be addressed as <room@service> (where "room" is the name of the chat room and "service" is the hostname of the multi-user conference service) and a specific occupant of such a room could be addressed as <room@service/nick> (where "nick" is the occupant's room nickname). Many other JID types are possible (e.g., <domain/resource> could be a server-side script or service).

Each allowable portion of a JID (node identifier, domain identifier, and resource identifier) MUST NOT be more than 1023 bytes in length, resulting in a maximum total size (including the '@' and '/' separators) of 3071 bytes.

Note: While the format of a JID is consistent with [\[URI\] \(Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier \(URI\): Generic Syntax," January 2005.\)](#), an entity's address on an XMPP network MUST be represented as a JID (without a URI scheme) and not a [\[URI\] \(Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier \(URI\): Generic Syntax," January 2005.\)](#) or [\[IRI\] \(Duerst, M. and M. Suignard, "Internationalized Resource Identifiers \(IRIs\)," January 2005.\)](#) as specified in [\[XMPP-URI\] \(Saint-Andre, P., "Internationalized Resource Identifiers \(IRIs\) and Uniform Resource Identifiers \(URIs\) for the Extensible Messaging and Presence Protocol \(XMPP\)," February 2008.\)](#); the latter specification

is provided only for identification and interaction outside the context of the XMPP wire protocol itself.

3.2. Domain Identifier

[TOC](#)

The DOMAIN IDENTIFIER portion 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 domain identifier is a valid JID). Typically a domain identifier identifies the "home" server to which clients connect for XML routing and data management functionality. However, it is not necessary for an XMPP domain identifier to identify an entity that provides core XMPP server functionality (e.g., a domain identifier can identify an entity such as a multi-user conference service, a publish-subscribe service, or a user directory).

Note: A single server can service multiple domain identifiers, i.e., multiple local domains; this is typically referred to as virtual hosting.

The domain identifier for every server or service that will communicate over a network SHOULD be a fully qualified domain name (see [\[DNS\] \(Mockapetris, P., "Domain names - implementation and specification," November 1987.\)](#)); while the domain identifier MAY be either an Internet Protocol (IPv4 or IPv6) address or a text label that is resolvable on a local network (commonly called an "unqualified hostname"), it is possible that domain identifiers that are IP addresses will not be acceptable to other services for the sake of interdomain communication. Furthermore, domain identifiers that are unqualified hostnames MUST NOT be used on public networks but MAY be used on private networks.

Note: If the domain identifier includes a final character considered to be a label separator (dot) by [\[IDNA\] \(Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications \(IDNA\)," March 2003.\)](#) or [\[DNS\] \(Mockapetris, P., "Domain names - implementation and specification," November 1987.\)](#), this character MUST be stripped from the domain identifier before the JID of which it is a part is used for the purpose of routing an XML stanza, comparing against another JID, or constructing an [\[XMPP-URI\] \(Saint-Andre, P., "Internationalized Resource Identifiers \(IRIs\) and Uniform Resource Identifiers \(URIs\) for the Extensible Messaging and Presence Protocol \(XMPP\)," February 2008.\)](#); in particular, the character MUST be stripped before any other canonicalization steps are taken, such as application of the [\[NAMEPREP\] \(Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names \(IDN\)," March 2003.\)](#) profile of [\[STRINGPREP\] \(Hoffman,](#)

[P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) or completion of the ToASCII operation as described in [\[IDNA\] \(Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications \(IDNA\)," March 2003.\)](#).

A domain identifier MUST be an "internationalized domain name" as defined in [\[IDNA\] \(Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications \(IDNA\)," March 2003.\)](#), that is, "a domain name in which every label is an internationalized label". When preparing a text label (consisting of a sequence of Unicode code points) for representation as an internationalized label in the process of constructing an XMPP domain identifier or comparing two XMPP domain identifiers, an application MUST ensure that for each text label it is possible to apply without failing the ToASCII operation specified in [\[IDNA\] \(Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications \(IDNA\)," March 2003.\)](#) with the UseSTD3ASCIIRules flag set (thus forbidding ASCII code points other than letters, digits, and hyphens). If the ToASCII operation can be applied without failing, then the label is an internationalized label. An internationalized domain name (and therefore an XMPP domain identifier) is constructed from its constituent internationalized labels by following the rules specified in [\[IDNA\] \(Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications \(IDNA\)," March 2003.\)](#).

Note: The ToASCII operation includes application of the [\[NAMEPREP\] \(Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names \(IDN\)," March 2003.\)](#) profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) and encoding using the algorithm specified in [\[PUNYCODE\] \(Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications \(IDNA\)," March 2003.\)](#); for details, see [\[IDNA\] \(Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications \(IDNA\)," March 2003.\)](#). Although the output of the ToASCII operation is not used in XMPP, it MUST be possible to apply that operation without failing.

3.3. Node Identifier

[TOC](#)

The NODE IDENTIFIER portion of a JID is an optional secondary identifier placed before the domain identifier and separated from the latter by the '@' character. Typically a node identifier uniquely identifies the entity requesting and using network access provided by a

server (i.e., a local account), although it can also represent other kinds of entities (e.g., a chat room associated with a multi-user conference service). The entity represented by an XMPP node identifier is addressed within the context of a specific domain.

A node identifier MUST be formatted such that the Nodeprep profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) can be applied without failing (see [Appendix A \(Nodeprep\)](#)). Before comparing two node identifiers, an application MUST first ensure that the Nodeprep profile has been applied to each identifier (the profile need not be applied each time a comparison is made, as long as it has been applied before comparison).

3.4. Resource Identifier

[TOC](#)

The RESOURCE IDENTIFIER portion of a JID is an optional tertiary identifier placed after the domain identifier and separated from the latter by the '/' character. A resource identifier can modify either a <node@domain> address or a mere <domain> address. Typically a resource identifier uniquely identifies a specific connection (e.g., a device or location) or object (e.g., a participant in a multi-user conference room) belonging to the entity associated with an XMPP node identifier at a local domain.

When an XMPP address does not include a resource identifier (i.e., when it is of the form <domain> or <node@domain>), it is referred to as a BARE JID. When an XMPP address includes a resource identifier (i.e., when it is of the form <domain/resource> or <node@domain/resource>), is referred to as a FULL JID.

A resource identifier MUST be formatted such that the Resourceprep profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) can be applied without failing (see [Appendix B \(Resourceprep\)](#)). Before comparing two resource identifiers, an application MUST first ensure that the Resourceprep profile has been applied to each identifier (the profile need not be applied each time a comparison is made, as long as it has been applied before comparison).

Note: For historical reasons, the term "resource identifier" is used in XMPP to refer to the optional portion of an XMPP address that follows the domain identifier and the "/" separator character; this use of the term "resource identifier" is not to be confused with the meanings of "resource" and "identifier" provided in Section 1.1 of [\[URI\] \(Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier \(URI\): Generic Syntax," January 2005.\)](#).

XMPP entities SHOULD consider resource identifiers to be opaque strings and SHOULD NOT impute meaning to any given resource identifier. In particular, the use of the '/' character as a separator between the domain identifier and the resource identifier does not imply that resource identifiers are hierarchical in the way that, say, HTTP addresses are hierarchical; thus for example an XMPP address of the form <node@domain/foo/bar> does not identify a resource "bar" that exists below a resource "foo" in a hierarchy of resources associated with the entity "node@domain".

3.5. Determination of Addresses

[TOC](#)

After the parties to an XML stream have completed the appropriate aspects of stream negotiation (typically [SASL negotiation \(SASL Negotiation\)](#) and, if appropriate, [resource binding \(Resource Binding\)](#)) the receiving entity for a stream MUST determine the initiating entity's JID.

For server-to-server communication, the initiating server's JID MUST be the authorization identity (as defined by [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#)), either (1) as directly communicated by the initiating server during [SASL negotiation \(SASL Negotiation\)](#) or (2) as derived by the receiving server from the authentication identity if no authorization identity was specified during [SASL negotiation \(SASL Negotiation\)](#). (For information about the determination of addresses in the absence of SASL negotiation when the older server dialback protocol is used, see [\[XEP-0220\] \(Saint-Andre, P. and J. Miller, "Server Dialback," October 2008.\)](#).)

For client-to-server communication, the client's bare JID (<node@domain>) MUST be the authorization identity (as defined by [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#)), either (1) as directly communicated by the client during [SASL negotiation \(SASL Negotiation\)](#) or (2) as derived by the server from the authentication identity if no authorization identity was specified during [SASL negotiation \(SASL Negotiation\)](#). The resource identifier portion of the full JID (<node@domain/resource>) MUST be the resource identifier negotiated by the client and server during [resource binding \(Resource Binding\)](#). The receiving entity MUST ensure that the resulting JID (including node identifier, domain identifier, resource identifier, and separator characters) conforms to the rules and formats defined earlier in this section; to meet this restriction, the receiving entity MAY replace the JID sent by the initiating entity with the canonicalized JID as determined by the receiving entity.

4. TCP Binding

[TOC](#)

4.1. Scope

[TOC](#)

As XMPP is defined in this specification, an initiating entity (client or server) MUST open a Transmission Control Protocol [\[TCP\] \(Postel, J., "Transmission Control Protocol," September 1981.\)](#) connection at the receiving entity (server) before it negotiates XML streams with the receiving entity. The rules specified in the following sections apply to the TCP binding.

4.2. Hostname Resolution

[TOC](#)

Before opening the TCP connection, the initiating entity first MUST resolve the Domain Name System (DNS) hostname associated with the receiving entity and determine the appropriate TCP port for communication with the receiving entity. The process is:

1. Attempt to resolve the hostname using (a) a [\[DNS-SRV\] \(Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services \(DNS SRV\)," February 2000.\)](#) Service of "xmpp-client" (for client-to-server connections) or "xmpp-server" (for server-to-server connections) and (b) a Proto of "tcp", resulting in resource records such as "_xmpp-client._tcp.example.net." or "_xmpp-server._tcp.im.example.com.". The result of the SRV lookup will be one or more combinations of a port and hostname, which hostnames the initiating entity MUST resolve according to returned SRV record weight (if the result of the SRV lookup is a single RR with a Target of ".", i.e. the root domain, the initiating entity MUST abort SRV processing but SHOULD attempt a fallback resolution as described below). The initiating entity uses the IP address(es) from the first successfully resolved hostname (with the corresponding port number returned by the SRV lookup) in order to connect to the receiving entity. If the initiating entity fails to connect using one of the IP addresses, the initiating entity uses the next resolved IP address to connect. If the initiating entity fails to connect using all resolved IP addresses, then the initiating entity repeats the process of resolution and connection for the next hostname returned by the SRV lookup.

2. If the SRV lookup aborts or fails, the fallback SHOULD be a normal IPv4 or IPv6 address record resolution to determine the IP address, 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.
3. For client-to-server connections, the fallback MAY be a [\[DNS-TXT\] \(Rosenbaum, R., "Using the Domain Name System To Store Arbitrary String Attributes," May 1993.\)](#) lookup for alternative connection methods, for example as described in [\[XEP-0156\] \(Hildebrand, J. and P. Saint-Andre, "Discovering Alternative XMPP Connection Methods," June 2007.\)](#).

Note: If the initiating entity has been explicitly configured to associate a particular hostname (and potentially port) with the original hostname of the receiving entity (say, to "hardcode" an association between an original hostname of example.net and a configured hostname and of webcm.example.com:80), the initiating entity SHALL use the configured name instead of performing the foregoing resolution process on the original name.

Note: Many XMPP servers are implemented in such a way that they can host additional services (beyond those defined in this specification and [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#)) at hostnames that are subdomains of the hostname of the main XMPP service (e.g., conference.example.net for a [\[XEP-0045\] \(Saint-Andre, P., "Multi-User Chat," July 2007.\)](#) service associated with the example.net XMPP service) or subdomains of the first-level domain of the underlying host (e.g., muc.example.com for a [\[XEP-0045\] \(Saint-Andre, P., "Multi-User Chat," July 2007.\)](#) service associated with the im.example.com XMPP service). If an entity from a remote domain wishes to use such additional services, it would generate an appropriate XML stanza and the remote domain itself would attempt to resolve the service's hostname via an SRV lookup on resource records such as "_xmpp-server._tcp.conference.example.net." or "_xmpp-server._tcp.muc.example.com.". Therefore if a service wishes to enable entities from remote domains to access these additional services, it needs to advertise the appropriate "_xmpp-server" SRV records in addition to the "_xmpp-server" record for its main XMPP service.

4.3. Client-to-Server Communication

[TOC](#)

Because a client is subordinate to a server and therefore a client authenticates to the server but the server does not necessarily

authenticate to the client, it is necessary to have only one TCP connection between client and server. Thus the server MUST allow the client to share a single TCP connection for XML stanzas sent from client to server and from server to client (i.e., the initial stream and response stream as specified under [Section 5 \(XML Streams\)](#)).

4.4. Server-to-Server Communication

[TOC](#)

Because two servers are peers and therefore each peer MUST authenticate with the other, the servers MUST use two TCP connections: one for XML stanzas sent from the first server to the second server and another (initiated by the second server) for XML stanzas from the second server to the first server.

This rule applies only to [XML stanzas \(XML Stanzas\)](#). Therefore during [STARTTLS negotiation \(STARTTLS Negotiation\)](#) and [SASL negotiation \(SASL Negotiation\)](#) the servers would use one TCP connection, but after stream setup that 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.

4.5. Reconnection

[TOC](#)

It can happen that an XMPP server goes offline while servicing TCP connections from local clients and from other servers. Because the number of such connections can be quite large, the reconnection algorithm employed by entities that seek to reconnect can have a significant impact on software and network performance. The following guidelines are RECOMMENDED:

- *The time to live (TTL) specified in Domain Name System records MUST be honored, even if DNS results are cached; if the TTL has not expired, an entity that seeks to reconnect MUST NOT re-resolve the server hostname before reconnecting.

- *The time that expires before an entity first seeks to reconnect MUST be randomized (e.g., so that all clients do not attempt to reconnect exactly 30 seconds after being disconnected).

- *If the first reconnection attempt does not succeed, an entity MUST back off increasingly on the time between subsequent reconnection attempts.

Note: Because it is possible that a disconnected entity cannot determine the cause of disconnection (e.g., because there was no explicit stream error) or does not require a new stream for immediate communication (e.g., because the stream was idle and therefore timed out), it SHOULD NOT assume that it needs to reconnect immediately.

4.6. Other Bindings

[TOC](#)

There is no necessary coupling of an XML stream to a TCP connection. For example, two entities could connect to each other via another transport, such as [\[HTTP\] \(Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999.\)](#) as specified in [\[XEP-0124\] \(Paterson, I., Smith, D., and P. Saint-Andre, "Bidirectional-streams Over Synchronous HTTP \(BOSH\)," February 2007.\)](#) and [\[XEP-0206\] \(Paterson, I., "XMPP Over BOSH," June 2007.\)](#). Although this specification neither encourages nor discourages other bindings, it defines only a binding of XMPP to TCP.

5. XML Streams

[TOC](#)

5.1. Overview

[TOC](#)

Two fundamental concepts make possible the rapid, asynchronous exchange of relatively small payloads of structured information between presence-aware entities: XML streams and XML stanzas. These terms are defined as follows.

Definition of XML Stream: An XML STREAM is a container for the exchange of XML elements between any two entities over a network. The start of an XML stream is denoted unambiguously by an opening STREAM HEADER (i.e., an XML <stream> tag with appropriate attributes and namespace declarations), while the end of the XML stream is denoted unambiguously by a closing XML </stream> tag. During the life of the stream, the entity that initiated it can send an unbounded number of XML elements over the stream, either elements used to negotiate the stream (e.g., to complete [TLS negotiation \(STARTTLS Negotiation\)](#) or [SASL negotiation \(SASL](#)

[Negotiation](#)) or XML stanzas. The INITIAL STREAM is negotiated from the initiating entity (typically a client or server) to the receiving entity (typically a server), and can be seen as corresponding to the initiating entity's "connection" or "session" with the receiving entity. The initial stream enables unidirectional communication from the initiating entity to the receiving entity; in order to enable information exchange from the receiving entity to the initiating entity, the receiving entity MUST negotiate a stream in the opposite direction (the RESPONSE STREAM).

Definition of XML Stanza: An XML STANZA is a discrete semantic unit of structured information that is sent from one entity to another over an XML stream, and is the basic unit of meaning in XMPP. An XML stanza exists at the direct child level of the root <stream/> element; the start of any XML stanza is denoted unambiguously by the element start tag at depth=1 of the XML stream (e.g., <presence>), and the end of any XML stanza is denoted unambiguously by the corresponding close tag at depth=1 (e.g., </presence>). The only XML stanzas defined herein are the <message/>, <presence/>, and <iq/> elements qualified by the default namespace for the stream, as described under [Section 9 \(XML Stanzas\)](#); for example, an XML element sent for the purpose of [TLS negotiation \(STARTTLS Negotiation\)](#) or [SASL negotiation \(SASL Negotiation\)](#) is not considered to be an XML stanza, nor is a stream error or a stream feature. An XML stanza MAY contain child elements (with accompanying attributes, elements, and XML character data) as necessary in order to convey the desired information, which MAY be qualified by any XML namespace (see [\[XML-NAMES\] \(Layman, A., Hollander, D., Tobin, R., and T. Bray, "Namespaces in XML 1.1 \(Second Edition\)," August 2006.\)](#) as well as [Section 9.4 \(Extended Content\)](#) herein).

Consider the example of a client's connection to a server. In order to connect to a server, a client initiates an XML stream by sending a stream header to the server, optionally preceded by a text declaration specifying the XML version and the character encoding supported (see [Section 12.5 \(Inclusion of Text Declaration\)](#) and [Section 12.6 \(Character Encoding\)](#)). Subject to local policies and service provisioning, the server then replies with a second XML stream back to the client, again optionally preceded by a text declaration. Once the client has completed [SASL negotiation \(SASL Negotiation\)](#) and [resource binding \(Resource Binding\)](#), the client can send an unbounded number of XML stanzas over the stream. When the client desires to close the stream, it simply sends a closing </stream> tag to the server as further described under [Section 5.7 \(Closing a Stream\)](#).

In essence, then, an XML stream acts as an envelope for all the XML stanzas sent during a connection. We can represent this in a simplistic fashion as follows.

```

+-----+
| <stream>          |
|-----|
| <presence>        |
|   <show/>         |
| </presence>       |
|-----|
| <message to='foo'>|
|   <body/>         |
| </message>        |
|-----|
| <iq to='bar'>      |
|   <query/>        |
| </iq>             |
|-----|
| <iq from='bar'>    |
|   <query/>        |
| </iq>             |
|-----|
| [ ... ]          |
|-----|
| </stream>         |
+-----+

```

Note: Those who are accustomed to thinking of XML in a document-centric manner might view a client's connection to a server as consisting of two open-ended XML documents: one from the client to the server and one from the server to the client. On this analogy, the two XML streams can be considered equivalent to two "documents" (matching production [1] content of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#)) that are built up through the accumulation of XML stanzas, the root <stream/> element can be considered equivalent to the "document entity" for each "document" (as described in Section 4.8 of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#)), and the XML stanzas sent over the streams can be considered equivalent to "fragments" of the "documents" as described in [\[XML-FRAG\] \(Grosso, P. and D. Veillard, "XML Fragment Interchange," February 2001.\)](#). However, this perspective is merely an analogy; XMPP does not deal in documents and fragments but in streams and stanzas.

5.2. Stream Security

[TOC](#)

For the purpose of stream security, both Transport Layer Security (see [Section 6 \(STARTTLS Negotiation\)](#)) and the Simple Authentication and Security Layer (see [Section 7 \(SASL Negotiation\)](#)) are mandatory to implement. Use of these technologies results in high security as described under [Section 15.1 \(High Security\)](#).

The initial stream and the response stream MUST be secured separately, although security in both directions MAY be established via mechanisms that provide mutual authentication.

The initiating entity MUST NOT attempt to send [XML stanzas \(XML Stanzas\)](#) over the stream before the stream has been authenticated.

However, if it does attempt to do so, the receiving entity MUST NOT accept such stanzas and MUST return a <not-authorized/> stream error. This rule applies to XML stanzas only (i.e., <message/>, <presence/>, and <iq/> elements qualified by the default namespace) and not to XML elements used for stream negotiation (e.g., elements used to complete [TLS negotiation \(STARTTLS Negotiation\)](#) or [SASL negotiation \(SASL Negotiation\)](#)).

5.3. Stream Attributes

[TOC](#)

The attributes of the root <stream/> element are defined in the following sections.

Note: The attributes of the root <stream/> element are not prepended by a 'stream:' prefix because, as explained in [\[XML-NAMES\] \(Layman, A., Hollander, D., Tobin, R., and T. Bray, "Namespaces in XML 1.1 \(Second Edition\)," August 2006.\)](#), "[d]efault namespace declarations do not apply directly to attribute names; the interpretation of unprefixed attributes is determined by the element on which they appear."

5.3.1. from

[TOC](#)

The 'from' attribute communicates an XMPP identity of the entity sending the stream element.

Note: It is possible for an entity to have more than one XMPP identity (e.g., in the case of a server that provides virtual hosting). It is also possible that an entity does not know the XMPP identity of the principal controlling the entity (e.g., because the

XMPP identity is assigned at a level other than the XMPP application layer, as in the General Security Service Application Program Interface [\[GSS-API\] \(Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1," January 2000.\)](#)).

For initial stream headers in client-to-server communication, if the client knows the XMPP identity of the principal controlling the client (typically an account name of the form <node@domain>), then it SHOULD include the 'from' attribute and set its value to that identity. If the client does not know the XMPP identity of the principal controlling the client, then it MUST NOT include the 'from' attribute.

```
I: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

For initial stream headers in server-to-server communication, a server MUST include the 'from' attribute and MUST set its value to a hostname serviced by the initiating entity.

```
I: <?xml version='1.0'?>
  <stream:stream
    from='example.net'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

For response stream headers in both client-to-server and server-to-server communication, the receiving entity MUST include the 'from' attribute and MUST set its value to a hostname serviced by the receiving entity (which MAY be a hostname other than that specified in the 'to' attribute of the initial stream header).

```
R: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

Whether or not the 'from' attribute is included, each entity MUST verify the identity of the other entity before exchanging XML stanzas with it (see [Section 15.3 \(Client-to-Server Communication\)](#) and [Section 15.4 \(Server-to-Server Communication\)](#)).

Note: It is possible that implementations based on an earlier revision of this specification will not include the 'from' address on stream headers; an entity SHOULD be liberal in accepting such stream headers.

5.3.2. to

[TOC](#)

For initial stream headers in both client-to-server and server-to-server communication, the initiating entity MUST include the 'to' attribute and MUST set its value to a hostname that the initiating entity knows or expects the receiving entity to service.

```
I: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

For response stream headers in client-to-server communication, if the client included a 'from' attribute in the initial stream header then the server MUST include a 'to' attribute in the response stream header and MUST 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 MUST NOT include a 'to' attribute in the response stream header.

```
R: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

For response stream headers in server-to-server communication, the receiving entity MUST include a 'to' attribute in the response stream header and MUST set its value to the hostname specified in the 'from' attribute of the initial stream header.

```
R: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='g4qSvGvBxJ+xeAd7QKez0QJFFlw='
    to='example.net'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

Whether or not the 'to' attribute is included, each entity MUST verify the identity of the other entity before exchanging XML stanzas with it (see [Section 15.3 \(Client-to-Server Communication\)](#) and [Section 15.4 \(Server-to-Server Communication\)](#)).

Note: It is possible that implementations based on an earlier revision of this specification will not include the 'to' address on stream headers; an entity SHOULD be liberal in accepting such stream headers.

5.3.3. id

[TOC](#)

The 'id' attribute communicates a unique identifier for the stream. This identifier is called a STREAM ID. The stream ID MUST be generated by the receiving entity when it sends a response stream header, MUST BE unique within the receiving application (normally a server), and MUST be both unpredictable and nonrepeating because it can be security-critical (see [\[RANDOM\] \(Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security," June 2005.\)](#) for recommendations regarding randomness for security purposes).

For initial stream headers, the initiating entity MUST NOT include the 'id' attribute; however, if the 'id' attribute is included, the receiving entity MUST silently ignore it.
For response stream headers, the receiving entity MUST include the 'id' attribute.

```
R: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

5.3.4. xml:lang

[TOC](#)

The 'xml:lang' attribute communicates an entity's preferred or default language for any human-readable XML character data to be sent over the stream. The syntax of this attribute is defined in Section 2.12 of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#); in particular, the value of the 'xml:lang' attribute MUST conform to the NMTOKEN datatype (as defined in Section 2.3 of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#)) and MUST conform to the language identifier format defined in [\[LANGTAGS\] \(Phillips, A. and M. Davis, "Tags for Identifying Languages," September 2006.\)](#).

For initial stream headers, the initiating entity SHOULD include the 'xml:lang' attribute.

```
I: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

For response stream headers, the receiving entity MUST include the 'xml:lang' attribute. If the initiating entity included an 'xml:lang' attribute in its initial stream header and the receiving entity supports that language in the human-readable XML character data that it

generates and sends to the initiating entity (e.g., in the <text/> element for stream and stanza errors), the value of the 'xml:lang' attribute MUST be identifier for the initiating entity's preferred language; if the receiving entity supports a language that closely matches the initiating entity's preferred language (e.g., "de" instead of "de-CH"), then the value of the 'xml:lang' attribute SHOULD be the identifier for the matching language but MAY be the identifier for the default language of the receiving entity; if the receiving entity does not support the initiating entity's preferred language or a closely matching language (or the initiating entity did not include the 'xml:lang' attribute in its initial stream header), then the value of the 'xml:lang' attribute MUST be the identifier for the default language of the receiving entity.

```
R: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

If the initiating entity included the 'xml:lang' attribute in its initial stream header, the receiving entity SHOULD remember that value as the default xml:lang for all stanzas sent by the initiating entity. As described under [Section 9.1.5 \(xml:lang\)](#), the initiating entity MAY include the 'xml:lang' attribute in any XML stanzas it sends over the stream. If the initiating entity does not include the 'xml:lang' attribute in any such stanza, the receiving entity SHOULD add the 'xml:lang' attribute to the stanza, whose value MUST be the identifier for the language preferred by the initiating entity (even if the receiving entity does not support that language for human-readable XML character data it generates and sends to the initiating entity, such as in stream or stanza errors). If the initiating entity includes the 'xml:lang' attribute in any such stanza, the receiving entity MUST NOT modify or delete it.

5.3.5. version

[TOC](#)

The inclusion of the version attribute set to a value of at least "1.0" signals support for the stream-related protocols defined in this specification, including ([TLS negotiation \(STARTTLS Negotiation\)](#), [SASL negotiation \(SASL Negotiation\)](#), [Section 5.5 \(Stream Features\)](#), and [stream errors \(Stream Errors\)](#)).

The version of XMPP specified herein is "1.0"; in particular, XMPP 1.0 encapsulates the stream-related protocols as well as the basic semantics of the three defined XML stanza types (<message/>, <presence/>, and <iq/>).

The numbering scheme for XMPP versions is "<major>.<minor>". The major and minor numbers MUST be treated as separate integers and each number MAY be incremented higher than a single digit. Thus, "XMPP 2.4" would be a lower version than "XMPP 2.13", which in turn would be lower than "XMPP 12.3". Leading zeros (e.g., "XMPP 6.01") MUST be ignored by recipients and MUST NOT be sent.

The major version number will be incremented only if the stream and stanza formats or required actions have changed so dramatically that an older version entity would not be able to interoperate with a newer version entity if it simply ignored the elements and attributes it did not understand and took the actions specified in the older specification.

The minor version number will be incremented only if significant new capabilities have been added to the core protocol (e.g., a newly defined value of the 'type' attribute for message, presence, or IQ stanzas). The minor version number MUST be ignored by an entity with a smaller minor version number, but MAY be used for informational purposes by the entity with the larger minor version number (e.g., the entity with the larger minor version number would simply note that its correspondent would not be able to understand that value of the 'type' attribute and therefore would not send it).

The following rules apply to the generation and handling of the 'version' attribute within stream headers:

1. The initiating entity MUST set the value of the 'version' attribute in the initial stream header to the highest version number it supports (e.g., if the highest version number it supports is that defined in this specification, it MUST set the value to "1.0").
2. The receiving entity MUST set the value of the 'version' attribute in the response stream header to either the value supplied by the initiating entity or the highest version number supported by the receiving entity, whichever is lower. The receiving entity MUST perform a numeric comparison on the major and minor version numbers, not a string match on "<major>.<minor>".
3. If the version number included in the response stream header is at least one major version lower than the version number included in the initial stream header and newer version entities cannot interoperate with older version entities as described, the initiating entity SHOULD generate an <unsupported-version/> stream error.

4. If either entity receives a stream header with no 'version' attribute, the entity MUST consider the version supported by the other entity to be "0.9" and SHOULD NOT include a 'version' attribute in the response stream header.

5.3.6. Summary of Stream Attributes

[TOC](#)

The following table summarizes the attributes of the root <stream/> element.

initiating to receiving		receiving to initiating	
to	JID of receiver	JID of initiator	
from	JID of initiator	JID of receiver	
id	silently ignored	stream identifier	
xml:lang	default language	default language	
version	XMPP 1.0+ supported	XMPP 1.0+ supported	

5.4. Namespace Declarations

[TOC](#)

The stream element MUST possess both a streams namespace declaration and a default namespace declaration (as "namespace declaration" is defined in [\[XML-NAMES\] \(Layman, A., Hollander, D., Tobin, R., and T. Bray, "Namespaces in XML 1.1 \(Second Edition\)," August 2006.\)](#)). For detailed information regarding the streams namespace and default namespace, see [Section 12.2 \(XML Namespace Names and Prefixes\)](#).

5.5. Stream Features

[TOC](#)

If the initiating entity includes the 'version' attribute set to a value of at least "1.0" in the initial stream header, after sending the response stream header the receiving entity MUST send a <features/> child element (prefixed by the streams namespace prefix) to the initiating entity in order to announce any stream-level features that can be negotiated or capabilities that otherwise need to be advertised.

```

R: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
R: <stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
    <required/>
  </starttls>
</stream:features>

```

Stream features are used mainly to advertise [TLS negotiation \(STARTTLS Negotiation\)](#), [SASL negotiation \(SASL Negotiation\)](#), and [resource binding \(Resource Binding\)](#); however, stream features also can be used to advertise features associated with various XMPP extensions. If it is mandatory for a feature to be successfully negotiated before the initiating entity will be allowed to proceed with the sending of XML stanzas or with further steps of the stream negotiation, the advertisement of that feature SHOULD include an empty <required/> child element but MAY include neither a <required/> element nor an <optional/> element (i.e., features default to required).

```

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

```

If successful negotiation of a feature is discretionary, the advertisement of that feature MUST include an empty <optional/> child element.

```

R: <stream:features>
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session'>
    <optional/>
  </session>
</stream:features>

```

If an entity does not understand or support a feature that has been advertised, it MUST still inspect the feature advertisement to determine if negotiation of the feature is mandatory. If negotiation of an unsupported feature is mandatory (as determined by inclusion of the <required/> child element or the absence of an <optional/> child element), then the entity MUST abort the stream negotiation process. If

negotiation of an unsupported feature is discretionary (as determined by inclusion of the <optional/> child element or the absence of a child element), the entity MUST silently ignore the associated feature advertisement and proceed with the stream negotiation process.

Note: Implementations based on an earlier revision of this specification do not include the <optional/> child element and they include the <required/> child element only in the case of the STARTTLS feature. Entities MUST accept stream feature advertisements without the child elements, and SHOULD consider negotiation of such features to be discretionary.

If it is necessary for a feature to be successfully negotiated before the initiating entity is allowed to proceed with the sending a non-security-related feature or with further steps of the stream negotiation, the receiving entity SHOULD NOT advertise any other stream features until the mandatory feature has been successfully negotiated; however, if the mandatory feature is security-critical (e.g., STARTTLS or SASL) then the receiving entity MUST NOT advertise any other stream features until the security-critical feature has been successfully negotiated.

The order of child elements contained in any given <features/> element is not significant.

After completing negotiation of any stream feature (even stream features that do not require a stream restart), the receiving entity MUST send an updated list of stream features to the initiating entity. However, if there are no features to be advertised then the receiving entity MUST send an empty <features/> element.

```
R: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

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

At any time after stream establishment, the receiving entity MAY send additional or modified stream feature advertisements (e.g., because a new feature has been enabled).

5.6. Restarts During Stream Negotiation

Certain stream features require the initiating entity to send a new initial stream header on successful negotiation of the feature (e.g., after successful negotiation of TLS or SASL). Both parties **MUST** consider the previous stream to be replaced on successful feature negotiation but **MUST NOT** terminate the underlying TCP connection; instead, the parties **MUST** reuse the existing connection, which might be in a new state (e.g., encrypted as a result of TLS negotiation). When the receiving entity receives the new initial stream header, it **MUST** generate a new stream ID (instead of re-using the old stream ID) before sending a new response stream header.

5.7. Closing a Stream

[TOC](#)

An XML stream between two entities can be closed because a stream error has occurred or in some cases in the absence of an error. Where feasible, it is preferable to close a stream only if a stream error has occurred.

A stream is closed by sending a closing `</stream>` tag over the TCP connection.

S: `</stream:stream>`

After an entity sends a closing stream tag, it **MUST NOT** send further data over that stream.

5.7.1. With Stream Error

[TOC](#)

If a stream error has occurred, the entity that detects the error **MUST** close the stream as described under [Section 5.8.1 \(Rules\)](#).

5.7.2. Without Stream Error

[TOC](#)

At any time after XML streams have been negotiated between two entities, either entity **MAY** close its stream to the other party in the absence of a stream error by sending a closing stream tag.

P: `</stream:stream>`

The entity that sends the closing stream tag **SHOULD** wait for the other party to also close its stream.

S: </stream:stream>

However, the entity that sends the first closing stream tag MAY consider both streams to be void if the other party does not send its closing stream tag within a reasonable amount of time (where the definition of "reasonable" is a matter of implementation or deployment).

After the entity that sent the first closing stream tag receives a reciprocal closing stream tag from the other party (or if it considers the stream to be void after a reasonable amount of time), it MUST terminate the underlying TCP connection or connections.

5.7.3. Handling of Idle Streams

[TOC](#)

An XML stream can become idle, i.e., neither entity has sent XMPP traffic over the stream for some period of time. The idle timeout period is a matter of implementation and local service policy; however, it is RECOMMENDED to be liberal in accepting idle streams, since experience has shown that doing so improves the reliability of communications over XMPP networks. In particular, it is typically more efficient to maintain a stream between two servers than it is to aggressively timeout such a stream, especially with regard to synchronization of presence information as described in [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#); therefore it is RECOMMENDED to maintain such a stream since experience has shown that server-to-server streams are cyclical and typically need to be re-established every 24 to 48 hours if they are timed out.

An XML stream can appear idle at the XMPP level because the underlying TCP connection has become idle (e.g., a client's network connection has been lost). The typical method for detecting an idle TCP connection is to send a space character (U+0020) over the TCP connection between XML stanzas, which is allowed for XML streams as described under [Section 12.7 \(Whitespace\)](#). The sending of such a space character is called a WHITESPACE PING. The time between such whitespace pings (or other connection-testing methods) is a matter of implementation and local service policy; however, it is RECOMMENDED that these pings be sent not more than once every 60 seconds.

To close an idle stream with a local client or remote server, a server MUST close the stream without error as explained under [Section 5.7.2 \(Without Stream Error\)](#).

[TOC](#)

5.8. Stream Errors

The root stream element MAY contain an <error/> child element that is prefixed by the streams namespace prefix. The error child shall be sent by a compliant entity if it perceives that a stream-level error has occurred.

5.8.1. Rules

[TOC](#)

The following rules apply to stream-level errors.

5.8.1.1. Stream Errors Are Unrecoverable

[TOC](#)

Stream-level errors are unrecoverable. Therefore, if an error occurs at the level of the stream, the entity that detects the error MUST send a <error/> element with an appropriate child element that specifies the error condition and at the same time send a closing </stream> tag.

C: <message><body></message>

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

The entity that generates the stream error then SHOULD immediately terminate the underlying TCP connection, although it MAY wait until after it receives a closing </stream> tag from the entity to which it sent the stream error.

C: </stream:stream>

5.8.1.2. Stream Errors Can Occur During Setup

[TOC](#)

If the error is triggered by the initial stream header, the receiving entity MUST still send the opening <stream> tag, include the <error/> element as a child of the stream element, and send the closing </stream> tag (preferably all at the same time).

```

C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://wrong.namespace.example.org/'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
  <stream:error>
    <invalid-namespace
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  </stream:error>
</stream:stream>

```

5.8.1.3. Stream Errors When the Host is Unspecified or Unknown

[TOC](#)

If the initiating entity provides no 'to' attribute or provides an unknown host in the 'to' attribute and the error occurs during stream setup, the receiving entity SHOULD provide an authoritative hostname in the 'from' attribute of the stream header sent before termination, but absent such an authoritative hostname MAY instead simply populate the response stream's 'from' attribute with the value of the initial stream header's 'to' attribute (where the value of the 'from' attribute MAY be empty if the initiating entity provided no 'to' attribute).


```

C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='unknown.host.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:error>
    <host-unknown
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
    </stream:error>
  </stream:stream>

```

5.8.2. Syntax

[TOC](#)

The syntax for stream errors is as follows, where "defined-condition" is a placeholder for one of the conditions defined under [Section 5.8.3 \(Defined Stream Error Conditions\)](#).

```

<stream:error>
  <defined-condition xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  [<text xmlns='urn:ietf:params:xml:ns:xmpp-streams'
    xml:lang='langcode'>
    [ ... descriptive text ... ]
  </text>]
  [application-specific condition element]
</stream:error>

```

The <error/> element:

*MUST contain a child element corresponding to one of the [defined stream error conditions \(Defined Stream Error Conditions\)](#); this element MUST be qualified by the 'urn:ietf:params:xml:ns:xmpp-streams' namespace.

*MAY contain a `<text/>` child element containing XML character data that describes the error in more detail; this element MUST be qualified by the `'urn:ietf:params:xml:ns:xmpp-streams'` namespace and SHOULD possess an `'xml:lang'` attribute specifying the natural language of the XML character data.

*MAY contain a child element for an application-specific error condition; this element MUST be qualified by an application-defined namespace, and its structure is defined by that namespace (see [Section 5.8.4 \(Application-Specific Conditions\)](#)).

The `<text/>` element is OPTIONAL. If included, it MUST be used only to provide descriptive or diagnostic information that supplements the meaning of a defined condition or application-specific condition. It MUST NOT be interpreted programmatically by an application. It MUST NOT be used as the error message presented to a human user, but MAY be shown in addition to the error message associated with the defined condition element (and, optionally, the application-specific condition element).

5.8.3. Defined Stream Error Conditions

[TOC](#)

The following stream-level error conditions are defined.

5.8.3.1. bad-format

[TOC](#)

The entity has sent XML that cannot be processed.
(In the following example, the client sends an XMPP message that is not well-formed XML.)

```
C: <message>
    <body>No closing body tag!
</message>

S: <stream:error>
    <bad-format
        xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
</stream:error>
</stream:stream>
```

This error MAY be used instead of the more specific XML-related errors, such as `<bad-namespace-prefix/>`, `<invalid-xml/>`, `<restricted-xml/>`, `<unsupported-encoding/>`, and `<xml-not-well-formed/>`. However, the more specific errors are RECOMMENDED.

5.8.3.2. bad-namespace-prefix

[TOC](#)

The entity has sent a namespace prefix that is unsupported, or has sent no namespace prefix on an element that requires such a prefix (see [Section 12.2 \(XML Namespace Names and Prefixes\)](#)). (In the following example, the client specifies a namespace prefix of "foobar" for the XML streams namespace.)

```
C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xmlns='jabber:client'
    xmlns:foobar='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:error>
    <bad-namespace-prefix
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  </stream:error>
</stream:stream>
```

5.8.3.3. conflict

[TOC](#)

The server is either (1) closing the existing stream for this entity because a new stream has been initiated that conflicts with the existing stream, or (2) is refusing a new stream for this entity because allowing the new stream would conflict with an existing stream (e.g., because the server allows only a certain number of connections from the same IP address).

```

C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:error>
    <conflict
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  </stream:error>
</stream:stream>

```

5.8.3.4. connection-timeout

[TOC](#)

The entity has not generated any traffic over the stream for some period of time (configurable according to a local service policy) and therefore the connection is being dropped.

```

P: <stream:error>
  <connection-timeout
    xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
</stream:error>
</stream:stream>

```

5.8.3.5. host-gone

[TOC](#)

The value of the 'to' attribute provided in the initial stream header corresponds to a hostname that is no longer serviced by the receiving entity.

(In the following example, the peer specifies a 'to' address of "foo.im.example.com" when connecting to the "im.example.com" server, but the server no longer hosts a service at that address.)

```
P: <?xml version='1.0'?>
  <stream:stream
    from='example.net'
    to='foo.im.example.com'
    version='1.0'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='g4qSvGvBxJ+xeAd7QKez0QJFFlw='
    to='example.net'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:error>
    <host-gone
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  </stream:error>
</stream:stream>
```

5.8.3.6. host-unknown

[TOC](#)

The value of the 'to' attribute provided in the initial stream header does not correspond to a hostname that is serviced by the receiving entity.

(In the following example, the peer specifies a 'to' address of "example.org" when connecting to the "im.example.com" server, but the server knows nothing of that address.)

```

P: <?xml version='1.0'?>
  <stream:stream
    from='example.net'
    to='example.org'
    version='1.0'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='g4qSvGvBxJ+xeAd7QKez0QJFFlw='
    to='example.net'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:error>
    <host-unknown
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  </stream:error>
</stream:stream>

```

5.8.3.7. improper-addressing

[TOC](#)

A stanza sent between two servers lacks a 'to' or 'from' attribute, the 'from' or 'to' attribute has no value, or the value is not a valid XMPP address.

(In the following example, the peer sends a stanza without a 'to' address.)

```

P: <message from='juliet@im.example.com'>
  <body>Wherefore art thou?</body>
</message>

S: <stream:error>
  <improper-addressing
    xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  </stream:error>
</stream:stream>

```

[TOC](#)

5.8.3.8. internal-server-error

The server has experienced a misconfiguration or an otherwise-undefined internal error that prevents it from servicing the stream.

```
S: <stream:error>
    <internal-server-error
        xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
    </stream:error>
</stream:stream>
```

5.8.3.9. invalid-from

[TOC](#)

The JID or hostname provided in a 'from' address is not a valid JID or does not match an authorized JID or validated domain as negotiated between servers via SASL or server dialback, or as negotiated between a client and a server via authentication and resource binding. (In the following example, a peer that has authenticated only as "example.net" attempts to send a stanza from an address at "example.org".)

```
P: <message from='romeo@example.org' to='juliet@im.example.com'>
    <body>Neither, fair saint, if either thee dislike.</body>
</message>

S: <stream:error>
    <invalid-from
        xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
    </stream:error>
</stream:stream>
```

5.8.3.10. invalid-id

[TOC](#)

The stream ID or server dialback ID is invalid or does not match an ID previously provided.

(In the following example, the server dialback ID is invalid; see [\[XEP-0220\] \(Saint-Andre, P. and J. Miller, "Server Dialback," October 2008.\)](#).)

```

P: <db:verify
    from='example.net'
    to='im.example.com'
    id='unknown-id'
    type='invalid'/>

S: <stream:error>
    <invalid-id
        xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
    </stream:error>
</stream:stream>

```

5.8.3.11. invalid-namespace

[TOC](#)

The streams namespace name is something other than "http://etherx.jabber.org/streams" (see [Section 12.2 \(XML Namespace Names and Prefixes\)](#)) or the default namespace is not supported (e.g., something other than "jabber:client" or "jabber:server"). (In the following example, the client specifies a streams namespace of 'http://wrong.namespace.example.org/'.)

```

C: <?xml version='1.0'?>
    <stream:stream
        from='juliet@im.example.com'
        to='im.example.com'
        version='1.0'
        xmlns='jabber:client'
        xmlns:stream='http://wrong.namespace.example.org/'>

S: <?xml version='1.0'?>
    <stream:stream
        from='im.example.com'
        id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
        to='juliet@im.example.com'
        version='1.0'
        xml:lang='en'
        xmlns='jabber:client'
        xmlns:stream='http://etherx.jabber.org/streams'>
    <stream:error>
        <invalid-namespace
            xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
        </stream:error>
    </stream:stream>

```

5.8.3.12. **invalid-xml**

[TOC](#)

The entity has sent invalid XML over the stream to a server that performs validation (see [Section 12.4 \(Validation\)](#)). (In the following example, the peer attempts to send an IQ stanza of type "subscribe" but the XML schema defines no such value for the 'type' attribute.)

```
P: <iq from='example.net'
    id='some-id'
    to='im.example.com'
    type='subscribe'>
  <ping xmlns='urn:xmpp:ping' />
</iq>

S: <stream:error>
  <invalid-xml
    xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
</stream:error>
</stream:stream>
```

5.8.3.13. **not-authorized**

[TOC](#)

The entity has attempted to send XML stanzas before the stream has been authenticated, or otherwise is not authorized to perform an action related to stream negotiation; the receiving entity MUST NOT process the offending stanza before sending the stream error. (In the following example, the client attempts to send XML stanzas before authenticating with the server.)

```

C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'

C: <message to='romeo@example.net'>
  <body>Wherefore art thou?</body>
</message>

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

```

5.8.3.14. policy-violation

[TOC](#)

The entity has violated some local service policy (e.g., the stanza exceeds a configured size limit); the server MAY choose to specify the policy in the <text/> element or in an application-specific condition element.

(In the following example, the client sends an XMPP message that is too large according to the server's local service policy.)

```

C: <message to='juliet@im.example.com' id='foo'>
  <body>[ ... the-emacs-manual ... ]</body>
</message>

S: <stream:error>
  <policy-violation
    xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
</stream:error>

S: </stream:stream>

```

5.8.3.15. remote-connection-failed

[TOC](#)

The server is unable to properly connect to a remote entity that is required for authentication or authorization, such as a remote authentication database or (in server dialback) the authoritative server.

```

C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:error>
    <remote-connection-failed
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  </stream:error>
</stream:stream>

```

[TOC](#)

5.8.3.16. resource-constraint

The server lacks the system resources necessary to service the stream.

```
C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:error>
    <resource-constraint
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  </stream:error>
</stream:stream>
```

5.8.3.17. restricted-xml

[TOC](#)

The entity has attempted to send restricted XML features such as a comment, processing instruction, DTD subset, or XML entity reference (see [Section 12.1 \(Restrictions\)](#)). (In the following example, the client sends an XMPP message containing an XML comment.)

```
C: <message to='juliet@im.example.com'>
  <!--<subject/-->
  <body>This message has no subject.</body>
</message>

S: <stream:error>
  <restricted-xml
    xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
</stream:error>
</stream:stream>
```

5.8.3.18. see-other-host

[TOC](#)

The server will not provide service to the initiating entity but is redirecting traffic to another host; the XML character data of the <see-other-host/> element returned by the server SHOULD specify the alternate hostname or IP address at which to connect, which SHOULD be a valid domain identifier but MAY also include a port number. When it receives a see-other-host stream error, the initiating entity SHOULD cleanly handle the disconnection and then reconnect to the host specified in the <see-other-host/> element; if no port is specified, the initiating entity SHOULD perform a [\[DNS-SRV\] \(Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services \(DNS SRV\)," February 2000.\)](#) lookup on the provided domain identifier but MAY assume that it can connect to that domain identifier at the standard XMPP ports (i.e., 5222 for client-to-server connections and 5269 for server-to-server connections).

```
C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <stream:error>
    <see-other-host
      xmlns='urn:ietf:params:xml:ns:xmpp-streams'>
      im.example.com:9090
    </see-other-host>
  </stream:error>
</stream:stream>
```

[TOC](#)

5.8.3.19. system-shutdown

The server is being shut down and all active streams are being closed.

```
S: <stream:error>
    <system-shutdown
        xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
    </stream:error>
</stream:stream>
```

5.8.3.20. undefined-condition

[TOC](#)

The error condition is not one of those defined by the other conditions in this list; this error condition SHOULD be used only in conjunction with an application-specific condition.

```
S: <stream:error>
    <undefined-condition
        xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
    <app-error xmlns='http://example.com/ns' />
</stream:error>
</stream:stream>
```

5.8.3.21. unsupported-encoding

[TOC](#)

The initiating entity has encoded the stream in an encoding that is not supported by the server (see [Section 12.6 \(Character Encoding\)](#)) or has otherwise improperly encoded the stream (e.g., by violating the rules of the [\[UTF-8\] \(Yergeau, F., "UTF-8, a transformation format of ISO 10646," November 2003.\)](#) encoding).

(In the following example, the client attempts to encode data using UTF-16 instead of UTF-8.)

```

C: <?xml version='1.0' encoding='UTF-16'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
  <stream:error>
    <unsupported-encoding
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
  </stream:error>
</stream:stream>

```

5.8.3.22. unsupported-stanza-type

[TOC](#)

The initiating entity has sent a first-level child of the stream that is not supported by the server or consistent with the default namespace.

(In the following example, the client attempts to send an XML stanza of <pubsub/> when the default namespace is "jabber:client".)

```

C: <pubsub>
  <publish node='princely_musings'>
    <item id='ae890ac52d0df67ed7cfd51b644e901'>
      <entry xmlns='http://www.w3.org/2005/Atom'>
        <title>Soliloquy</title>
        <summary>
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them?
        </summary>
        <link rel='alternate' type='text/html'
          href='http://denmark.example/2003/12/13/atom03' />
        <id>tag:denmark.example,2003:entry-32397</id>
        <published>2003-12-13T18:30:02Z</published>
        <updated>2003-12-13T18:30:02Z</updated>
      </entry>
    </item>
  </publish>
</pubsub>

S: <stream:error>
  <unsupported-stanza-type
    xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
</stream:error>
</stream:stream>

```

5.8.3.23. unsupported-version

[TOC](#)

The value of the 'version' attribute provided by the initiating entity in the stream header specifies a version of XMPP that is not supported by the server; the server MAY specify the version(s) it supports in the <text/> element.

(In the following example, the client specifies an XMPP version of "11.0" but the server supports only version "1.0" and "1.1".)


```

C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
  <stream:error>
    <unsupported-version
      xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
    <text xmlns='urn:ietf:params:xml:ns:xmpp-streams'>
      1.0, 1.1
    </text>
  </stream:error>
</stream:stream>

```

5.8.3.24. xml-not-well-formed

[TOC](#)

The initiating entity has sent XML that violates the well-formedness rules of [\[XML\]](#) (Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, “Extensible Markup Language (XML) 1.0 (Fourth Edition),” August 2006.) or [\[XML-NAMES\]](#) (Layman, A., Hollander, D., Tobin, R., and T. Bray, “Namespaces in XML 1.1 (Second Edition),” August 2006.).

(In the following example, the client sends an XMPP message that is not well-formed XML.)

```

C: <message>
    <body>No closing body tag!
</message>

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

```

5.8.4. Application-Specific Conditions

[TOC](#)

As noted, an application MAY provide application-specific stream error information by including a properly-namespaced child in the error element. The application-specific element SHOULD supplement or further qualify a defined element. Thus the <error/> element will contain two or three child elements.

```

C: <message>
    <body>
        My keyboard layout is:

        QWERTYUIOP{}|
        ASDFGHJKL:"
        ZXCVBNM<>?
    </body>
</message>

S: <stream:error>
    <xml-not-well-formed
        xmlns='urn:ietf:params:xml:ns:xmpp-streams' />
    <text xml:lang='en' xmlns='urn:ietf:params:xml:ns:xmpp-streams'>
        Some special application diagnostic information!
    </text>
    <escape-your-data xmlns='http://example.com/ns' />
</stream:error>
</stream:stream>

```

[TOC](#)

5.9. Simplified Stream Examples

This section contains two simplified examples of a stream-based connection between a client and a server; these examples are included for the purpose of illustrating the concepts introduced thus far.

A basic connection:

```
C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'

S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>

[ ... channel encryption ... ]

[ ... authentication ... ]

[ ... resource binding ... ]

C:  <message from='juliet@im.example.com/balcony'
      to='romeo@example.net'
      xml:lang='en'>
    <body>Art thou not Romeo, and a Montague?</body>
  </message>

S:  <message from='romeo@example.net/orchard'
      to='juliet@im.example.com/balcony'
      xml:lang='en'>
    <body>Neither, fair saint, if either thee dislike.</body>
  </message>

C: </stream:stream>

S: </stream:stream>
```

A connection gone bad:

```
C: <?xml version='1.0'?>
  <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

```
S: <?xml version='1.0'?>
  <stream:stream
    from='im.example.com'
    id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

[... channel encryption ...]

[... authentication ...]

[... resource binding ...]

```
C:  <message from='juliet@im.example.com/balcony'
      to='romeo@example.net'
      xml:lang='en'>
    <body>No closing body tag!
  </message>
```

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

More detailed examples are provided under [Section 10 \(Examples\)](#).

6. STARTTLS Negotiation

[TOC](#)

6.1. Overview

[TOC](#)

XMPP includes a method for securing the stream from tampering and eavesdropping. This channel encryption method makes use of the Transport Layer Security [\[TLS\]](#) ([Dierks, T. and E. Rescorla, "The Transport Layer Security \(TLS\) Protocol Version 1.2," August 2008.](#)) protocol, specifically a "STARTTLS" extension that is modelled after similar extensions for the [\[IMAP\]](#) ([Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1," March 2003.](#)), [\[POP3\]](#) ([Myers, J. and M. Rose, "Post Office Protocol - Version 3," May 1996.](#)), and [\[ACAP\]](#) ([Newman, C. and J. Myers, "ACAP -- Application Configuration Access Protocol," November 1997.](#)) protocols as described in [\[USINGTLS\]](#) ([Newman, C., "Using TLS with IMAP, POP3 and ACAP," June 1999.](#)). The XML namespace name for the STARTTLS extension is 'urn:ietf:params:xml:ns:xmpp-tls'.

Support for STARTTLS is REQUIRED in XMPP client and server implementations. An administrator of a given deployment MAY require the use of TLS for client-to-server communication, server-to-server communication, or both. A deployed client SHOULD use TLS to secure its stream with a server prior to attempting the completion of [SASL negotiation](#) ([SASL Negotiation](#)), and deployed servers SHOULD use TLS between two domains for the purpose of securing server-to-server communication.

6.2. Rules

[TOC](#)

6.2.1. Data Formatting

[TOC](#)

During STARTTLS negotiation, the entities MUST NOT send any whitespace within the root stream element as separators between XML elements (i.e., from the last character of the <starttls/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace at depth=1 of the stream as sent by the initiating entity until the last character of the <proceed/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace at depth=1 of the stream as sent by the receiving entity). This prohibition helps to ensure proper security layer byte precision. Any such whitespace shown in the STARTTLS examples provided in this document is included only for the sake of readability.

[TOC](#)

6.2.2. Order of Negotiation

If the initiating entity chooses to use TLS, STARTTLS negotiation MUST be completed before proceeding to [SASL negotiation \(SASL Negotiation\)](#); this order of negotiation is required 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 (or other credentials) provided during prior TLS negotiation.

6.3. Process

[TOC](#)

6.3.1. Exchange of Stream Headers and Stream Features

[TOC](#)

The initiating entity resolves the hostname of the receiving entity as specified under [Section 4 \(TCP Binding\)](#), opens a TCP connection to the advertised port at the resolved IP address, and sends an initial stream header to the receiving entity; if the initiating entity is capable of STARTTLS negotiation, it MUST include the 'version' attribute set to a value of at least "1.0" in the initial stream header.

```
I: <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

The receiving entity MUST send a response stream header to the initiating entity over the TCP connection opened by the initiating entity; if the receiving entity is capable of STARTTLS negotiation, it MUST include the 'version' attribute set to a value of at least "1.0" in the response stream header.

```
R: <stream:stream
  from='im.example.com'
  id='t7AMCin9zjMNwQKDnplntZPIDEI='
  to='juliet@im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'
```

The receiving entity then MUST send stream features to the initiating entity. If the receiving entity supports TLS, the stream features MUST include an advertisement for support of STARTTLS negotiation, i.e., a <starttls/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace.

If the receiving entity considers STARTTLS negotiation to be discretionary, the <starttls/> element MUST contain an empty <optional/> child element.

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

If the receiving entity considers STARTTLS negotiation to be mandatory, the <starttls/> element MUST contain an empty <required/> child element.

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

6.3.2. Initiation of STARTTLS Negotiation

[TOC](#)

6.3.2.1. STARTTLS Command

[TOC](#)

In order to begin the STARTTLS negotiation, the initiating entity issues 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.

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

The receiving entity MUST reply with either a <proceed/> element (proceed case) or a <failure/> element (failure case) qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace.

6.3.2.2. Failure Case

[TOC](#)

If the failure case occurs, the receiving entity MUST 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.

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

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

Causes for the failure case include but are not limited to:

1. The initiating entity has sent a malformed STARTTLS command.
2. The receiving entity does not offer STARTTLS negotiation either temporarily or permanently.
3. The receiving entity cannot complete STARTTLS negotiation because of an internal error.

Note: STARTTLS failure is not triggered by TLS errors such as bad certificate or unknown certificate authority; those errors are generated and handled during the TLS negotiation itself as described in [\[TLS\] \(Dierks, T. and E. Rescorla, "The Transport Layer Security \(TLS\) Protocol Version 1.2," August 2008.\)](#).

If the failure case occurs, the initiating entity MAY attempt to reconnect as explained under [Section 4.5 \(Reconnection\)](#).

6.3.2.3. Proceed Case

[TOC](#)

If the proceed case occurs, the receiving entity MUST return a <proceed/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace.

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

The receiving entity MUST consider the TLS negotiation to have begun immediately after sending the closing '>' character of the <proceed/> element to the initiating entity. The initiating entity MUST consider the TLS negotiation to have begun immediately after receiving the closing '>' character of the <proceed/> element from the receiving entity.

The entities now proceed to TLS negotiation as explained in the next section.

6.3.3. TLS Negotiation

[TOC](#)

6.3.3.1. Rules

[TOC](#)

In order to complete TLS negotiation over the TCP connection, the entities MUST follow the process defined in [\[TLS\] \(Dierks, T. and E. Rescorla, "The Transport Layer Security \(TLS\) Protocol Version 1.2," August 2008.\)](#).

The following rules apply:

1. The entities MUST NOT send any further XML data until the TLS negotiation has either failed or succeeded.
2. The receiving entity MUST present a certificate.
3. The receiving entity SHOULD send a certificate request to the initiating entity so that mutual authentication will be possible.
4. The initiating entity MUST validate the certificate to determine if the TLS negotiation shall succeed; see [Section 15.2.2 \(Certificate Validation\)](#) regarding certificate validation procedures.
5. The receiving entity SHOULD choose which certificate to present based on the 'to' attribute of the initial stream header.

Note: See [Section 15.6 \(Mandatory-to-Implement Technologies\)](#) regarding ciphers that MUST be supported for TLS; naturally, other ciphers MAY be supported as well.

6.3.3.2. TLS Failure

[TOC](#)

If the TLS negotiation results in failure, the receiving entity MUST terminate the TCP connection.

The receiving entity MUST NOT send a closing `</stream>` tag before terminating the TCP connection, since the receiving entity and initiating entity MUST consider the original stream to be replaced upon failure of the TLS negotiation.

6.3.3.3. TLS Success

[TOC](#)

If the TLS negotiation is successful, then the entities MUST proceed as follows.

1. The receiving entity MUST discard any knowledge obtained in an insecure manner from the initiating entity before TLS took effect.
2. The initiating entity MUST discard any knowledge obtained in an insecure manner from the receiving entity before TLS took effect.
3. The initiating entity MUST send a new initial stream header to the receiving entity over the encrypted connection.

```
I: <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

Note: The initiating entity MUST 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.

4. The receiving entity MUST respond with a new response stream header over the encrypted connection.

```
R: <stream:stream
    from='im.example.com'
    id='vgKi/bkYME80Aj4rlXMkpucAqe4='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
```

5. The receiving entity also MUST send stream features to the initiating entity, which MUST NOT include the STARTTLS feature but which SHOULD include the SASL stream feature as described under [Section 7 \(SASL Negotiation\)](#).

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

7. SASL Negotiation

[TOC](#)

7.1. Overview

[TOC](#)

XMPP includes a method for authenticating a stream by means of an XMPP-specific profile of the Simple Authentication and Security Layer protocol (see [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#)). SASL provides a generalized method for adding authentication support to connection-based protocols, and XMPP uses an XML namespace profile of SASL that conforms to the profiling requirements of [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#).

Support for SASL negotiation is REQUIRED in XMPP client and server implementations.

7.2. Rules

[TOC](#)

7.2.1. Mechanism Preferences

[TOC](#)

Any entity that will act as a SASL client or a SASL server MUST maintain an ordered list of its preferred SASL mechanisms according to the client or server, where the list is ordered by the perceived strength of the mechanisms. A server MUST offer and a client MUST try SASL mechanisms in the order of their perceived strength. For example, if the server offers the ordered list "PLAIN DIGEST-MD5 GSSAPI" or "DIGEST-MD5 GSSAPI PLAIN" but the client's ordered list is "GSSAPI DIGEST-MD5", the client shall try GSSAPI first and then DIGEST-MD5 but shall never try PLAIN (since PLAIN is not on its list).

7.2.2. Mechanism Offers

[TOC](#)

If the receiving entity considers [TLS negotiation \(STARTTLS Negotiation\)](#) to be mandatory before use of a particular SASL authentication mechanism will be acceptable, the receiving entity MUST NOT advertise that mechanism in its list of available SASL authentication mechanisms prior to successful TLS negotiation. If during prior TLS negotiation the initiating entity presented a certificate that is acceptable to the receiving entity for purposes of strong identity verification in accordance with local service policies, the receiving entity MUST offer the SASL EXTERNAL mechanism to the initiating entity during SASL negotiation (refer to [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#)) and SHOULD prefer that mechanism. However, the EXTERNAL mechanism MAY be offered under other circumstances as well. See [Section 15.6 \(Mandatory-to-Implement Technologies\)](#) regarding mechanisms that MUST be supported; naturally, other SASL mechanisms MAY be supported as well. Best practices for the use of several SASL mechanisms in the context of XMPP are described in [\[XEP-0175\] \(Saint-Andre, P., "Best Practices for Use of SASL ANONYMOUS," September 2006.\)](#) and [\[XEP-0178\] \(Saint-Andre, P. and P. Millard, "Best Practices for Use of SASL EXTERNAL with Certificates," February 2007.\)](#).

[TOC](#)

7.2.3. Data Formatting

The following data formatting rules apply to the SASL negotiation:

1. During SASL negotiation, the entities MUST NOT send any whitespace within the root stream element as separators between XML elements (i.e., from the last character of the <auth/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace at depth=1 of the stream as sent by the initiating entity until the last character of the <success/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace at depth=1 of the stream as sent by the receiving entity). This prohibition helps to ensure proper security layer byte precision. Any such whitespace shown in the SASL examples provided in this document is included only for the sake of readability.
2. Any XML character data contained within the XML elements MUST be encoded using base64, where the encoding adheres to the definition in Section 4 of [\[BASE64\] \(Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," October 2006.\)](#) and where the padding bits are set to zero.
3. As formally specified in the XML schema for the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace under [Appendix C.4 \(SASL Namespace\)](#), the receiving entity MAY include one or more application-specific child elements inside the <mechanisms/> element to provide information that might be needed by the initiating entity in order to complete successful SASL negotiation using one or more of the offered mechanisms; however, the syntax and semantics of all such elements are out of scope for this specification.

7.2.4. Security Layers

[TOC](#)

Upon successful SASL negotiation that involves negotiation of a security layer, both the initiating entity and the receiving MUST discard any application-layer state (i.e, state from the XMPP layer, excluding state from the TLS negotiation or SASL negotiation).

[TOC](#)

7.2.5. Simple Usernames

It is possible that provision of a "simple username" is supported by the selected SASL mechanism (e.g., this is supported by the DIGEST-MD5 and CRAM-MD5 mechanisms but not by the EXTERNAL and GSSAPI mechanisms). The simple username provided during authentication MUST be as follows:

Client-to-server communication: The initiating entity's registered account name, i.e., a user name or node name as described under [Section 3.3 \(Node Identifier\)](#) (this is not a bare JID of the form <node@domain> but only the node portion of the JID). The simple username MUST adhere to the [Nodeprep \(Nodeprep\) profile of \[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

Server-to-server communication: The initiating entity's sending domain, i.e., IP address or fully qualified domain name as contained in an XMPP domain identifier. The simple username MUST adhere to the [\[NAMEPREP\] \(Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names \(IDN\)," March 2003.\)](#) profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

7.2.6. Authorization Identities

[TOC](#)

If the initiating entity wishes to act on behalf of another entity and the selected SASL mechanism supports transmission of an authorization identity, the initiating entity MUST provide an authorization identity during SASL negotiation. If the initiating entity does not wish to act on behalf of another entity, it MUST NOT provide an authorization identity. As specified in [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#), the initiating entity MUST NOT provide an authorization identity unless the authorization identity is different from the default authorization identity derived from the authentication identity. If provided, the value of the authorization identity MUST be a bare JID of the form <domain> (i.e., an XMPP domain identifier only) for servers and a bare JID of the form <node@domain> (i.e., node identifier and domain identifier) for clients.

Note: The authorization identity communicated during SASL negotiation is used to determine the canonical address for the initiating client or server according to the receiving server, as described under [Section 3.5 \(Determination of Addresses\)](#).

7.2.7. Realms

[TOC](#)

The receiving entity MAY include a realm when negotiating certain SASL mechanisms. If the receiving entity does not communicate a realm, the initiating entity MUST NOT assume that any realm exists. The realm MUST be used only for the purpose of authentication; in particular, an initiating entity MUST NOT attempt to derive an XMPP hostname from the realm information provided by the receiving entity.

7.2.8. Round Trips

[TOC](#)

[\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#) specifies that a using protocol such as XMPP can define two methods by which the protocol can save round trips where allowed for the SASL mechanism:

1. When the SASL client (the XMPP "initiating entity") requests an authentication exchange, it can include "initial response" data with its request if appropriate for the SASL mechanism in use. In XMPP this is done by including the initial response as the XML character data of the <auth/> element.
2. At the end of the authentication exchange, the SASL server (the XMPP "receiving entity") can include "additional data with success" if appropriate for the SASL mechanism in use. In XMPP this is done by including the additional data as the XML character data of the <success/> element.

For the sake of protocol efficiency, it is RECOMMENDED for XMPP clients and servers to use these methods, however they MUST support the less efficient modes as well.

7.3. Process

[TOC](#)

The process for SASL negotiation is as follows.

[TOC](#)

7.3.1. Exchange of Stream Headers and Stream Features

If SASL negotiation follows successful [STARTTLS negotiation \(STARTTLS Negotiation\)](#), then the SASL negotiation occurs over the encrypted stream that has already been negotiated. If not, the initiating entity resolves the hostname of the receiving entity as specified under [Section 4 \(TCP Binding\)](#), opens a TCP connection to the advertised port at the resolved IP address, and sends an initial stream header to the receiving entity; if the initiating entity is capable of STARTTLS negotiation, it MUST include the 'version' attribute set to a value of at least "1.0" in the initial stream header.

```
I: <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

The receiving entity MUST send a response stream header to the initiating entity; if the receiving entity is capable of SASL negotiation, it MUST include the 'version' attribute set to a value of at least "1.0" in the response stream header.

```
R: <stream:stream
    from='im.example.com'
    id='vgKi/bkYME80Aj4rlXMkpucAqe4='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
```

The receiving entity also MUST send stream features to the initiating entity. If the receiving entity supports SASL, the stream features MUST include an advertisement for support of SASL negotiation, i.e., a `<mechanisms/>` element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace.

The `<mechanisms/>` element MUST contain one `<mechanism/>` child element for each authentication mechanism the receiving entity offers to the initiating entity. The order of `<mechanism/>` elements in the XML indicates the preference order of the SASL mechanisms according to the receiving entity; however the initiating entity MUST maintain its own preference order independent of the preference order of the receiving entity.


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

If the receiving entity considers SASL negotiation to be discretionary, the <mechanisms/> element MUST contain an empty <optional/> child element.

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

If the receiving entity considers SASL negotiation to be mandatory, the <mechanisms/> element MUST contain an empty <required/> child element.

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

7.3.2. Initiation

[TOC](#)

In order to begin the SASL negotiation, the initiating entity sends an <auth/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace and includes an appropriate value for the 'mechanism' attribute. This element MAY contain XML character data (in SASL terminology, the "initial response") if the mechanism supports or requires it; if the initiating entity needs to send a zero-length initial response, it MUST transmit the response as a single equals sign character ("="), which indicates that the response is present but contains no data.

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

7.3.3. Challenge-Response Sequence

[TOC](#)

If necessary, the receiving entity challenges the initiating entity by sending a <challenge/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace; this element MAY contain XML character data (which MUST be generated in accordance with the definition of the SASL mechanism chosen by the initiating entity). The initiating entity responds to the challenge by sending a <response/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace; this element MAY contain XML character data (which MUST be generated in accordance with the definition of the SASL mechanism chosen by the initiating entity). If necessary, the receiving entity sends more challenges and the initiating entity sends more responses. This series of challenge/response pairs continues until one of three things happens:

- *The initiating entity aborts the handshake.
- *The receiving entity reports failure of the handshake.
- *The receiving entity reports success of the handshake.

These scenarios are described in the following sections.

7.3.4. Abort

[TOC](#)

The initiating entity aborts the handshake by sending an <abort/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace.

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

Upon receiving an <abort/> element, the receiving entity MUST return a <failure/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace and containing an <aborted/> child element.

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

7.3.5. Failure

[TOC](#)

The receiving entity reports failure of the handshake by sending a `<failure/>` element qualified by the `'urn:ietf:params:xml:ns:xmpp-sasl'` namespace (the particular cause of failure MUST be communicated in an appropriate child element of the `<failure/>` element as defined under [Section 7.4 \(SASL Errors\)](#)).

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

Where appropriate for the chosen SASL mechanism, the receiving entity SHOULD allow a configurable but reasonable number of retries (at least 2 and no more than 5); this enables the initiating entity (e.g., an end-user client) to tolerate incorrectly-provided credentials (e.g., a mistyped password) without being forced to reconnect.

If the initiating entity attempts a reasonable number of retries with the same SASL mechanism and all attempts fail, it MAY fall back to the next mechanism in its ordered list by sending a new `<auth/>` request to the receiving entity. If there are no remaining mechanisms in its list, the initiating entity SHOULD instead send an `<abort/>` element to the receiving entity.

If the initiating entity exceeds the number of retries, the receiving entity MUST return a stream error (which SHOULD be `<policy-violation/>` but MAY be `<not-authorized/>`).

7.3.6. Success

[TOC](#)

The receiving entity reports success of the handshake by sending a `<success/>` element qualified by the `'urn:ietf:params:xml:ns:xmpp-sasl'` namespace; this element MAY contain XML character data (in SASL terminology, "additional data with success") if the chosen SASL mechanism supports or requires it; if the receiving entity needs to send additional data of zero length, it MUST transmit the data as a single equals sign character (`"="`).

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

Note: The authorization identity communicated during SASL negotiation is used to determine the canonical address for the initiating client

or server according to the receiving server, as described under [Section 3.5 \(Determination of Addresses\)](#).

Upon receiving the <success/> element, the initiating entity MUST initiate a new stream over the existing TCP connection by sending a new initial stream header to the receiving entity.

```
I: <stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
```

Note: The initiating entity MUST 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.

Upon receiving the new initial stream header from the initiating entity, the receiving entity MUST respond by sending a new response XML stream header to the initiating entity.

```
R: <stream:stream
    from='im.example.com'
    id='gPybza0zBmaADgxKXu9UC1bprp0='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

The receiving entity MUST also send stream features, containing any further available features or containing no features (via an empty <features/> element).

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

7.4. SASL Errors

The syntax of SASL errors is as follows:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <defined-condition/>
  [<text xml:lang='langcode'>
    OPTIONAL descriptive text
  </text>]
</failure>
```

Where "defined-condition" is one of the SASL-related error conditions defined in the following sections.

Inclusion of a defined condition is REQUIRED.

Inclusion of the <text/> element is OPTIONAL, and can be used to provide application-specific information about the error condition, which information MAY be displayed to a human but only as a supplement to the defined condition.

7.4.1. aborted

[TOC](#)

The receiving entity acknowledges an <abort/> element sent by the initiating entity; sent in reply to the <abort/> element.

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

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

7.4.2. account-disabled

[TOC](#)

The account of the initiating entity has been temporarily disabled; sent in reply to an <auth/> element (with or without initial response data) or a <response/> element.

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

R: <failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <account-disabled/>
  <text xml:lang='en'>Call 212-555-1212 for assistance.</text>
</failure>
```

7.4.3. credentials-expired

[TOC](#)

The authentication failed because the initiating entity provided credentials that have expired; sent in reply to a <response/> element or an <auth/> element with initial response data.

```
I: <response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    [ ... ]
</response>
```

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

7.4.4. encryption-required

[TOC](#)

The mechanism requested by the initiating entity cannot be used unless the underlying stream is encrypted; sent in reply to an <auth/> element (with or without initial response data).

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

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

7.4.5. incorrect-encoding

[TOC](#)

The data provided by the initiating entity could not be processed because the [\[BASE64\] \(Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," October 2006.\)](#) encoding is incorrect (e.g., because the encoding does not adhere to the definition in Section 4 of [\[BASE64\] \(Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," October 2006.\)](#)); sent in reply to a <response/> element or an <auth/> element with initial response data.

```
I: <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='DIGEST-MD5'>[ ... ]</auth>

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

7.4.6. invalid-authzid

[TOC](#)

The authzid provided by the initiating entity is invalid, either because it is incorrectly formatted or because the initiating entity does not have permissions to authorize that ID; sent in reply to a <response/> element or an <auth/> element with initial response data.

```
I: <response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    [ ... ]
  </response>

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

7.4.7. invalid-mechanism

[TOC](#)

The initiating entity did not provide a mechanism or requested a mechanism that is not supported by the receiving entity; sent in reply to an <auth/> element.

```
I: <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='CRAM-MD5' />

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

[TOC](#)

7.4.8. malformed-request

The request is malformed (e.g., the <auth/> element includes initial response data but the mechanism does not allow that, or the data sent violates the syntax for the specified SASL mechanism); sent in reply to an <abort/>, <auth/>, <challenge/>, or <response/> element.

(In the following example, the XML character data of the <auth/> element contains more than 255 UTF-8-encoded Unicode characters and therefore violates the "token" production for the SASL ANONYMOUS mechanism as specified in [\[ANONYMOUS\] \(Zeilenga, K., "Anonymous Simple Authentication and Security Layer \(SASL\) Mechanism," June 2006.\)](#).)

```
I: <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='ANONYMOUS'>[ ... some-long-token ... ]</auth>

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

7.4.9. mechanism-too-weak

[TOC](#)

The mechanism requested by the initiating entity is weaker than server policy permits for that initiating entity; sent in reply to an <auth/> element (with or without initial response data).

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

R: <failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism-too-weak/>
  </failure>
```

7.4.10. not-authorized

[TOC](#)

The authentication failed because the initiating entity did not provide proper credentials or the receiving entity has detected an attack but wishes to disclose as little information as possible to the attacker; sent in reply to a <response/> element or an <auth/> element with initial response data.


```
I: <response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  [ ... ]
</response>

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

Note: This error condition includes but is not limited to the case of incorrect credentials or an unknown username. In order to discourage directory harvest attacks, no differentiation is made between incorrect credentials and an unknown username.

7.4.11. temporary-auth-failure

[TOC](#)

The authentication failed because of a temporary error condition within the receiving entity, and it is advisable for the initiating entity to try again later; sent in reply to an <auth/> element or a <response/> element.

```
I: <response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  [ ... ]
</response>

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

7.4.12. transition-needed

[TOC](#)

The authentication failed because the mechanism cannot be used until the initiating entity provides (for one time only) a plaintext password so that the receiving entity can build a hashed password for use in future authentication attempts; sent in reply to an <auth/> element with or without initial response data.

```
I: <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
    mechanism='CRAM-MD5'>[ ... ]</auth>

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

Note: An XMPP client MUST treat a <transition-needed/> error with extreme caution, SHOULD NOT provide a plaintext password over an XML stream that is not encrypted via Transport Layer Security, and MUST warn a human user before allowing the user to provide a plaintext password over an unencrypted connection.

7.5. SASL Definition

[TOC](#)

The profiling requirements of [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#) require that the following information be supplied by the definition of a using protocol.

service name: "xmpp"

initiation sequence: After the initiating entity provides an opening XML stream header and the receiving entity replies in kind, the receiving entity provides a list of acceptable authentication methods. The initiating entity chooses one method from the list and sends it to the receiving entity as the value of the 'mechanism' attribute possessed by an <auth/> element, optionally including an initial response to avoid a round trip.

exchange sequence: Challenges and responses are carried through the exchange of <challenge/> elements from receiving entity to initiating entity and <response/> elements from initiating entity to receiving entity. The receiving entity reports failure by sending a <failure/> element and success by sending a <success/> element; the initiating entity aborts the exchange by sending an <abort/> element. Upon successful negotiation, both sides consider the original XML stream to be closed and new stream headers are sent by both entities.

security layer negotiation: The security layer takes effect immediately after sending the closing '>' character of the <success/> element for the receiving entity, and immediately after receiving the closing '>' character of the <success/>

element for the initiating entity. The order of layers is first [\[TCP\] \(Postel, J., "Transmission Control Protocol," September 1981.\)](#), then [\[TLS\] \(Dierks, T. and E. Rescorla, "The Transport Layer Security \(TLS\) Protocol Version 1.2," August 2008.\)](#), then [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#), then XMPP.

use of the authorization identity: The authorization identity can be used in XMPP to denote the non-default <node@domain> of a client or the sending <domain> of a server; an empty string is equivalent to an absent authorization identity.

8. Resource Binding

[TOC](#)

8.1. Overview

[TOC](#)

After a client authenticates with a server, it MUST bind a specific resource to the stream so that the server can properly address the client (see [Section 3 \(Addresses\)](#)). That is, there MUST be an XMPP resource identifier associated with the bare JID (<node@domain>) of the client, so that the address for use over that stream is a full JID of the form <node@domain/resource>. This ensures that the server can deliver XML stanzas to and receive XML stanzas from the client (see [Section 11 \(Server Rules for Processing XML Stanzas\)](#)).

After a client has bound a resource to the stream, it is referred to as a CONNECTED RESOURCE. A server SHOULD allow an entity to maintain multiple connected resources simultaneously, where each connected resource is differentiated by a distinct resource identifier; however, a server MUST enable the administrator of an XMPP service to limit the number of connected resources in order to prevent certain denial of service attacks as described under [Section 15.13 \(Denial of Service\)](#). If, before completing the resource binding step, the client attempts to send an outbound XML stanza (i.e., a stanza not directed to the server itself or to the client's own account), the server MUST NOT process the stanza and MUST either ignore the stanza or return a <not-authorized/> stream error to the client.

Support for resource binding is REQUIRED in XMPP client and server implementations.

8.2. Advertising Support

[TOC](#)

Upon sending a new response stream header to the client after successful SASL negotiation, the server MUST include a <bind/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-bind' namespace in the stream features it presents to the client; this <bind/> element MUST include an empty <required/> element to explicitly indicate that it is mandatory for the client to complete resource binding at this stage of the stream negotiation process.

Note: The server MUST NOT include the resource binding stream feature until after successful SASL negotiation.

```
S: <stream:stream
    from='im.example.com'
    id='gPybza0zBmaADgxKXu9UC1bprp0='
    to='juliet@im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

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

Upon being so informed that resource binding is required, the client MUST bind a resource to the stream as described in the following sections.

8.3. Generation of Resource Identifiers

[TOC](#)

A resource identifier MUST at a minimum be unique among the connected resources for that <node@domain>. Enforcement of this policy is the responsibility of the server.

A resource identifier can be security-critical. For example, if a malicious entity can guess a client's resource identifier then it might be able to determine if the client (and therefore the controlling principal) is online or offline, thus resulting in a presence leak as described under [Section 15.14 \(Presence Leaks\)](#). To prevent that possibility, a client can either (1) generate a random resource identifier on its own or (2) ask the server to generate a resource identifier on its behalf, which MUST be random (see [\[RANDOM\]](#) ([Eastlake,](#)

[D., Schiller, J., and S. Crocker, "Randomness Requirements for Security," June 2005.](#)). When generating a random resource identifier, it is RECOMMENDED that the resource identifier be a Universally Unique Identifier (UUID), for which the format specified in [\[UUID\] \(Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier \(UUID\) URN Namespace," July 2005.\)](#) is RECOMMENDED.

8.4. Server-Generated Resource Identifier

[TOC](#)

A server that supports resource binding MUST be able to generate an XMPP resource identifier on behalf of a client.

8.4.1. Success Case

[TOC](#)

A client requests a server-generated resource identifier by sending an IQ stanza of type "set" (see [Section 9.2.3 \(IQ Semantics\)](#)) containing an empty <bind/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-bind' namespace.

```
C: <iq id='bind_1' type='set'>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind' />
</iq>
```

Once the server has generated an XMPP resource identifier for the client, it MUST return an IQ stanza of type "result" to the client, which MUST include a <jid/> child element that specifies the full JID for the connected resource as determined by the server.

```
S: <iq id='bind_1' type='result'>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
        <jid>
            juliet@im.example.com/4db06f06-1ea4-11dc-aca3-000bcd821bfb
        </jid>
    </bind>
</iq>
```

8.4.2. Error Cases

[TOC](#)

When a client asks the server to generate a resource identifier during resource binding, the following stanza error conditions are possible:

*The account has reached a limit on the number of simultaneous connected resources allowed.

*The client is otherwise not allowed to bind a resource to the stream.

8.4.2.1. Resource Constraint

[TOC](#)

If the account has reached a limit on the number of simultaneous connected resources allowed, the server MUST return a <resource-constraint/> error.

```
S: <iq id='bind_2' type='error'>
  <error type='cancel'>
    <resource-constraint
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
  </iq>
```

8.4.2.2. Not Allowed

[TOC](#)

If the client is otherwise not allowed to bind a resource to the stream, the server MUST return a <not-allowed/> error.

```
S: <iq id='bind_2' type='error'>
  <error type='cancel'>
    <not-allowed
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
  </iq>
```

8.5. Client-Submitted Resource Identifier

[TOC](#)

Instead of asking the server to generate a resource identifier on its behalf, a client MAY attempt to submit a resource identifier that it has generated or that the controlling user has provided.

8.5.1. Success Case

[TOC](#)

A client asks its server to accept a client-submitted resource identifier by sending an IQ stanza of type "set" containing a <bind/> element with a child <resource/> element containing non-zero-length XML character data.

```
C: <iq id='bind_2' type='set'>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
        <resource>balcony</resource>
    </bind>
</iq>
```

The server SHOULD accept the client-submitted resource identifier. It does so by returning an IQ stanza of type "result" to the client, including a <jid/> child element that specifies the full JID for the connected resource and contains without modification the client-submitted text.

```
S: <iq id='bind_2' type='result'>
    <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
        <jid>juliet@im.example.com/balcony</jid>
    </bind>
</iq>
```

8.5.2. Error Cases

[TOC](#)

When a client attempts to submit its own XMPP resource identifier during resource binding, the following stanza error conditions are possible in addition to those described under [Section 8.4.2 \(Error Cases\)](#):

- *The provided resource identifier cannot be processed by the server, e.g. because it is not in accordance with the [Resourceprep \(Resourceprep\)](#) profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).
- *The provided resource identifier is already in use.

[TOC](#)

8.5.2.1. Bad Request

If the provided resource identifier cannot be processed by the server, the server MAY return a <bad-request/> error (but SHOULD instead apply the [Resourceprep \(Resourceprep\)](#) profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) or otherwise process the resource identifier so that it is in conformance).

```
S: <iq id='bind_2' type='error'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.5.2.2. Conflict

[TOC](#)

If there is already a connected resource of the same name, the server MUST do one of the following:

1. Not accept the resource identifier provided by the client but instead override it with an XMPP resource identifier that the server generates.
2. Terminate the current resource and allow the newly-requested resource.
3. Disallow the newly-requested resource and maintain the current resource.

Which of these the server does is up to the implementation, although it is RECOMMENDED to implement case #1.

```
S: <iq id='bind_2' type='result'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <jid>
      juliet@im.example.com/balcony 4db06f06-1ea4-11dc-aca3-000bcd821bfb
    </jid>
  </bind>
</iq>
```

In case #2, the server MUST send a <conflict/> stream error to the current resource and return an IQ stanza of type "result" (indicating success) to the newly-requested resource.


```
S: <iq id='bind_2' type='result'/>
```

In case #3, the server MUST send a <conflict/> stanza error to the newly-requested resource but maintain the XML stream for that connection so that the newly-requested resource has an opportunity to negotiate a non-conflicting resource identifier before sending another request for resource binding.

```
S: <iq id='bind_2' type='error'>
  <error type='modify'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.5.3. Retries

[TOC](#)

If an error occurs when a client submits a resource identifier, the server SHOULD allow a configurable but reasonable number of retries (at least 2 and no more than 5); this enables the client to tolerate incorrectly-provided resource identifiers (e.g., bad data formats or duplicate text strings) without being forced to reconnect. After the client has reached the retry limit, the server MUST return a <policy-violation/> stream error to the client.

8.6. Binding Multiple Resources

[TOC](#)

A server MAY support binding of multiple resources to the same stream. This functionality is desirable in certain environments (e.g., for devices that are unable to open more than one TCP connection or when a machine runs a local XMPP client daemon that is used by multiple applications).

8.6.1. Support

[TOC](#)

If a server supports binding of multiple resources to a stream, it MUST enable a client to unbind resources. A server that supports unbinding MUST also support binding of multiple resources. Thus a client can discover whether a server supports binding of multiple resources by determining if the server advertises a stream feature of <unbind/>, as follows.

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

If a server supports binding of multiple resources, it MUST also send the unbind feature advertisement after resource binding has been completed.

8.6.2. Binding an Additional Resource

[TOC](#)

A connected client binds an additional resource by following the protocol for binding of the original resource, i.e., by sending an IQ stanza of type "set" containing a <bind/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-bind' namespace (either empty to request server generation of the resource identifier or containing a <resource/> element with XML character data to request a client-submitted resource identifier).

8.6.3. Unbinding a Resource

[TOC](#)

8.6.3.1. Success Case

[TOC](#)

A client unbinds a resource by sending an IQ stanza of type "set" containing an <unbind/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-bind' namespace, which in turn contains a child element of <resource/> whose XML character data specifies the resource to be unbound:

```
C: <iq id='unbind_1' type='set'>
  <unbind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </unbind>
</iq>
```

If no error occurs, the server MUST unbind the resource and no longer accept stanzas whose 'from' address specifies the full JID associated with that resource.

```
S: <iq id='unbind_1' type='result'/>
```

When a client unbinds the only resource associated with the stream, the server SHOULD close the stream and terminate the TCP connection.

```
S: <iq id='unbind_1' type='result'/>
```

```
S: </stream:stream>
```

8.6.3.2. Error Cases

[TOC](#)

8.6.3.2.1. Unbind Not Supported

[TOC](#)

If the server understands the 'urn:ietf:params:xml:ns:xmpp-bind' namespace but does not understand the <unbind/> element, it MUST return a stanza error, which MUST be <feature-not-implemented/>.

```
S: <iq id='unbind_1' type='error'>
  <error type='modify'>
    <feature-not-implemented
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</iq>
```

8.6.3.2.2. No Such Resource

[TOC](#)

If there is no such resource for that stream, the server MUST return an error of <item-not-found/>.

```
S: <iq id='unbind_1' type='error'>
  <error type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</iq>
```

8.6.4. From Addresses

[TOC](#)

When a client binds multiple resources to the same stream, proper management of 'from' addresses is imperative. In particular, a client MUST specify a 'from' address on every stanza it sends over a stream to which it has bound multiple resources, where the 'from' address is the full JID (<node@domain.tld/resource>) associated with the relevant resource. If a client does not specify a 'from' address on a stanza it sends over a stream to which it has bound multiple resources, the server MUST return the stanza to the client with an <unknown-sender/> stanza error.

```
C: <message to='romeo@example.net'>
    <body>Wherefore art thou?</body>
</message>

S: <message from='romeo@example.net'
    type='error'>
    <body>Wherefore art thou?</body>
    <error type='modify'>
        <unknown-sender xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
</message>
```

Naturally, the rules regarding validation of asserted 'from' addresses still apply (see [Section 11 \(Server Rules for Processing XML Stanzas\)](#)).

9. XML Stanzas

[TOC](#)

After a client has connected to a server or two servers have connected to each other, either party can send XML stanzas over the negotiated stream. Three kinds of XML stanza are defined for the 'jabber:client' and 'jabber:server' namespaces: <message/>, <presence/>, and <iq/>. In addition, there are five common attributes for these stanza types. These common attributes, as well as the basic semantics of the three stanza types, are defined herein; more detailed information regarding the syntax of XML stanzas for instant messaging and presence applications is provided in [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#), and for other applications in the relevant XMPP extension specifications.

A server MUST NOT process a partial stanza and MUST NOT attach meaning to the transmission timing of any part of a stanza (before receipt of the close tag).
Support for the XML stanza syntax and semantics defined herein is REQUIRED in XMPP client and server implementations.

9.1. Common Attributes

[TOC](#)

The following five attributes are common to message, presence, and IQ stanzas.

9.1.1. to

[TOC](#)

The 'to' attribute specifies the JID of the intended recipient for the stanza.

```
<message to='romeo@example.net'>
  <body>Art thou not Romeo, and a Montague?</body>
</message>
```

For information about server processing of inbound and outbound XML stanzas based on the nature of the 'to' address, refer to [Section 11 \(Server Rules for Processing XML Stanzas\)](#).

9.1.1.1. Client-to-Server Streams

[TOC](#)

The following rules apply to inclusion of the 'to' attribute in the context of XML streams qualified by the 'jabber:client' namespace (i.e., client-to-server streams).

1. A stanza with a specific intended recipient MUST possess a 'to' attribute whose value is an XMPP address.
 2. 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) MUST NOT possess a 'to' attribute.
-

[TOC](#)

9.1.1.2. Server-to-Server Streams

The following rules apply to inclusion of the 'to' attribute in the context of XML streams qualified by the 'jabber:server' namespace (i.e., server-to-server streams).

1. A stanza MUST possess a 'to' attribute whose value is an XMPP address; if a server receives a stanza that does not meet this restriction, it MUST generate an <improper-addressing/> stream error.
2. The domain identifier portion of the JID in the 'to' attribute MUST match a hostname serviced by the receiving server; if a server receives a stanza that does not meet this restriction, it MUST generate a <host-unknown/> or <host-gone/> stream error.

9.1.2. from

[TOC](#)

The 'from' attribute specifies the JID of the sender.

```
<message from='juliet@im.example.com/balcony'
        to='romeo@example.net'>
  <body>Art thou not Romeo, and a Montague?</body>
</message>
```

9.1.2.1. Client-to-Server Streams

[TOC](#)

The following rules apply to the 'from' attribute in the context of XML streams qualified by the 'jabber:client' namespace (i.e., client-to-server streams).

1. When the server receives an XML stanza from a client and the stanza does not include a 'from' attribute, the server MUST add a 'from' attribute to the stanza, where the value of the 'from' attribute is the full JID (<node@domain/resource>) determined by the server for the connected resource that generated the stanza (see [Section 3.5 \(Determination of Addresses\)](#)), or the bare JID (<node@domain>) in the case of subscription-related presence stanzas (see [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#)); the only exception to

this rule occurs when multiple resources are bound to the same stream as described under [Section 8.6 \(Binding Multiple Resources\)](#).

2. When the server receives an XML stanza from a client and the stanza includes a 'from' attribute, the server MUST either (a) validate that the value of the 'from' attribute provided by the client is that of a connected resource for the associated entity or (b) override the provided 'from' attribute by adding a 'from' attribute as specified under Rule #1.
3. 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 MUST either (a) not include a 'from' attribute or (b) include a 'from' attribute whose value is the account's bare JID (<node@domain>).
4. When the server generates a stanza from the server itself for delivery to the client, the stanza MUST include a 'from' attribute whose value is the bare JID (i.e., <domain>) of the server.
5. A server MUST 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); therefore, when a client receives a stanza that does not include a 'from' attribute, it MUST assume that the stanza is from the server to which the client is connected.

9.1.2.2. Server-to-Server Streams

[TOC](#)

The following rules apply to the 'from' attribute in the context of XML streams qualified by the 'jabber:server' namespace (i.e., server-to-server streams).

1. A stanza MUST possess a 'from' attribute whose value is an XMPP address; if a server receives a stanza that does not meet this restriction, it MUST generate an <improper-addressing/> stream error.
2. The domain identifier portion of the JID contained in the 'from' attribute MUST match the hostname of the sending server (or any validated domain thereof) as communicated in the SASL negotiation (see [Section 7 \(SASL Negotiation\)](#)), server dialback (see [\[XEP-0220\] \(Saint-Andre, P. and J. Miller, "Server](#)

[Dialback," October 2008.](#)), or similar means; if a server receives a stanza that does not meet this restriction, it MUST generate an <invalid-from/> stream error.

Enforcement of these rules helps to prevent certain denial of service attacks as described under [Section 15.13 \(Denial of Service\)](#).

9.1.3. id

[TOC](#)

The 'id' attribute MAY be used by a sending entity for internal tracking of stanzas that it sends and receives (especially for tracking the request-response interaction inherent in the semantics of IQ stanzas). The value of the 'id' attribute MAY be unique globally, within a domain, or within a stream. The semantics of IQ stanzas impose additional restrictions; see [Section 9.2.3 \(IQ Semantics\)](#).

9.1.4. type

[TOC](#)

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 specified in [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#), 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 9.2.3 \(IQ Semantics\)](#). The only 'type' value common to all three stanzas is "error"; see [Section 9.3 \(Stanza Errors\)](#).

9.1.5. xml:lang

[TOC](#)

A stanza SHOULD possess an 'xml:lang' attribute (as defined in Section 2.12 of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#)) if the stanza contains XML character data that is intended to be presented to a human user (as explained in [\[CHARSET\] \(Alvestrand, H., "IETF Policy on Character Sets and Languages," January 1998.\)](#), "internationalization is for humans"). The value of the

'xml:lang' attribute specifies the default language of any such human-readable XML character data.

```
<presence from='romeo@example.net/orchard' xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
</presence>
```

The value of the 'xml:lang' attribute MAY be overridden by the 'xml:lang' attribute of a specific child element.

```
<presence from='romeo@example.net/orchard' xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
  <status xml:lang='cs'>Dvořák; m se Julii</status>
</presence>
```

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 specified for the stream as defined under [Section 5.3.4 \(xml:lang\)](#).

```
C: <presence from='romeo@example.net/orchard'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
</presence>

S: <presence from='romeo@example.net/orchard'
  to='juliet@im.example.com'
  xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
</presence>
```

If an inbound stanza received by a client or server does not possess an 'xml:lang' attribute, an implementation MUST assume that the default language is that specified for the stream as defined under [Section 5.3.4 \(xml:lang\)](#).

The value of the 'xml:lang' attribute MUST conform to the NMTOKEN datatype (as defined in Section 2.3 of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#)) and MUST conform to the format defined in [\[LANGTAGS\] \(Phillips, A. and M. Davis, "Tags for Identifying Languages," September 2006.\)](#).

A server MUST NOT modify or delete 'xml:lang' attributes on stanzas it receives from other entities.

9.2. Basic Semantics

9.2.1. Message Semantics

[TOC](#)

The <message/> stanza can be seen as a "push" mechanism whereby one entity pushes information to another entity, similar to the communications that occur in a system such as email. All message stanzas SHOULD possess a 'to' attribute that specifies the intended recipient of the message; upon receiving such a stanza, a server SHOULD route or deliver it to the intended recipient (see [Section 11 \(Server Rules for Processing XML Stanzas\)](#) for general routing and delivery rules related to XML stanzas).

9.2.2. Presence Semantics

[TOC](#)

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. In general, a publishing entity (client) SHOULD send a presence stanza with no 'to' attribute, in which case the server to which the entity is connected SHOULD broadcast or multiplex that stanza to all subscribed entities. However, a publishing entity MAY also send a presence stanza with a 'to' attribute, in which case the server SHOULD route or deliver that stanza to the intended recipient. See [Section 11 \(Server Rules for Processing XML Stanzas\)](#) for general routing and delivery rules related to XML stanzas, and [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#) for rules specific to presence applications.

9.2.3. IQ Semantics

[TOC](#)

Info/Query, or IQ, is a request-response mechanism, similar in some ways to the Hypertext Transfer Protocol [\[HTTP\] \(Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999.\)](#). The semantics of IQ enable 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 (see [Section 9.4 \(Extended Content\)](#)), and the interaction is tracked by the requesting entity through use of the 'id' attribute. Thus, IQ interactions follow a common pattern of structured data exchange such as get/result or set/result (although an error can be returned in reply to a request if appropriate):

Requesting Entity	Responding Entity
-----	-----
<iq id='1' type='get'>	
[... payload ...]	
</iq>	
----->	
<iq id='1' type='result'>	
[... payload ...]	
</iq>	
<-----	
<iq id='2' type='set'>	
[... payload ...]	
</iq>	
----->	
<iq id='2' type='error'>	
[... condition ...]	
</iq>	
<-----	

To enforce these semantics, the following rules apply:

1. The 'id' attribute is REQUIRED for IQ stanzas.
2. The 'type' attribute is REQUIRED for IQ stanzas. The value MUST be one of the following (if the value is other than one of the following strings, the recipient or an intermediate router MUST return a stanza error of <bad-request/>):

*get -- The stanza requests information, inquires about what data is needed in order to complete further operations, etc.

*set -- The stanza provides data that is needed for an operation to be completed, sets new values, replaces existing values, etc.

*result -- The stanza is a response to a successful get or set request.

*error -- The stanza reports an error that has occurred regarding processing or delivery of a previously-sent get or set request (see [Section 9.3 \(Stanza Errors\)](#)).

3. An entity that receives an IQ request of type "get" or "set" MUST reply with an IQ response of type "result" or "error". The response MUST preserve the 'id' attribute of the request.
4. An entity that receives a stanza of type "result" or "error" MUST NOT respond to the stanza by sending a further IQ response of type "result" or "error"; however, the requesting entity MAY send another request (e.g., an IQ of type "set" to provide required information discovered through a get/result pair).
5. An IQ stanza of type "get" or "set" MUST contain exactly one child element, which specifies the semantics of the particular request.
6. An IQ stanza of type "result" MUST include zero or one child elements.
7. An IQ stanza of type "error" MAY include the child element contained in the associated "get" or "set" and MUST include an <error/> child; for details, see [Section 9.3 \(Stanza Errors\)](#).

9.3. Stanza Errors

[TOC](#)

Stanza-related errors are handled in a manner similar to [stream errors \(Stream Errors\)](#). Unlike stream errors, stanza errors are recoverable; therefore they do not result in termination of the XML stream and underlying TCP connection. Instead, the entity that discovers the error condition returns an ERROR STANZA to the sender, i.e., a stanza of the same kind (message, presence, or IQ) whose 'type' attribute is set to a value of "error" and which contains an <error/> child element that specifies the error condition. The specified error condition provides a hint regarding actions that the sender can take to remedy the error if possible.

9.3.1. Rules

[TOC](#)

The following rules apply to stanza errors:

1. The receiving or processing entity that detects an error condition in relation to a stanza SHOULD return an error stanza (and MUST do so for IQ stanzas).
2. The entity that generates an error stanza MAY include the original XML sent so that the sender can inspect and, if necessary, correct the XML before attempting to resend.
3. An error stanza MUST contain an <error/> child element.
4. An <error/> child MUST NOT be included if the 'type' attribute has a value other than "error" (or if there is no 'type' attribute).
5. An entity that receives an error stanza MUST NOT respond to the stanza with a further error stanza; this helps to prevent looping.

9.3.2. Syntax

[TOC](#)

The syntax for stanza-related errors is:

```
<stanza-kind from='intended-recipient' to='sender' type='error'>
  [OPTIONAL to include sender XML here]
  <error type='error-type'>
    <defined-condition xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    [<text xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'
      xml:lang='langcode'>
      OPTIONAL descriptive text
    </text>]
    [OPTIONAL application-specific condition element]
  </error>
</stanza-kind>
```

The "stanza-kind" MUST be one of message, presence, or iq.

The "error-type" MUST be one of the following:

- *auth -- retry after providing credentials
- *cancel -- do not retry (the error cannot be remedied)
- *continue -- proceed (the condition was only a warning)
- *modify -- retry after changing the data sent

*wait -- retry after waiting (the error is temporary)

The <error/> element:

*MUST contain a child element corresponding to one of the stanza error conditions defined under [Section 9.3.3 \(Defined Conditions\)](#); this element MUST be qualified by the 'urn:ietf:params:xml:ns:xmpp-stanzas' namespace.

*MAY contain a <text/> child element containing XML character data that describes the error in more detail; this element MUST be qualified by the 'urn:ietf:params:xml:ns:xmpp-stanzas' namespace and SHOULD possess an 'xml:lang' attribute specifying the natural language of the XML character data.

*MAY contain a child element for an application-specific error condition; this element MUST be qualified by an application-specific namespace that defines the syntax and semantics of the element.

The <text/> element is OPTIONAL. If included, it MUST be used only to provide descriptive or diagnostic information that supplements the meaning of a defined condition or application-specific condition. It MUST NOT be interpreted programmatically by an application. It MUST NOT be used as the error message presented to a human user, but MAY be shown in addition to the error message associated with the defined condition element (and, optionally, the application-specific condition element).

9.3.3. Defined Conditions

[TOC](#)

The following conditions are defined for use in stanza errors.

9.3.3.1. bad-request

[TOC](#)

The sender has sent a stanza containing XML that does not conform to the appropriate schema or that cannot be processed (e.g., an IQ stanza that includes an unrecognized value of the 'type' attribute, or an element that is qualified by a recognized namespace but that violates the defined syntax for the element); the associated error type SHOULD be "modify".

```
C: <iq from='juliet@im.example.com/balcony'
    id='some-id'
    to='im.example.com'
    type='subscribe'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>

S: <iq from='im.example.com'
    id='some-id'
    to='juliet@im.example.com/balcony'
    type='error'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</iq>
```

9.3.3.2. conflict

[TOC](#)

Access cannot be granted because an existing resource exists with the same name or address; the associated error type SHOULD be "cancel".

```
C: <iq id='bind_2' type='set'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>balcony</resource>
  </bind>
</iq>

S: <iq id='bind_2' type='error'>
  <error type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</iq>
```

9.3.3.3. feature-not-implemented

[TOC](#)

The feature represented in the XML stanza is not implemented by the intended recipient or an intermediate server and therefore the stanza cannot be processed (e.g., the entity understands the namespace but does not recognize the element name); the associated error type SHOULD be "cancel" or "modify".

```

C: <iq from='juliet@im.example.com/balcony'
    id='subscriptions1'
    to='pubsub.example.com'
    type='get'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <subscriptions/>
    </pubsub>
</iq>

E: <iq from='pubsub.example.com'
    id='subscriptions1'
    to='juliet@im.example.com/balcony'
    type='error'>
  <error type='cancel'>
    <feature-not-implemented
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    <unsupported
      xmlns='http://jabber.org/protocol/pubsub#errors'
      feature='retrieve-subscriptions' />
    </error>
  </iq>

```

9.3.3.4. forbidden

[TOC](#)

The requesting entity does not possess the required permissions to perform the action; the associated error type SHOULD be "auth".

```

C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
  <x xmlns='http://jabber.org/protocol/muc' />
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'
    type='error'>
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</presence>

```

9.3.3.5. gone

[TOC](#)

The recipient or server can no longer be contacted at this address, typically on a permanent basis; the associated error type SHOULD be "cancel" or "modify" and the error stanza SHOULD include a new address as the XML character data of the <gone/> element (which MUST be a URI or IRI at which the entity can be contacted, typically an XMPP IRI as specified in [\[XMPP-URI\] \(Saint-Andre, P., "Internationalized Resource Identifiers \(IRIs\) and Uniform Resource Identifiers \(URIs\) for the Extensible Messaging and Presence Protocol \(XMPP\)," February 2008.\)](#)).

```
C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'
    type='error'>
  <error type='modify'>
    <gone xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
      xmpp:conference.example.com
    </gone>
  </error>
</presence>
```

9.3.3.6. internal-server-error

[TOC](#)

The server could not process the stanza because of a misconfiguration or an otherwise-undefined internal server error; the associated error type SHOULD be "wait" or "cancel".

```

C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
    <x xmlns='http://jabber.org/protocol/muc' />
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'
    type='error'>
    <error type='wait'>
        <internal-server-error
            xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
        </error>
    </presence>

```

9.3.3.7. item-not-found

[TOC](#)

The addressed JID or item requested cannot be found; the associated error type SHOULD be "cancel" or "modify".

```

C: <iq id='unbind_1' type='set'>
    <unbind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
        <resource>someresource</resource>
    </unbind>
</iq>

S: <iq id='unbind_1' type='error'>
    <error type='cancel'>
        <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
</iq>

```

Note: An application MUST NOT return this error if doing so would provide information about the intended recipient's network availability to an entity that is not authorized to know such information; instead it MUST return a <service-unavailable/> error.

[TOC](#)

9.3.3.8. jid-malformed

The sending entity has provided or communicated an XMPP address (e.g., a value of the 'to' attribute) or aspect thereof (e.g., an XMPP resource identifier) that does not adhere to the syntax defined under [Section 3 \(Addresses\)](#); the associated error type SHOULD be "modify".

```
C: <presence
    from='juliet@im.example.com/balcony'
    to='ch@r@cters@muc.example.com/JulieC'>
    <x xmlns='http://jabber.org/protocol/muc' />
</presence>

E: <presence
    from='ch@r@cters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'
    type='error'>
    <error type='modify'>
        <jid-malformed
            xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
</presence>
```

9.3.3.9. not-acceptable

[TOC](#)

The recipient or server understands the request but is refusing to process it because it does not meet criteria defined by the recipient or server (e.g., a local policy regarding stanza size limits or acceptable words in messages); the associated error type SHOULD be "modify".

```
C: <message to='juliet@im.example.com' id='foo'>
    <body>[ ... the-emacs-manual ... ]</body>
</message>

S: <message from='juliet@im.example.com' id='foo'>
    <error type='modify'>
        <not-acceptable
            xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
</message>
```

[TOC](#)

9.3.3.10. not-allowed

The recipient or server does not allow any entity to perform the action (e.g., sending to entities at a blacklisted domain); the associated error type SHOULD be "cancel".

```
C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
  <x xmlns='http://jabber.org/protocol/muc' />
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'
    type='error'>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</presence>
```

9.3.3.11. not-authorized

[TOC](#)

The sender needs to provide proper credentials before being allowed to perform the action, or has provided improper credentials; the associated error type SHOULD be "auth".

```
C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
  <x xmlns='http://jabber.org/protocol/muc' />
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'>
  <error type='auth'>
    <not-authorized xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</presence>
```

[TOC](#)

9.3.3.12. not-modified

The item requested has not changed since it was last requested; the associated error type SHOULD be "continue".

```
C: <iq from='juliet@capulet.com/balcony'
    id='roster2'
    type='get'>
  <query xmlns='jabber:iq:roster'>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='If-None-Match'>
        some-long-opaque-string
      </header>
    </headers>
  </query>
</iq>

S: <iq type='error'
    to='juliet@capulet.com/balcony'
    id='roster2'>
  <query xmlns='jabber:iq:roster'>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='If-None-Match'>
        some-long-opaque-string
      </header>
    </headers>
  </query>
  <error type='modify'>
    <not-modified xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</iq>
```

9.3.3.13. payment-required

[TOC](#)

The requesting entity is not authorized to access the requested service because payment is required; the associated error type SHOULD be "auth".

```

C: <iq from='romeo@example.net/foo'
    id='items1'
    to='pubsub.example.com'
    type='get'>
    <pubsub xmlns='http://jabber.org/protocol/pubsub'>
      <items node='my_musings' />
    </pubsub>
</iq>

E: <iq from='pubsub.example.com'
    id='items1'
    to='romeo@example.net/foo'
    type='error'>
    <error type='auth'>
      <payment-required
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
</iq>

```

9.3.3.14. recipient-unavailable

[TOC](#)

The intended recipient is temporarily unavailable; the associated error type SHOULD be "wait".

```

C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
    <x xmlns='http://jabber.org/protocol/muc' />
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'>
    <error type='wait'>
      <recipient-unavailable
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
</presence>

```

Note: An application MUST NOT return this error if doing so would provide information about the intended recipient's network availability to an entity that is not authorized to know such information; instead it MUST return a <service-unavailable/> error.

9.3.3.15. **redirect**

[TOC](#)

The recipient or server is redirecting requests for this information to another entity, typically in a temporary fashion (the <gone/> condition is used for permanent addressing failures); the associated error type SHOULD be "modify" and the error stanza SHOULD contain the alternate address in the XML character data of the <redirect/> element (which MUST be a URI or IRI at which the entity can be contacted, typically an XMPP IRI as specified in [\[XMPP-URI\] \(Saint-Andre, P., "Internationalized Resource Identifiers \(IRIs\) and Uniform Resource Identifiers \(URIs\) for the Extensible Messaging and Presence Protocol \(XMPP\)," February 2008.\)](#)).

```
C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
  <x xmlns='http://jabber.org/protocol/muc'/>
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'
    type='error'>
  <error type='modify'>
    <redirect xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
      xmpp:characters@conference.example.org
    </redirect>
  </error>
</presence>
```

9.3.3.16. **registration-required**

[TOC](#)

The requesting entity is not authorized to access the requested service because prior registration is required; the associated error type SHOULD be "auth".

```
C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
    <x xmlns='http://jabber.org/protocol/muc' />
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'>
    <error type='auth'>
        <registration-required
            xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
</presence>
```

9.3.3.17. remote-server-not-found

[TOC](#)

A remote server or service specified as part or all of the JID of the intended recipient does not exist; the associated error type SHOULD be "cancel".

```
C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
    <x xmlns='http://jabber.org/protocol/muc' />
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'>
    <error type='wait'>
        <remote-server-not-found
            xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
</presence>
```

9.3.3.18. remote-server-timeout

[TOC](#)

A remote server or service specified as part or all of the JID of the intended recipient (or required to fulfill a request) could not be

contacted within a reasonable amount of time; the associated error type SHOULD be "wait".

```
C: <presence
    from='juliet@im.example.com/balcony'
    to='characters@muc.example.com/JulieC'>
  <x xmlns='http://jabber.org/protocol/muc' />
</presence>

E: <presence
    from='characters@muc.example.com/JulieC'
    to='juliet@im.example.com/balcony'>
  <error type='wait'>
    <remote-server-timeout
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
  </presence>
```

9.3.3.19. resource-constraint

[TOC](#)

The server or recipient lacks the system resources necessary to service the request; the associated error type SHOULD be "wait" or "modify".

```
C: <iq from='romeo@example.net/foo'
    id='items1'
    to='pubsub.example.com'
    type='get'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='my_musings' />
  </pubsub>
</iq>

E: <iq from='pubsub.example.com'
    id='items1'
    to='romeo@example.net/foo'
    type='error'>
  <error type='wait'>
    <resource-constraint
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
  </iq>
```

[TOC](#)

9.3.3.20. service-unavailable

The server or recipient does not currently provide the requested service; the associated error type SHOULD be "cancel".

```
C: <message from='romeo@example.net/foo'
      to='juliet@im.example.com'>
  <body>Hello?</body>
</message>

S: <message from='juliet@im.example.com/foo'
      to='romeo@example.net'>
  <error type='cancel'>
    <service-unavailable
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</message>
```

An application MUST return a <service-unavailable/> error instead of <item-not-found/> or <recipient-unavailable/> if sending one of the latter errors would provide information about the intended recipient's network availability to an entity that is not authorized to know such information.

9.3.3.21. subscription-required

[TOC](#)

The requesting entity is not authorized to access the requested service because a prior subscription is required; the associated error type SHOULD be "auth".

```
C: <message
  from='romeo@example.net/orchard'
  to='playbot@shakespeare.example.com'
  <body>help</body>
</message>

E: <message
  from='playbot@shakespeare.example.com'
  to='romeo@example.net/orchard'
  type='error'>
  <error type='auth'>
    <subscription-required
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  </error>
</message>
```

9.3.3.22. undefined-condition

[TOC](#)

The error condition is not one of those defined by the other conditions in this list; any error type can be associated with this condition, and it SHOULD be used only in conjunction with an application-specific condition.

```
C: <message
    from='northumberland@shakespeare.example'
    id='richard2-4.1.247'
    to='kingrichard@royalty.england.example'>
  <body>My lord, dispatch; read o'er these articles.</body>
  <amp xmlns='http://jabber.org/protocol/amp'>
    <rule action='notify'
      condition='deliver'
      value='stored' />
  </amp>

S: <message from='example.org'
    id='amp1'
    to='northumberland@example.net/field'
    type='error'>
  <amp xmlns='http://jabber.org/protocol/amp'
    from='kingrichard@example.org'
    status='error'
    to='northumberland@example.net/field'>
    <rule action='error'
      condition='deliver'
      value='stored' />
  </amp>
  <error type='modify'>
    <undefined-condition
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    <failed-rules xmlns='http://jabber.org/protocol/amp#errors'>
      <rule action='error'
        condition='deliver'
        value='stored' />
    </failed-rules>
  </error>
</message>
```

[TOC](#)

9.3.3.23. unexpected-request

The recipient or server understood the request but was not expecting it at this time (e.g., the request was out of order); the associated error type SHOULD be "wait" or "modify".

```
C: <iq from='romeo@example.net/foo'
    id='unsub1'
    to='pubsub.example.com'
    type='set'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <unsubscribe
      node='my_musings'
      jid='romeo@example.net' />
  </pubsub>
</iq>

E: <iq from='pubsub.example.com'
    id='unsub1'
    to='romeo@example.net/foo'
    type='error'>
  <error type='cancel'>
    <unexpected-request
      xmlns='urn:iETF:params:xml:ns:xmpp-stanzas' />
    <not-subscribed
      xmlns='http://jabber.org/protocol/pubsub#errors' />
  </error>
</iq>
```

9.3.3.24. unknown-sender

[TOC](#)

The stanza 'from' address specified by a connected client is not valid for the stream (e.g., the stanza does not include a 'from' address when multiple resources are bound to the stream as described under [Section 8.6.4 \(From Addresses\)](#)); the associated error type SHOULD be "modify".

```

C: <message to='romeo@example.net'>
    <body>Wherefore art thou?</body>
</message>

S: <message from='romeo@example.net'
    type='error'>
    <body>Wherefore art thou?</body>
    <error type='modify'>
        <unknown-sender xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    </error>
</message>

```

9.3.4. Application-Specific Conditions

[TOC](#)

As noted, an application MAY provide application-specific stanza error information by including a properly-namespaced child in the error element. The application-specific element SHOULD supplement or further qualify a defined element. Thus, the <error/> element will contain two or three child elements.

```

<iq id='some-id' type='error'>
    <error type='modify'>
        <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
        <too-many-parameters xmlns='http://example.com/ns' />
    </error>
</iq>

```

```

<message type='error' id='another-id'>
    <error type='modify'>
        <undefined-condition
            xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
        <text xml:lang='en'
            xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
            [ ... application-specific information ... ]
        </text>
        <too-many-parameters xmlns='http://example.com/ns' />
    </error>
</message>

```

An entity that receives an application-specific error condition it does not understand MUST ignore the condition.

While the message, presence, and IQ stanzas provide basic semantics for messaging, availability, and request-response interactions, XMPP uses XML namespaces (see [\[XML-NAMES\] \(Layman, A., Hollander, D., Tobin, R., and T. Bray, "Namespaces in XML 1.1 \(Second Edition\)," August 2006.\)](#) to extend the basic stanza syntax for the purpose of providing additional functionality. Thus a message or presence stanza MAY contain one or more optional child elements specifying content that extends the meaning of the message (e.g., an XHTML-formatted version of the message body as described in [\[XEP-0071\] \(Saint-Andre, P., "XHTML-IM," September 2007.\)](#)), and an IQ stanza of type "get" or "set" MUST contain one such child element. This child element MAY have any name and MUST possess a namespace declaration (other than "jabber:client", "jabber:server", or "http://etherx.jabber.org/streams") that defines all data contained within the child element. Such a child element is said to be EXTENDED CONTENT and its namespace name is said to be an EXTENDED NAMESPACE.

Support for any given extended namespace is OPTIONAL on the part of any implementation. If an entity does not understand such a namespace, the entity's expected behavior depends on whether the entity is (1) the recipient or (2) an entity that is routing the stanza to the recipient.

Recipient: If a recipient receives a stanza that contains a child element it does not understand, it MUST silently ignore that particular XML data, i.e., it MUST NOT process it or present it to a user or associated application (if any). In particular:

- *If an entity receives a message or presence stanza that contains XML data qualified by a namespace it does not understand, the portion of the stanza that qualified by the unknown namespace MUST be ignored.
- *If an entity receives a message stanza whose only child element is qualified by a namespace it does not understand, it MUST ignore the entire stanza.
- *If an entity receives an IQ stanza of type "get" or "set" containing a child element qualified by a namespace it does not understand, the entity MUST return an IQ stanza of type "error" with an error condition of <service-unavailable/>.

Router: If a routing entity (typically a server) handles a stanza that contains a child element it does not understand, it MUST ignore the associated XML data by routing or delivering it untouched to the recipient.

9.5. Stanza Size

[TOC](#)

XMPP is optimized for the exchange of relatively large numbers of relatively small stanzas. A client or server MAY enforce a maximum stanza size. The maximum stanza size MUST NOT be smaller than 10000 bytes, from the opening "<" character to the closing ">" character. If an entity receives a stanza that exceeds its maximum stanza size, it MUST return a <not-acceptable/> stanza error or a <policy-violation/> stream error.

10. Examples

[TOC](#)

10.1. Client-to-Server

[TOC](#)

The following examples show the XMPP data flow for a client negotiating an XML stream with a server, exchanging XML stanzas, and closing the negotiated stream. The server is "im.example.com", the server requires use of TLS, the client authenticates via the SASL PLAIN mechanism as "juliet@im.example.com", and the client binds a client-submitted resource to the stream. It is assumed that before sending the initial stream header, the client has already resolved an SRV record of _xmpp-client._tcp.im.example.com and has opened a TCP connection to the advertised port at the resolved IP address.

Note: The alternate steps shown are provided only to illustrate the protocol for failure cases; they are not exhaustive and would not necessarily be triggered by the data sent in the examples.

10.1.1. TLS

[TOC](#)

Step 1: Client initiates stream to server:

```
C: <stream:stream
  from='juliet@im.example.com'
  to='im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
```

Step 2: Server responds by sending a response stream header to client:

```
S: <stream:stream
  from='im.example.com'
  id='t7AMCin9zjMNwQKDnplntZPIDEI='
  to='juliet@im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
```

Step 3: Server sends stream features to client (STARTTLS extension only at this point):

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

Step 4: Client sends STARTTLS command to server:

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

Step 5: Server informs client that it is allowed to proceed:

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

Step 5 (alt): Server informs client that STARTTLS negotiation has failed and closes both XML stream and TCP connection:

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

```
S: </stream:stream>
```

Step 6: Client and server attempt to complete TLS negotiation over the existing TCP connection (see [\[TLS\] \(Dierks, T. and E. Rescorla, "The Transport Layer Security \(TLS\) Protocol Version 1.2," August 2008.\)](#) for details).

Step 7: If TLS negotiation is successful, client initiates a new stream to server:


```
C: <stream:stream
  from='juliet@im.example.com'
  to='im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
```

Step 7 (alt): If TLS negotiation is unsuccessful, server closes TCP connection.

10.1.2. SASL

[TOC](#)

Step 8: Server responds by sending a stream header to client along with any available stream features:

```
S: <stream:stream
  from='im.example.com'
  id='vgKi/bkYME80Aj4r1XMkpucAqe4='
  to='juliet@im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'

S: <stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
    <required/>
  </mechanisms>
</stream:features>
```

Step 9: Client selects an authentication mechanism, in this case [\[PLAIN\] \(Zeilenga, K., "The PLAIN Simple Authentication and Security Layer \(SASL\) Mechanism," August 2006.\)](#):

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

Step 10: Server informs client of success:

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

Step 10 (alt): Server returns error to client:

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

Step 11: Client initiates a new stream to server:

```
C: <stream:stream
  from='juliet@im.example.com'
  to='im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'
```

10.1.3. Resource Binding

[TOC](#)

Step 12: Server responds by sending a stream header to client along with supported features (in this case resource binding):

```
S: <stream:stream
  from='im.example.com'
  id='gPybza0zBmaADgxKXu9UC1bprp0='
  to='juliet@im.example.com'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>

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

Upon being so informed that resource binding is mandatory, the client needs to bind a resource to the stream; here we assume that the client submits a human-readable text string.

Step 13: Client binds a resource:

```
C: <iq id='bind_1' type='set'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    balcony
  </bind>
</iq>
```

Step 14: Server accepts submitted resource identifier and informs client of successful resource binding:

```
S: <iq id='bind_1' type='result'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <jid>
      juliet@im.example.com/balcony
    </jid>
  </bind>
</iq>
```

10.1.4. Stanza Exchange

[TOC](#)

Now the client is allowed to send XML stanzas over the negotiated stream.

```
C: <message from='juliet@im.example.com/balcony'
  to='romeo@example.net'
  xml:lang='en'>
  <body>Art thou not Romeo, and a Montague?</body>
</message>
```

If necessary, sender's server negotiates XML streams with intended recipient's server (see [Section 10.2 \(Server-to-Server Examples\)](#)). The intended recipient replies and the message is delivered to the client.

```
E: <message from='romeo@example.net/orchard'
  to='juliet@im.example.com/balcony'
  xml:lang='en'>
  <body>Neither, fair saint, if either thee dislike.</body>
</message>
```

The client can subsequently send and receive an unbounded number of subsequent XML stanzas over the stream.

10.1.5. Close

[TOC](#)

Desiring to send no further messages, the client closes the stream.

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

Consistent with the recommended stream closing handshake, the server closes the stream as well:

```
S: </stream:stream>
```

Client now terminates the underlying TCP connection.

10.2. Server-to-Server Examples

[TOC](#)

The following examples show the data flow for a server negotiating an XML stream with another server, exchanging XML stanzas, and closing the negotiated stream. The initiating server ("Server1") is im.example.com; the receiving server ("Server2") is example.net and it requires use of TLS; im.example.com presents a certificate and authenticates via the SASL EXTERNAL mechanism. It is assumed that before sending the initial stream header, Server1 has already resolved an SRV record of _xmpp-server._tcp.example.net and has opened a TCP connection to the advertised port at the resolved IP address.

Note: The alternate steps shown are provided only to illustrate the protocol for failure cases; they are not exhaustive and would not necessarily be triggered by the data sent in the examples.

10.2.1. TLS

[TOC](#)

Step 1: Server1 initiates stream to Server2:

```
S1: <stream:stream
    from='im.example.com'
    to='example.net'
    version='1.0'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

Step 2: Server2 responds by sending a response stream header to Server1:

```
S2: <stream:stream
    from='example.net'
    id='hTiXkW+ih9k2SqdgGkk/AZi00J/Q='
    to='im.example.com'
    version='1.0'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

Step 3: Server2 sends stream features to Server1:

```
S2: <stream:features>
    <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
        <required/>
    </starttls>
</stream:features>
```

Step 4: Server1 sends the STARTTLS command to Server2:

```
S1: <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls' />
```

Step 5: Server2 informs Server1 that it is allowed to proceed:

```
S2: <proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls' />
```

Step 5 (alt): Server2 informs Server1 that STARTTLS negotiation has failed and closes stream:

```
S2: <failure xmlns='urn:ietf:params:xml:ns:xmpp-tls' />
```

```
S2: </stream:stream>
```

Step 6: Server1 and Server2 attempt to complete TLS negotiation via TCP (see [\[TLS\] \(Dierks, T. and E. Rescorla, "The Transport Layer Security \(TLS\) Protocol Version 1.2," August 2008.\)](#) for details).

Step 7: If TLS negotiation is successful, Server1 initiates a new stream to Server2:

```
S1: <stream:stream
    from='im.example.com'
    to='example.net'
    version='1.0'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

Step 7 (alt): If TLS negotiation is unsuccessful, Server2 closes TCP connection.

10.2.2. SASL

[TOC](#)

Step 8: Server2 sends a response stream header to Server1 along with available stream features (including a preference for the SASL EXTERNAL mechanism):

```
S2: <stream:stream
    from='example.net'
    id='RChdjlgj/TIBcbT9Kcu31zDihH4='
    to='im.example.com'
    version='1.0'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>

S2: <stream:features>
    <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
        <mechanism>EXTERNAL</mechanism>
        <required/>
    </mechanisms>
</stream:features>
```

Step 9: Server1 selects the EXTERNAL mechanism, in this case with an authorization identity encoded according to [\[BASE64\] \(Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," October 2006.\)](#):

```
S1: <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
    mechanism='EXTERNAL' />eG1wcC5leGFtcGx1LmNvbQ</auth>
```

The decoded authorization identity is "im.example.com".

Step 10: Server2 determines that the authorization identity provided by Server1 matches the information in the presented certificate and therefore returns success:

```
S2: <success xmlns='urn:ietf:params:xml:ns:xmpp-sasl' />
```

Step 10 (alt): Server2 informs Server1 of failed authentication:

```
S2: <failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <not-authorized/>
</failure>
```

```
S2: </stream:stream>
```

Step 11: Server1 initiates a new stream to Server2:

```
S1: <stream:stream
    from='im.example.com'
    to='example.net'
    version='1.0'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

Step 12: Server2 responds by sending a stream header to Server1 along with any additional features (or, in this case, an empty features element):

```
S2: <stream:stream
    from='example.net'
    id='MbbV2FeojySpUIP6J91qaa+TWHM='
    to='im.example.com'
    version='1.0'
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'>
```

```
S2: <stream:features/>
```

10.2.3. Stanza Exchange

[TOC](#)

Now Server1 is allowed to send XML stanzas to Server2 over the negotiated stream; here we assume that the transferred stanzas are those shown earlier for client-to-server communication, albeit over a server-to-server stream qualified by the 'jabber:server' namespace. Server1 sends XML stanza to Server2:

```
S1: <message from='juliet@im.example.com/balcony'
    to='romeo@example.net'
    xml:lang='en'>
    <body>Art thou not Romeo, and a Montague?</body>
</message>
```

The intended recipient replies and the message is delivered from Server2 to Server1.

Server2 sends XML stanza to Server1:

```
S2: <message from='romeo@example.net/orchard'
    to='juliet@im.example.com/balcony'
    xml:lang='en'>
    <body>Neither, fair saint, if either thee dislike.</body>
</message>
```

10.2.4. Close

[TOC](#)

Desiring to send no further messages, Server1 closes the stream. (In practice, the stream would most likely remain open for some time, since Server1 and Server2 do not immediately know if the stream will be needed for further communication.)

S1: </stream:stream>

Consistent with the recommended stream closing handshake, Server2 closes the stream as well:

S2: </stream:stream>

Server1 now terminates the underlying TCP connection.

11. Server Rules for Processing XML Stanzas

[TOC](#)

An XMPP server MUST ensure in-order processing of XML stanzas between any two entities. This includes stanzas sent by a client to its server for direct processing by the server (e.g., in-order processing of a roster get and initial presence as described in [\[rfc3921bis\]](#) (Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," March 2009.)).

Beyond the requirement for in-order processing, each server implementation will contain its own logic for processing stanzas it receives. Such logic determines whether the server needs to ROUTE a given stanza to another domain, DELIVER it to a local entity (typically a connected client associated with a local account), or HANDLE it directly within the server itself. The following rules apply.

Note: Particular XMPP applications MAY specify delivery rules that modify or supplement the following rules; for example, a set of delivery rules for instant messaging and presence applications is defined in [\[rfc3921bis\]](#) (Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," March 2009.).

[TOC](#)

11.1. No 'to' Address

11.1.1. Overview

[TOC](#)

If the stanza possesses no 'to' attribute, the server MUST handle it directly on behalf of the entity that sent it, where the meaning of "handle it directly" depends on whether the stanza is message, presence, or IQ. Because all stanzas received from other servers MUST possess a 'to' attribute, this rule applies only to stanzas received from a local entity (such as a client) that is connected to the server.

11.1.2. Message

[TOC](#)

If the server receives a message stanza with no 'to' attribute, it MUST treat the message as if the 'to' address were the bare JID `<node@domain>` of the sending entity.

11.1.3. Presence

[TOC](#)

If the server receives a presence stanza with no 'to' attribute, it MUST broadcast it to the entities that are subscribed to the sending entity's presence, if applicable ([\[rfc3921bis\]](#) ([Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.](#)) defines the semantics of such broadcasting for presence applications).

11.1.4. IQ

[TOC](#)

If the server receives an IQ stanza with no 'to' attribute, it MUST process the stanza on behalf of the account from which received the stanza, as follows:

1. If the IQ stanza is of type "get" or "set" and the server understands the namespace that qualifies the payload, the server MUST handle the stanza on behalf of the sending entity or return an appropriate error to the sending entity. While the meaning of "handle" is determined by the semantics of the qualifying namespace, in general the server shall respond to

the IQ stanza of type "get" or "set" by returning an appropriate IQ stanza of type "result" or "error", responding as if the server were the bare JID of the sending entity. As an example, if the sending entity sends an IQ stanza of type "get" where the payload is qualified by the 'jabber:iq:roster' namespace (as described in [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#)), then the server shall return the roster associated with the sending entity's bare JID to the particular resource of the sending entity that requested the roster.

2. If the IQ stanza is of type "get" or "set" and the server does not understand the namespace that qualifies the payload, the server MUST return an error to the sending entity, which MUST be <service-unavailable/>.
3. If the IQ stanza is of type "error" or "result", the server MUST handle the error or result as appropriate for the request-response interaction, responding as if the server were the bare JID of the sending entity.

11.2. Local Domain

[TOC](#)

If the hostname of the domain identifier portion of the JID contained in the 'to' attribute matches one of the configured hostnames of the server itself, the server MUST first determine if the hostname is serviced by the server or by a specialized local service. If the latter, the server MUST route the stanza to that service. If the former, the server MUST proceed as follows.

11.2.1. Mere Domain

[TOC](#)

If the JID contained in the 'to' attribute is of the form <domain>, then the server MUST either handle the stanza as appropriate for the stanza kind or return an error stanza to the sender.

[TOC](#)

11.2.2. Domain with Resource

If the JID contained in the 'to' attribute is of the form <domain/resource>, then the server MUST either handle the stanza as appropriate for the stanza kind or return an error stanza to the sender.

11.2.3. Node at Domain

[TOC](#)

Note: For addresses of this type, more detailed rules in the context of instant messaging and presence applications are provided in [\[rfc3921bis\]](#) (Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," March 2009.).

11.2.3.1. No Such User

[TOC](#)

If there is no local account associated with the <node@domain>, how the stanza shall be processed depends on the stanza type.

- *For a message stanza, the server MUST return a <service-unavailable/> stanza error to the sender.

- *For a presence stanza, the server SHOULD silently discard the stanza.

- *For an IQ stanza, the server MUST return a <service-unavailable/> stanza error to the sender.

11.2.3.2. Bare JID

[TOC](#)

If the JID contained in the 'to' attribute is of the form <node@domain>, how the stanza shall be processed depends on the stanza type.

- *For a message stanza, if there exists at least one connected resource for the node the server SHOULD deliver it to at least one of the connected resources. If there exists no connected resource, the server MUST either return an error or store the message offline for delivery when the account next has a connected resource.

*For a presence stanza, if there exists at least one connected resource for the node the server SHOULD deliver it to at least one of the connected resources. If there exists no connected resource, the server MUST silently discard the stanza.

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

11.2.3.3. Full JID

[TOC](#)

If the JID contained in the 'to' attribute is of the form <node@domain/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 <node@domain>.

If the JID contained in the 'to' attribute is of the form <node@domain/resource> and there is a connected resource that exactly matches the full JID, the server SHOULD deliver the stanza to that connected resource.

11.3. Foreign Domain

[TOC](#)

If the hostname of the domain identifier portion of the JID contained in the 'to' attribute does not match one of the configured hostnames of the server itself, the server SHOULD attempt to route the stanza to the foreign domain (subject to local service provisioning and security policies regarding inter-domain communication, since such communication is optional for any given deployment). There are two possible cases.

11.3.1. Existing Stream

[TOC](#)

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 foreign domain over the existing stream.

11.3.2. No Existing Stream

[TOC](#)

If there exists no server-to-server stream between the two domains, the sender's server shall proceed as follows:

1. Resolve the hostname of the foreign domain (as defined under [Section 15.4 \(Server-to-Server Communication\)](#)).
2. Negotiate a server-to-server stream between the two domains (as defined under [Section 6 \(STARTTLS Negotiation\)](#) and [Section 7 \(SASL Negotiation\)](#)).
3. Route the stanza to the authoritative server for the foreign domain over the newly-established stream.

11.3.3. Error Handling

[TOC](#)

If routing of a stanza to the intended recipient's server is unsuccessful, the sender's server MUST return an error to the sender. If resolution of the foreign domain is unsuccessful, the stanza error MUST be `<remote-server-not-found/>`. If resolution succeeds but streams cannot be negotiated, the stanza error MUST be `<remote-server-timeout/>`.

If stream negotiation with the intended recipient's server is successful but the foreign server cannot deliver the stanza to the recipient, the foreign server shall return an appropriate error to the sender by way of the sender's server.

12. XML Usage

[TOC](#)

12.1. Restrictions

[TOC](#)

The Extensible Messaging and Presence Protocol (XMPP) defines a class of data objects called XML streams as well as the behavior of computer programs that process XML streams. XMPP is an application profile or restricted form of the Extensible Markup Language [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#), and a complete XML stream (including start and end stream tags) is a conforming XML document.

However, XMPP does not deal with XML documents but with XML streams. Because XMPP does not require the parsing of arbitrary and complete XML documents, there is no requirement that XMPP needs to support the full feature set of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C.,](#)

[Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.](#)). In particular, the following features of XML are prohibited in XMPP:

- *comments (as defined in Section 2.5 of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#))
- *processing instructions (Section 2.6 therein)
- *internal or external DTD subsets (Section 2.8 therein)
- *internal or external entity references (Section 4.2 therein) with the exception of predefined entities (Section 4.6 therein)

An XMPP implementation MUST behave as follows with regard to these features:

1. An XMPP implementation MUST NOT inject characters matching such features into an XML stream.
2. If an XMPP implementation receives characters matching such features over an XML stream, it MUST return a stream error, which SHOULD be <restricted-xml/> but MAY be <bad-format/>.

12.2. XML Namespace Names and Prefixes

[TOC](#)

XML namespaces (see [\[XML-NAMES\] \(Layman, A., Hollander, D., Tobin, R., and T. Bray, "Namespaces in XML 1.1 \(Second Edition\)," August 2006.\)](#)) are used within XMPP streams to create strict boundaries of data ownership. The basic function of namespaces is to separate different vocabularies of XML elements that are structurally mixed together. Ensuring that XMPP streams are namespace-aware enables any allowable XML to be structurally mixed with any data element within XMPP. XMPP-specific rules for XML namespace names and prefixes are defined in the following subsections.

12.2.1. Streams Namespace

[TOC](#)

A streams namespace declaration is REQUIRED in all XML stream headers and the name of the streams namespace MUST be 'http://etherx.jabber.org/streams'. If this rule is violated, the entity that receives the offending stream header MUST return a stream error to the

sending entity, which SHOULD be <invalid-namespace/> but MAY be <bad-format/>.

The element names of the <stream/> element and its <features/> and <error/> children MUST be qualified by the streams namespace prefix in all instances. If this rule is violated, the entity that receives the offending element MUST return a stream error to the sending entity, which SHOULD be <bad-format/>.

An implementation SHOULD generate only the 'stream:' prefix for these elements, and for historical reasons MAY accept only the 'stream:' prefix. If an entity receives a stream header with a streams namespace prefix it does not accept, it MUST return a stream error to the sending entity, which SHOULD be <bad-namespace-prefix/> but MAY be <bad-format/>.

12.2.2. Default Namespace

[TOC](#)

A default namespace declaration is REQUIRED and defines the allowable first-level children of the root stream element. This namespace declaration MUST be the same for the initial stream and the response stream so that both streams are qualified consistently. The default namespace declaration applies to the stream and all first-level child element sent within a stream unless explicitly qualified by the streams namespace or another namespace.

A server implementation MUST support the following two default namespaces:

*jabber:client -- this default namespace is declared when the stream is used for communication between a client and a server

*jabber:server -- this default namespace is declared when the stream is used for communication between two servers

A client implementation MUST support the 'jabber:client' default namespace.

If an implementation accepts a stream that is qualified by the 'jabber:client' or 'jabber:server' namespace, it MUST support the [common attributes \(Common Attributes\)](#) and [basic semantics \(Basic Semantics\)](#) of all three core stanza types (message, presence, and IQ). For historical reasons, an implementation MAY refuse to support any other default namespaces. If an entity receives a stream header with a default namespace it does not support, it MUST return an <invalid-namespace/> stream error.

An implementation MUST NOT generate namespace prefixes for elements qualified by the default namespace if the default namespace is 'jabber:client' or 'jabber:server'.

Note: The 'jabber:client' and 'jabber:server' namespaces are nearly identical but are used in different contexts (client-to-server communication for 'jabber:client' and server-to-server communication for 'jabber:server'). The only difference between the two is that the 'to' and 'from' attributes are OPTIONAL on stanzas sent over XML streams qualified by the 'jabber:client' namespace, whereas they are REQUIRED on stanzas sent over XML streams qualified by the 'jabber:server' namespace.

An implementation MAY support a default namespace other than "jabber:client" or "jabber:server". However, because such namespaces would define applications other than XMPP, they are to be defined in separate specifications.

12.2.3. Extended Namespaces

[TOC](#)

An EXTENDED NAMESPACE is an XML namespace that qualifies extended content as defined under [Section 9.4 \(Extended Content\)](#). For example, in the following stanza, the extended namespace is 'jabber:iq:roster':

```
<iq from='juliet@capulet.com/balcony'
  id='roster1'
  type='get'>
  <query xmlns='jabber:iq:roster'/>
</iq>
```

An XML stanza MAY contain XML data qualified by more than one extended namespace, either at the direct child level of the stanza (for presence and message stanzas) or in any mix of levels (for all stanzas).

```
<presence from='juliet@capulet.com/balcony'>
  <c xmlns='http://jabber.org/protocol/caps'
    node='http://exodus.jabberstudio.org/caps'
    ver='0.9'/>
  <x xmlns='vcard-temp:x:update'>
    <photo>sha1-hash-of-image</photo>
  </x>
</presence>
```



```

<message to='juliet@capulet.com'>
  <body>Hello?</body>
  <html xmlns='http://jabber.org/protocol/xhtml-im'>
    <body xmlns='http://www.w3.org/1999/xhtml'>
      <p style='font-weight:bold'>Hello?</t>
    </body>
  </html>
</message>

<iq from='juliet@capulet.com/balcony'
  id='roster2'
  type='get'>
  <query xmlns='jabber:iq:roster'>
    <headers xmlns='http://jabber.org/protocol/shim'>
      <header name='If-None-Match'>some-long-opaque-string</header>
    </headers>
  </query>
</iq>

```

An implementation SHOULD NOT generate namespace prefixes for elements qualified by content (as opposed to stream) namespaces other than the default namespace. However, if included, the namespace declarations for those prefixes MUST be included on the stanza root or a child thereof, not at the level of the stream element (this helps to ensure that any such namespace declaration is routed and delivered with the stanza, instead of assumed from the stream).

12.3. Well-Formedness

[TOC](#)

There are two varieties of well-formedness:

- *"XML-well-formedness" in accordance with the definition of "well-formed" in Section 2.1 of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#).

- *"Namespace-well-formedness" in accordance with the definition of "namespace-well-formed" in Section 7 of [\[XML-NAMES\] \(Layman, A., Hollander, D., Tobin, R., and T. Bray, "Namespaces in XML 1.1 \(Second Edition\)," August 2006.\)](#).

The following rules apply.

An XMPP entity MUST NOT generate data that is not XML-well-formed. An XMPP entity MUST NOT accept data that is not XML-well-formed; instead

it MUST return an <xml-not-well-formed/> stream error and close the stream over which the data was received.

An XMPP entity MUST NOT generate data that is not namespace-well-formed. An XMPP server SHOULD NOT route or deliver data that is not namespace-well-formed, and SHOULD return a stanza error of <not-acceptable/> or a stream error of <xml-not-well-formed/> in response to the receipt of such data.

Note: Because these restrictions were underspecified in an earlier revision of this specification, it is possible that implementations based on that revision will send data that does not comply with the restrictions; an entity SHOULD be liberal in accepting such data.

12.4. Validation

[TOC](#)

A server is not responsible for ensuring that XML data delivered to a client or routed to another server is valid, in accordance with the definition of "valid" provided in Section 2.8 of [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#). An implementation MAY choose to accept or provide only validated data, but such behavior is OPTIONAL. A client SHOULD NOT rely on the ability to send data that does not conform to the schemas, and SHOULD ignore any non-conformant elements or attributes on the incoming XML stream.

Note: The terms "valid" and "well-formed" are distinct in XML.

12.5. Inclusion of Text Declaration

[TOC](#)

Implementations SHOULD send a text declaration before sending a stream header. Applications MUST follow the rules provided in [\[XML\] \(Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, "Extensible Markup Language \(XML\) 1.0 \(Fourth Edition\)," August 2006.\)](#) regarding the circumstances under which a text declaration is included.

12.6. Character Encoding

[TOC](#)

Implementations MUST support the UTF-8 transformation of Universal Character Set [\[UCS2\] \(International Organization for Standardization, "Information Technology - Universal Multiple-octet coded Character Set](#)

[\(UCS\) - Amendment 2: UCS Transformation Format 8 \(UTF-8\)," October 1996.\)](#) characters, as required by [\[CHARSET\] \(Alvestrand, H., "IETF Policy on Character Sets and Languages," January 1998.\)](#) and defined in [\[UTF-8\] \(Yergeau, F., "UTF-8, a transformation format of ISO 10646," November 2003.\)](#). Implementations MUST NOT attempt to use any other encoding. If one party to an XML stream detects that the other party has attempted to send XML data with an encoding other than UTF-8, it MUST return a stream error, which SHOULD be <unsupported-encoding/> but MAY be <bad-format/>.

Note: Because it is mandatory for an XMPP implementation to support all and only the UTF-8 encoding and because UTF-8 always has the same byte order, an implementation MUST NOT send a byte order mark ("BOM") at the beginning of the data stream. If an entity receives the Unicode character U+FEFF anywhere in an XML stream (including as the first character of the stream), it MUST interpret that character as a zero width no-break space, not as a byte order mark.

12.7. Whitespace

[TOC](#)

Except where explicitly disallowed (e.g., during [TLS negotiation \(STARTTLS Negotiation\)](#) and [SASL negotiation \(SASL Negotiation\)](#)), either entity MAY send whitespace within the root stream element as separators between XML stanzas or between any other first-level elements sent over the stream. One common use for sending such whitespace is explained under [Section 5.7.3 \(Handling of Idle Streams\)](#).

12.8. XML Versions

[TOC](#)

XMPP is an application profile of XML 1.0. A future version of XMPP might be defined in terms of higher versions of XML, but this specification addresses XML 1.0 only.

13. Compliance Requirements

[TOC](#)

This section summarizes the specific aspects of the Extensible Messaging and Presence Protocol that MUST be supported by servers and clients in order to be considered compliant implementations, as well as additional protocol aspects that SHOULD be supported. For compliance purposes, we draw a distinction between core protocols (which MUST be supported by any server or client, regardless of the specific application) and instant messaging and presence protocols (which MUST

be supported only by instant messaging and presence applications built on top of the core protocols). Compliance requirements that apply to all servers and clients are specified in this section; compliance requirements for instant messaging and presence applications are specified in the corresponding section of [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#).

13.1. Servers

[TOC](#)

A server MUST support the following core protocols in order to be considered compliant:

- *Conformance with [\[IDNA\] \(Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications \(IDNA\)," March 2003.\)](#) for domain identifiers, the [Nodeprep \(Nodeprep\)](#) profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) for node identifiers, and the [Resourceprep \(Resourceprep\)](#) profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) for resource identifiers, as well as enforcement thereof for clients that authenticate with the server
- *[XML streams \(XML Streams\)](#), including [TLS negotiation \(STARTTLS Negotiation\)](#), [SASL negotiation \(SASL Negotiation\)](#), [stream features \(Stream Features\)](#), and [Resource Binding \(Resource Binding\)](#)
- *The basic semantics of the three defined stanza types (i.e., `<message/>`, `<presence/>`, and `<iq/>`)
- *Generation (and, where appropriate, handling) of error syntax and semantics related to streams, TLS, SASL, and XML stanzas

For backward compatibility with the large deployed base of XMPP servers, server developers are advised to implement the server dialback protocol first specified in [\[RFC3920\] \(Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol \(XMPP\): Core," October 2004.\)](#) and now documented in [\[XEP-0220\] \(Saint-Andre, P. and J. Miller, "Server Dialback," October 2008.\)](#), since that protocol is widely used for weak identity verification of peer servers in the absence of domain certificates.

13.2. Clients

[TOC](#)

A client MUST support the following core protocols in order to be considered compliant:

- *[XML streams \(XML Streams\)](#), including [TLS negotiation \(STARTTLS Negotiation\)](#), [SASL negotiation \(SASL Negotiation\)](#), [stream features \(Stream Features\)](#), and [Resource Binding \(Resource Binding\)](#)

- *The basic semantics of the three defined stanza types (i.e., <message/>, <presence/>, and <iq/>)

- *Handling (and, where appropriate, generation) of error syntax and semantics related to streams, TLS, SASL, and XML stanzas

In addition, a client SHOULD support the following core protocols:

- *Conformance with [\[IDNA\] \(Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications \(IDNA\)," March 2003.\)](#) for domain identifiers, the [Nodeprep \(Nodeprep\)](#) profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) for node identifiers, and the [Resourceprep \(Resourceprep\)](#) profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) for resource identifiers.

14. Internationalization Considerations

[TOC](#)

As specified under [Section 12.6 \(Character Encoding\)](#), XML streams MUST be encoded in UTF-8.

As specified under [Section 5.3 \(Stream Attributes\)](#), an XML stream SHOULD include an 'xml:lang' attribute specifying the default language for any XML character data that is intended to be presented to a human user. As specified under [Section 9.1.5 \(xml:lang\)](#), an XML stanza SHOULD include an 'xml:lang' attribute if the stanza contains XML character data that is intended to be presented to a human user. A server SHOULD apply the default 'xml:lang' attribute to stanzas it routes or delivers on behalf of connected entities, and MUST NOT modify or delete 'xml:lang' attributes on stanzas it receives from other entities.

As specified under [Section 3 \(Addresses\)](#), a server MUST support and enforce [\[IDNA\] \(Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications \(IDNA\)," March 2003.\)](#) for domain identifiers, the [Nodeprep \(Nodeprep\)](#) profile of [\[STRINGPREP\]](#)

([Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.](#)) for node identifiers, and the [Resourceprep \(Resourceprep\) profile of \[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) for resource identifiers; this enables XMPP addresses to include a wide variety of Unicode characters outside the US-ASCII range.

15. Security Considerations

[TOC](#)

15.1. High Security

[TOC](#)

For the purposes of XMPP communication (client-to-server and server-to-server), the term "high security" refers to the use of security technologies that provide both mutual authentication and integrity checking; in particular, when using certificate-based authentication to provide high security, a chain-of-trust SHOULD be established out-of-band, although a shared certification authority signing certificates could allow a previously unknown certificate to establish trust in-band. See [Section 15.2 \(Certificates\)](#) regarding certificate validation procedures.

Implementations MUST support high security. Service provisioning SHOULD use high security, subject to local security policies.

15.2. Certificates

[TOC](#)

Channel encryption of an XML stream using Transport Layer Security as described under [Section 6 \(STARTTLS Negotiation\)](#), and in some cases also authentication as described under [Section 7 \(SASL Negotiation\)](#), is commonly based on a digital certificate presented by the receiving entity (or, in the case of mutual authentication, both the receiving entity and the initiating entity). This section describes best practices regarding the generation of digital certificates to be presented by XMPP entities and the verification of digital certificates presented by XMPP entities.

[TOC](#)

15.2.1. Certificate Generation

15.2.1.1. Server Certificates

[TOC](#)

In a digital certificate to be presented by an XMPP server (i.e., a SERVER CERTIFICATE), it is RECOMMENDED for the certificate to include one or more JIDs (i.e., domain identifiers) associated with domains serviced at the server. The representations described in the following sections are RECOMMENDED. These representations are provided in preference order.

15.2.1.1.1. SRVName

[TOC](#)

A server's domain identifier SHOULD be represented as an SRVName, i.e., as an otherName field of type "id-on-dnsSRV" as specified in [\[X509-SRV\] \(Santesson, S., "Internet X.509 Public Key Infrastructure Subject Alternative Name for Expression of Service Name," August 2007.\)](#).

15.2.1.1.2. dNSName

[TOC](#)

A server's domain identifier SHOULD be represented as a dNSName, i.e., as a subjectAltName extension of type dNSName. The dNSName MAY contain the wildcard character '*'. The wildcard character applies only to the left-most domain name component or component fragment and matches any single component or component fragment. For instance, a dNSName of *.example.com matches foo.example.com but not bar.foo.example.com or example.com itself; similarly, a dNSName of im*.example.net matches im1.example.net and im2.example.net but not chat.example.net or example.net itself.

15.2.1.1.3. XmppAddr

[TOC](#)

A server's domain identifier MAY be represented as an XmppAddr, i.e., as a UTF8String within an otherName entity inside the subjectAltName, using the [\[ASN.1\] \(CCITT, "Recommendation X.208: Specification of Abstract Syntax Notation One \(ASN.1\)," 1988.\)](#) Object Identifier "id-on-xmppAddr" specified under [Section 15.2.1.3 \(ASN.1 Object Identifier\)](#).

In server certificates, this representation is included only for the sake of backward-compatibility.

15.2.1.1.4. Common Name

[TOC](#)

A server's domain identifier SHOULD NOT be represented as a Common Name; instead, the Common Name field SHOULD be reserved for representation of a human-friendly name.

15.2.1.1.5. Examples

[TOC](#)

For our first (relatively simple) example, consider a company called "Example Products, Inc." It hosts an XMPP service at "im.example.com" (i.e., user addresses at the service are of the form "user@im.example.com"), and SRV lookups for the xmpp-client and xmpp-server services at "im.example.com" yield one machine, called "x.example.com", as follows:

```
_xmpp-client._tcp.im.example.com. 400 IN SRV 20 0 5222 x.example.com
_xmpp-server._tcp.im.example.com. 400 IN SRV 20 0 5269 x.example.com
```

The certificate presented by x.example.com contains the following representations:

- *An otherName type of SRVName (id-on-dnsSRV) containing an IA5String (ASCII) string of: "_xmpp-client.im.example.com"
- *An otherName type of SRVName (id-on-dnsSRV) containing an IA5String (ASCII) string of: "_xmpp-server.im.example.com"
- *A dNSName containing an ASCII string of "im.example.com"
- *An otherName type of XmppAddr (id-on-xmppAddr) containing a UTF-8 string of: "im.example.com"
- *A CN containing an ASCII string of "Example Products, Inc."

For our second (more complex) example, consider an ISP called "Example Internet Services". It hosts an XMPP service at "example.net" (i.e., user addresses at the service are of the form "user@example.net"), but SRV lookups for the xmpp-client and xmpp-server services at "example.net" yield two machines ("x1.example.net" and "x2.example.net"), as follows:


```
_xmpp-client._tcp.example.net. 68400 IN SRV 20 0 5222 x1.example.net.  
_xmpp-client._tcp.example.net. 68400 IN SRV 20 0 5222 x2.example.net.  
_xmpp-server._tcp.example.net. 68400 IN SRV 20 0 5269 x1.example.net.  
_xmpp-server._tcp.example.net. 68400 IN SRV 20 0 5269 x2.example.net.
```

Example Internet Services also hosts chatrooms at chat.example.net, and provides an xmpp-server SRV record for that service as well (thus enabling entity from foreign domains to access that service). It also might provide other such services in the future, so it wishes to represent a wildcard in its certificate to handle such growth. The certificate presented by either x1.example.net or x2.example.net contains the following representations:

- *An otherName type of SRVName (id-on-dnsSRV) containing an IA5String (ASCII) string of: "_xmpp-client.example.net"
- *An otherName type of SRVName (id-on-dnsSRV) containing an IA5String (ASCII) string of: "_xmpp-server.example.net"
- *An otherName type of SRVName (id-on-dnsSRV) containing an IA5String (ASCII) string of: "_xmpp-server.chat.example.net"
- *A dNSName containing an ASCII string of "example.net"
- *A dNSName containing an ASCII string of "*.example.net"
- *An otherName type of XmppAddr (id-on-xmppAddr) containing a UTF-8 string of: "example.net"
- *An otherName type of XmppAddr (id-on-xmppAddr) containing a UTF-8 string of: "chat.example.net"
- *A CN containing an ASCII string of "Example Internet Services"

15.2.1.2. Client Certificates

[TOC](#)

In a digital certificate to be presented by an XMPP client controlled by a human user (i.e., a CLIENT CERTIFICATE), it is RECOMMENDED for the certificate to include one or more JIDs associated with an XMPP user. If included, a JID MUST be represented as an XmppAddr, i.e., as a UTF8String within an otherName entity inside the subjectAltName, using the [\[ASN.1\] \(CCITT, "Recommendation X.208: Specification of Abstract Syntax Notation One \(ASN.1\)," 1988.\)](#) Object Identifier "id-on-xmppAddr" specified under [Section 15.2.1.3 \(ASN.1 Object Identifier\)](#).

15.2.1.3. ASN.1 Object Identifier

[TOC](#)

The [\[ASN.1\]](#) (CCITT, "Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1)," 1988.) Object Identifier "id-on-xmppAddr" (also called an XmppAddr) is defined as follows.

```
id-pkix OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
    dod(6) internet(1) security(5) mechanisms(5) pkix(7) }

id-on OBJECT IDENTIFIER ::= { id-pkix 8 } -- other name forms

id-on-xmppAddr OBJECT IDENTIFIER ::= { id-on 5 }

XmppAddr ::= UTF8String
```

As an alternative to the "id-on-xmppAddr" notation, this Object Identifier MAY be represented in dotted display format (i.e., "1.3.6.1.5.5.7.8.5") or in the Uniform Resource Name notation specified in [\[URN-OID\]](#) (Mealling, M., "A URN Namespace of Object Identifiers," February 2001.) (i.e., "urn:oid:1.3.6.1.5.5.7.8.5"). Thus for example the JID "juliet@im.example.com" as included in a certificate could be formatted in any of the following three ways:

```
id-on-xmppAddr:  subjectAltName=otherName:id-on-
    xmppAddr;UTF8:juliet@im.example.com
```

```
dotted display format:  subjectAltName=otherName:
    1.3.6.1.5.5.7.8.5;UTF8:juliet@im.example.com
```

```
URN notation:  subjectAltName=otherName:urn:oid:
    1.3.6.1.5.5.7.8.5;UTF8:juliet@im.example.com
```

Use of the "id-on-xmppAddr" format is RECOMMENDED in the generation of certificates, but all three formats MUST be supported for the purpose of certificate validation.

15.2.2. Certificate Validation

[TOC](#)

When an XMPP entity is presented with a server certificate or client certificate by a peer for the purpose of encryption or authentication of XML streams as described under [Section 6 \(STARTTLS Negotiation\)](#) and [Section 7 \(SASL Negotiation\)](#), the entity MUST validate the certificate to determine if the certificate shall be considered a TRUSTED CERTIFICATE, i.e., a certificate that is acceptable for encryption and/

or authentication in accordance with the XMPP entity's local service policies or configured settings.

For both server certificates and client certificates, the validating entity MUST verify the integrity of the certificate, MUST verify that the certificate has been properly signed by the issuing Certificate Authority, and MUST support certificate revocation messages. An implementation MUST enable a human user to view information about the full chain of certificates.

The following sections describe certificate validation rules for server-to-server and client-to-server streams.

15.2.2.1. Server-to-Server Streams

[TOC](#)

When an XMPP entity (client or server) validates a certificate presented by an XMPP server, there are three possible cases, as discussed in the following sections.

15.2.2.1.1. Case #1

[TOC](#)

If the server certificate appears to be certified by a chain of certificates terminating in a trust anchor (as described in Section 6.1 of [\[X509\] \(Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List \(CRL\) Profile," April 2002.\)](#)), the entity MUST check the certificate for any instances of the SRVName, dNSName, and XmppAddr (in that order of preference) as described under [Section 15.2.1.1.1 \(SRVName\)](#), [Section 15.2.1.1.2 \(dNSName\)](#), and [Section 15.2.1.1.3 \(XmppAddr\)](#). There are three possible sub-cases:

Sub-Case #1: The entity finds at least one SRVName, dNSName, or XmppAddr that matches the hostname to which it attempted to connect; the entity MUST use this represented domain identifier as the validated identity of the XMPP server. The server certificate MUST be checked against the hostname as provided by the entity (client or server), not the hostname as resolved via the Domain Name System; e.g., if a user specifies a hostname of "example.net" but a [\[DNS-SRV\] \(Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services \(DNS SRV\)," February 2000.\)](#) lookup returns "x1.example.net", the certificate MUST be checked as "example.net". A user-oriented client MAY provide a configuration setting that enables a human user to explicitly specify a hostname to be checked for connection purposes.

Sub-Case #2:

The entity finds no SRVName, dNSName, or XmppAddr that matches the hostname to which it attempted to connect and a human user has not permanently accepted the certificate during a previous connection attempt; the entity MUST NOT use the represented domain identifier (if any) as the validated identity of the XMPP server. Instead, if the connecting entity is a user-oriented client then it MUST either (1) automatically terminate the connection with a bad certificate error or (2) show the certificate (including the entire certificate chain) to the user and give the user the choice of terminating the connecting or accepting the certificate temporarily (i.e., for this connection attempt only) or permanently (i.e., for all future connection attempts) and then continuing with the connection; if a user permanently accepts a certificate in this way, the client MUST cache the certificate (or some non-forgable representation such as a hash) and in future connection attempts behave as in Sub-Case #3. (It is the responsibility of the human user to verify the hash or fingerprint of the certificate with the peer over a trusted communication layer.) If the connecting entity is an XMPP server or an automated client, the application SHOULD terminate the connection (with a bad certificate error) and log the error to an appropriate audit log; an XMPP server or automated client MAY provide a configuration setting that disables this check, but MUST provide a setting that enables the check.

Sub-Case #3: The entity finds no SRVName, dNSName, or XmppAddr that matches the hostname to which it attempted to connect but a human user has permanently accepted the certificate during a previous connection attempt; the entity MUST verify that the cached certificate was presented and MUST notify the user if the certificate has changed.

15.2.2.1.2. Case #2[TOC](#)

If the server certificate is certified by a Certificate Authority not known to the entity, the entity MUST proceed as under Case #1, Sub-Case #2 or Case #1, Sub-Case #3 as appropriate.

15.2.2.1.3. Case #3[TOC](#)

If the server certificate is self-signed, the entity MUST proceed as under Case #1, Sub-Case #2 or Case #1, Sub-Case #3 as appropriate.

15.2.2.2. Client-to-Server Streams

[TOC](#)

When an XMPP server validates a certificate presented by a client, there are three possible cases, as discussed in the following sections.

15.2.2.2.1. Case #1

[TOC](#)

If the client certificate appears to be certified by a chain of certificates terminating in a trust anchor (as described in Section 6.1 of [\[X509\] \(Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List \(CRL\) Profile," April 2002.\)](#)), the server MUST check the certificate for any instances of the XmppAddr as described under [Section 15.2.1.3 \(ASN.1 Object Identifier\)](#). There are three possible sub-cases:

Sub-Case #1: The server finds one XmppAddr for which the domain identifier portion of the represented JID matches one of the configured hostnames of the server itself; the server SHOULD use this represented JID as the validated identity of the client.

Sub-Case #2: The server finds more than one XmppAddr for which the domain identifier portion of the represented JID matches one of the configured hostnames of the server itself; the server SHOULD use one of these represented JIDs as the validated identity of the client, choosing among them according to local service policies or based on the 'to' address of the initial stream header.

Sub-Case #3: The server finds no XmppAddrs, or finds at least one XmppAddr but the domain identifier portion of the represented JID does not match one of the configured hostnames of the server itself; the server MUST NOT use the represented JID (if any) as the validated identity of the client but instead MUST either validate the identity of the client using other means.

[TOC](#)

15.2.2.2.2. Case #2

If the client certificate is certified by a Certificate Authority not known to the server, the server MUST proceed as under Case #1, Sub-Case #3.

15.2.2.2.3. Case #3

[TOC](#)

If the client certificate is self-signed, the server MUST proceed as under Case #1, Sub-Case #3.

15.2.2.3. Use of Certificates in XMPP Extensions

[TOC](#)

Certificates MAY be used in extensions to XMPP for the purpose of application-layer encryption or authentication above the level of XML streams (e.g., for end-to-end encryption). Such extensions shall define their own certificate handling rules, which at a minimum SHOULD be consistent with the rules specified herein but MAY specify additional rules.

15.3. Client-to-Server Communication

[TOC](#)

A compliant client implementation MUST support both TLS and SASL for connections to a server.

The TLS protocol for encrypting XML streams (defined under [Section 6 \(STARTTLS Negotiation\)](#)) provides a reliable mechanism for helping to ensure the confidentiality and data integrity of data exchanged between two entities.

The SASL protocol for authenticating XML streams (defined under [Section 7 \(SASL Negotiation\)](#)) provides a reliable mechanism for validating that a client connecting to a server is who it claims to be. Client-to-server communication MUST NOT proceed until the DNS hostname asserted by the server has been resolved as specified under [Section 4 \(TCP Binding\)](#). If there is a mismatch between the hostname to which a client attempted to connect (e.g., "example.net") and the hostname to which the client actually connects (e.g., "x1.example.net"), the client MUST warn a human user about the mismatch and the human user MUST approve the connection before the client proceeds; however, the client MAY also allow the user to add the presented hostname to a configured set of accepted hostnames to expedite future connections.

A client's IP address and method of access MUST NOT be made public by a server, nor are any connections other than the original server connection required. This helps to protect the client's server from direct attack or identification by third parties.

15.4. Server-to-Server Communication

[TOC](#)

A compliant server implementation MUST support both TLS and SASL for inter-domain communication.

Because service provisioning is a matter of policy, it is optional for any given domain to communicate with other domains, and server-to-server communication can be disabled by the administrator of any given deployment. If a particular domain enables inter-domain communication, it SHOULD enable high security.

Administrators might want to require use of SASL for server-to-server communication to ensure both authentication and confidentiality (e.g., on an organization's private network). Compliant implementations SHOULD support SASL for this purpose.

Server-to-server communication MUST NOT proceed until the DNS hostnames asserted by both servers have been resolved as specified under [Section 4 \(TCP Binding\)](#).

15.5. Order of Layers

[TOC](#)

The order of layers in which protocols MUST be stacked is:

1. TCP
2. TLS
3. SASL
4. XMPP

The rationale for this order is that [\[TCP\] \(Postel, J., "Transmission Control Protocol," September 1981.\)](#) is the base connection layer used by all of the protocols stacked on top of TCP, [\[TLS\] \(Dierks, T. and E. Rescorla, "The Transport Layer Security \(TLS\) Protocol Version 1.2," August 2008.\)](#) is often provided at the operating system layer, [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#) is often provided at the application layer, and XMPP is the application itself.

15.6. Mandatory-to-Implement Technologies

[TOC](#)

At a minimum, all implementations MUST support the following mechanisms:

for confidentiality only: TLS (using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher)

for both confidentiality and authentication: TLS plus the SASL PLAIN mechanism (See [\[PLAIN\] \(Zeilenga, K., "The PLAIN Simple Authentication and Security Layer \(SASL\) Mechanism," August 2006.\)](#)) for password-based authentication and TLS plus the SASL EXTERNAL mechanism (see Appendix A of [\[SASL\] \(Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer \(SASL\)," June 2006.\)](#)) for non-password-based authentication (using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher supporting peer certificates)

Naturally, implementations MAY support other ciphers with TLS and MAY support other SASL mechanisms.

Note: The use of TLS plus SASL PLAIN replaces the SASL DIGEST-MD5 mechanism as XMPP's mandatory-to-implement password-based method for authentication. For backward-compatibility, implementations are encouraged to continue supporting the SASL DIGEST-MD5 mechanism as specified in [\[DIGEST-MD5\] \(Leach, P. and C. Newman, "Using Digest Authentication as a SASL Mechanism," May 2000.\)](#). Refer to [\[PLAIN\] \(Zeilenga, K., "The PLAIN Simple Authentication and Security Layer \(SASL\) Mechanism," August 2006.\)](#) for important security considerations related to the SASL PLAIN mechanism.

15.7. SASL Downgrade Attacks

[TOC](#)

Because the initiating entity chooses an acceptable SASL mechanism from the list presented by the receiving entity, the initiating entity depends on the receiving entity's list for authentication. This dependency introduces the possibility of a downgrade attack if an attacker can gain control of the channel and therefore present a weak list of mechanisms. To prevent this attack, the parties SHOULD protect the channel using TLS before attempting SASL negotiation.

[TOC](#)

15.8. Lack of SASL Channel Binding to TLS

The SASL framework itself does not provide a method for binding SASL authentication to a security layer providing confidentiality and integrity protection that was negotiated at a lower layer. Such a binding is known as a "channel binding" (see [\[CHANNEL\] \(Williams, N., "On the Use of Channel Bindings to Secure Channels," November 2007.\)](#)). Some SASL mechanisms provide channel bindings. However, if a SASL mechanism does not provide a channel binding, then the mechanism cannot provide a way to verify that the source and destination end points to which the lower layer's security is bound are equivalent to the end points that SASL is authenticating; furthermore, if the end points are not identical, then the lower layer's security cannot be trusted to protect data transmitted between the SASL-authenticated entities. In such a situation, a SASL security layer SHOULD be negotiated that effectively ignores the presence of the lower-layer security.

15.9. Use of base64 in SASL

[TOC](#)

Both the client and the server MUST verify any base64 data received during [SASL negotiation \(SASL Negotiation\)](#). An implementation MUST reject (not ignore) any characters that are not explicitly allowed by the base64 alphabet; this helps to guard against creation of a covert channel that could be used to "leak" information.

An implementation MUST NOT break on invalid input and MUST reject any sequence of base64 characters containing the pad ('=') character if that character is included as something other than the last character of the data (e.g., "=AAA" or "BBBB=CCC"); this helps to guard against buffer overflow attacks and other attacks on the implementation.

While base 64 encoding visually hides otherwise easily recognized information (such as passwords), it does not provide any computational confidentiality.

All uses of base 64 encoding MUST follow the definition in Section 4 of [\[BASE64\] \(Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," October 2006.\)](#) and padding bits MUST be set to zero.

15.10. Stringprep Profiles

[TOC](#)

XMPP makes use of the [\[NAMEPREP\] \(Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names \(IDN\)," March 2003.\)](#) profile of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#) for processing of domain identifiers; for security considerations related to Nameprep, refer to the appropriate section of

[\[NAMEPREP\]](#) ([Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names \(IDN\)," March 2003.](#)).

In addition, XMPP defines two profiles of [\[STRINGPREP\]](#) ([Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.](#)): [Nodeprep \(Nodeprep\)](#) for node identifiers and [Resourceprep \(Resourceprep\)](#) for resource identifiers.

The Unicode and ISO/IEC 10646 repertoires have many characters that look similar. In many cases, users of security protocols might perform visual matching, such as when comparing the names of trusted third parties. Because it is impossible to map similar-looking characters without a great deal of context (such as knowing the fonts used), stringprep does nothing to map similar-looking characters together, nor to prohibit some characters because they look like others.

A node identifier can be employed as one part of an entity's address in XMPP. One common usage is as the username of an instant messaging user; another is as the name of a multi-user conference room; and many other kinds of entities could use node identifiers as part of their addresses. The security of such services could be compromised based on different interpretations of the internationalized node identifier; for example, a user entering a single internationalized node identifier could access another user's account information, or a user could gain access to a hidden or otherwise restricted chat room or service.

A resource identifier can be employed as one part of an entity's address in XMPP. One common usage is as the name for an instant messaging user's connected resource; another is as the nickname of a user in a multi-user conference room; and many other kinds of entities could use resource identifiers as part of their addresses. The security of such services could be compromised based on different interpretations of the internationalized resource identifier; for example, a user could attempt to initiate multiple connections with the same name, or a user could send a message to someone other than the intended recipient in a multi-user conference room.

15.11. Address Spoofing

[TOC](#)

As discussed in [\[XEP-0165\]](#) ([Saint-Andre, P., "Best Practices to Prevent JID Mimicking," July 2007.](#)), there are two forms of address spoofing: forging and mimicking.

15.11.1. Address Forging

[TOC](#)

In the context of XMPP technologies, address forging occurs when an entity is able to generate an XML stanza whose 'from' address does not correspond to the account credentials with which the entity

authenticated onto the network (or an authorization identity provided during [SASL negotiation \(SASL Negotiation\)](#)). For example, address forging occurs if an entity that authenticated as "juliet@im.example.com" is able to send XML stanzas from "nurse@im.example.com" or "romeo@example.net".

Address forging is difficult in XMPP systems, given the requirement for sending servers to stamp 'from' addresses and for receiving servers to verify sending domains via server-to-server authentication. However, address forging is not impossible, since a rogue server could forge JIDs at the sending domain by ignoring the stamping requirement. A rogue server could even forge JIDs at other domains by means of a DNS poisoning attack if [\[DNSSEC\] \(Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements," March 2005.\)](#) is not used. This specification does not define methods for discovering or counteracting such rogue servers.

15.11.2. Address Mimicking

[TOC](#)

Address mimicking occurs when an entity provides legitimate authentication credentials for and sends XML stanzas from an account whose JID appears to a human user to be the same as another JID. For example, in some XMPP clients the address "paypa1@example.org" (spelled with the number one as the final character of the node identifier) might appear to be the same as "paypal@example.org" (spelled with the lower-case version of the letter "l"), especially on casual visual inspection; this phenomenon is sometimes called "typejacking". A more sophisticated example of address mimicking might involve the use of characters from outside the US-ASCII range, such as the Cherokee characters U+13DA U+13A2 U+13B5 U+13AC U+13A2 U+13AC U+13D2 instead of the US-ASCII characters "STPETER".

In some examples of address mimicking, it is unlikely that the average user could tell the difference between the real JID and the fake JID. (Naturally, there is no way to distinguish with full certainty which is the fake JID and which is the real JID; in some communication contexts, the JID with Cherokee characters might be the real JID and the JID with US-ASCII characters might thus appear to be the fake JID.) Because JIDs can contain almost any Unicode character, it can be relatively easy to mimic some JIDs in XMPP systems. The possibility of address mimicking introduces security vulnerabilities of the kind that have also plagued the World Wide Web, specifically the phenomenon known as phishing. Mimicked addresses that involve characters from only one character set or from the character set typically employed by a particular user are not easy to combat (e.g., the simple typejacking attack previously described, which relies on a surface similarity between the characters "1" and "l" in some presentations). However, mimicked addresses that involve characters from more than one character set, or from a

character set not typically employed by a particular user, can be mitigated somewhat through intelligent presentation. In particular, every human user of an XMPP technology presumably has a preferred language (or, in some cases, a small set of preferred languages), which an XMPP application SHOULD gather either explicitly from the user or implicitly via the operating system of the user's device. Furthermore, every language has a range (or a small set of ranges) of characters normally used to represent that language in textual form. Therefore, an XMPP application SHOULD warn the user when presenting a JID that uses characters outside the normal range of the user's preferred language(s). This recommendation is not intended to discourage communication across language communities; instead, it recognizes the existence of such language communities and encourages due caution when presenting unfamiliar character sets to human users. For more detailed recommendations regarding prevention of address mimicking in XMPP systems, refer to [\[XEP-0165\] \(Saint-Andre, P., "Best Practices to Prevent JID Mimicking," July 2007.\)](#).

15.12. Firewalls

[TOC](#)

Communication using XMPP normally occurs over TCP connections on port 5222 (client-to-server) or port 5269 (server-to-server), as registered with the IANA (see [Section 16 \(IANA Considerations\)](#)). Use of these well-known ports allows administrators to easily enable or disable XMPP activity through existing and commonly-deployed firewalls.

15.13. Denial of Service

[TOC](#)

[\[DOS\] \(Handley, M., Rescorla, E., and IAB, "Internet Denial-of-Service Considerations," December 2006.\)](#) defines denial of service as follows:

A Denial-of-Service (DoS) attack is an attack in which one or more machines target a victim and attempt to prevent the victim from doing useful work. The victim can be a network server, client or router, a network link or an entire network, an individual Internet user or a company doing business using the Internet, an Internet Service Provider (ISP), country, or any combination of or variant on these.

[\[XEP-0205\] \(Saint-Andre, P., "Best Practices to Discourage Denial of Service Attacks," July 2007.\)](#) provides a detailed discussion of potential denial of service attacks against XMPP systems and best practices for preventing such attacks. The recommendations include:

1. A server implementation SHOULD enable a server administrator to limit the number of TCP connections that it will accept from a given IP address at any one time. If an entity attempts to connect but the maximum number of TCP connections has been reached, the receiving server MUST NOT allow the new connection to proceed.
2. A server implementation SHOULD enable a server administrator to limit the number of TCP connection attempts that it will accept from a given IP address in a given time period. (While it is possible to limit the number of connections at the TCP layer rather than at the XMPP application layer, this is not advisable because limits at the TCP layer might result in an inability to access non-XMPP services.) If an entity attempts to connect but the maximum number of connections has been reached, the receiving server MUST NOT allow the new connection to proceed.
3. A server MUST NOT process XML stanzas from clients that have not yet provided appropriate authentication credentials and MUST NOT process XML stanzas from peer servers whose identity it has not either authenticated via SASL or weakly verified via server dialback (see [\[XEP-0220\] \(Saint-Andre, P. and J. Miller, "Server Dialback," October 2008.\)](#)).
4. A server implementation SHOULD enable a server administrator to limit the number of connected resources it will allow an account to bind at any one time. If a client attempts to bind a resource but it has already reached the configured number of allowable resources, the receiving server MUST return a <resource-constraint/> stanza error.
5. A server implementation SHOULD enable a server administrator to limit the size of stanzas it will accept from a connected client or peer server. If a connected resource or peer server sends a stanza that violates the upper limit, the receiving server SHOULD NOT process the stanza and instead SHOULD return a <not-allowed/> stanza error. Alternatively (e.g., if the sender has sent an egregiously large stanza), the server MAY instead return a <policy-violation/> stream error.
6. A server implementation SHOULD enable a server administrator to limit the number of XML stanzas that a connected client is allowed to send to distinct recipients within a given time period. If a connected client sends too many stanzas to distinct recipients in a given time period, the receiving server SHOULD NOT process the stanza and instead SHOULD return an <unexpected-request/> stanza error.

7. A server implementation SHOULD enable a server administrator to limit the amount of bandwidth it will allow a connected client or peer server to use in a given time period.
8. A server implementation MAY enable a server administrator to limit the types of stanzas (based on the extended content "payload") that it will allow a connected resource or peer server send over an active connection. Such limits and restrictions are a matter of deployment policy.
9. A server implementation MAY refuse to route or deliver any stanza that it considers to be abusive, with or without returning an error to the sender.

For more detailed recommendations regarding denial of service attacks in XMPP systems, refer to [\[XEP-0205\] \(Saint-Andre, P., "Best Practices to Discourage Denial of Service Attacks," July 2007.\)](#).

15.14. Presence Leaks

[TOC](#)

One of the core aspects of XMPP is presence: information about the network availability of an XMPP entity (i.e., whether the entity is currently online or offline). A PRESENCE LEAK occurs when an entity's network availability is inadvertently and involuntarily revealed to a second entity that is not authorized to know the first entity's network availability.

Although presence is discussed more fully in [\[rfc3921bis\] \(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#), it is important to note that an XMPP server MUST NOT leak presence. In particular at the core XMPP level, real-time addressing and network availability is associated with a specific connected resource; therefore, any disclosure of a connected resource's full JID comprises a presence leak. To help prevent such a presence leak, a server MUST NOT return different stanza errors if a potential attacker sends XML stanzas to the entity's bare JID (<node@domain>) or full JID (<node@domain/resource>).

15.15. Directory Harvesting

[TOC](#)

To help prevent directory harvesting attacks, a server MUST NOT return different stanza errors if a potential attacker sends XML stanzas to an existing entity or a nonexistent entity.

16. IANA Considerations

[TOC](#)

The following sections update the registrations provided in [\[RFC3920\] \(Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol \(XMPP\): Core," October 2004.\)](#).

16.1. XML Namespace Name for TLS Data

[TOC](#)

A URN sub-namespace for STARTTLS negotiation data in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows. (This namespace name adheres to the format defined in [\[XML-REG\] \(Mealling, M., "The IETF XML Registry," January 2004.\)](#).)

URI: urn:ietf:params:xml:ns:xmpp-tls

Specification: XXXX

Description: This is the XML namespace name for STARTTLS negotiation data in the Extensible Messaging and Presence Protocol (XMPP) as defined by XXXX.

Registrant Contact: IETF, XMPP Working Group, <xmppwg@xmpp.org>

16.2. XML Namespace Name for SASL Data

[TOC](#)

A URN sub-namespace for SASL negotiation data in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows. (This namespace name adheres to the format defined in [\[XML-REG\] \(Mealling, M., "The IETF XML Registry," January 2004.\)](#).)

URI: urn:ietf:params:xml:ns:xmpp-sasl

Specification: XXXX

Description: This is the XML namespace name for SASL negotiation data in the Extensible Messaging and Presence Protocol (XMPP) as defined by XXXX.

Registrant Contact: IETF, XMPP Working Group, <xmppwg@xmpp.org>

16.3. XML Namespace Name for Stream Errors

[TOC](#)

A URN sub-namespace for stream error data in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows. (This namespace name adheres to the format defined in [\[XML-REG\] \(Mealling, M., "The IETF XML Registry," January 2004.\)](#).)

URI: urn:ietf:params:xml:ns:xmpp-streams

Specification: XXXX

Description: This is the XML namespace name for stream error data in the Extensible Messaging and Presence Protocol (XMPP) as defined by XXXX.

Registrant Contact: IETF, XMPP Working Group, <xmppwg@xmpp.org>

16.4. XML Namespace Name for Resource Binding

[TOC](#)

A URN sub-namespace for resource binding in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows. (This namespace name adheres to the format defined in [\[XML-REG\] \(Mealling, M., "The IETF XML Registry," January 2004.\)](#).)

URI: urn:ietf:params:xml:ns:xmpp-bind

Specification: XXXX

Description: This is the XML namespace name for resource binding in the Extensible Messaging and Presence Protocol (XMPP) as defined by XXXX.

Registrant Contact: IETF, XMPP Working Group, <xmppwg@xmpp.org>

16.5. XML Namespace Name for Stanza Errors

[TOC](#)

A URN sub-namespace for stanza error data in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows. (This namespace name adheres to the format defined in [\[XML-REG\] \(Mealling, M., "The IETF XML Registry," January 2004.\)](#).)

URI: urn:ietf:params:xml:ns:xmpp-stanzas

Specification:

XXXX

Description: This is the XML namespace name for stanza error data in the Extensible Messaging and Presence Protocol (XMPP) as defined by XXXX.

Registrant Contact: IETF, XMPP Working Group, <xmppwg@xmpp.org>

16.6. Nodeprep Profile of Stringprep[TOC](#)

The Nodeprep profile of stringprep is defined under [Nodeprep \(Nodeprep\)](#). The IANA has registered Nodeprep in the stringprep profile registry.

Name of this profile:

Nodeprep

RFC in which the profile is defined:

XXXX

Indicator whether or not this is the newest version of the profile:

This is the first version of Nodeprep

16.7. Resourceprep Profile of Stringprep[TOC](#)

The Resourceprep profile of stringprep is defined under [Resourceprep \(Resourceprep\)](#). The IANA has registered Resourceprep in the stringprep profile registry.

Name of this profile:

Resourceprep

RFC in which the profile is defined:

XXXX

Indicator whether or not this is the newest version of the profile:

This is the first version of Resourceprep

16.8. GSSAPI Service Name

[TOC](#)

The IANA has registered "xmpp" as a [GSSAPI \(Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1," January 2000.\)](#) [GSS-API] service name, as defined under [Section 7.5 \(SASL Definition\)](#).

16.9. Port Numbers

[TOC](#)

The IANA has registered "xmpp-client" and "xmpp-server" as keywords for [\[TCP\] \(Postel, J., "Transmission Control Protocol," September 1981.\)](#) ports 5222 and 5269 respectively.
These ports SHOULD be used for client-to-server and server-to-server communications respectively, but other ports MAY be used.

17. References

[TOC](#)

17.1. Normative References

[TOC](#)

[ABNF]	Crocker, D. and P. Overell, " Augmented BNF for Syntax Specifications: ABNF ," STD 68, RFC 5234, January 2008 (TXT).
[BASE64]	Josefsson, S., " The Base16, Base32, and Base64 Data Encodings ," RFC 4648, October 2006 (TXT).
[CHARSET]	Alvestrand, H. , " IETF Policy on Character Sets and Languages ," BCP 18, RFC 2277, January 1998 (TXT , HTML , XML).
[DNS]	Mockapetris, P., " Domain names - implementation and specification ," STD 13, RFC 1035, November 1987 (TXT).
[DNS-SRV]	Gulbrandsen, A. , Vixie, P., and L. Esibov , " A DNS RR for specifying the location of services (DNS SRV) ," RFC 2782, February 2000 (TXT).
[IDNA]	Faltstrom, P., Hoffman, P., and A. Costello, " Internationalizing Domain Names in Applications (IDNA) ," RFC 3490, March 2003 (TXT).
[LANGTAGS]	Phillips, A. and M. Davis, " Tags for Identifying Languages ," BCP 47, RFC 4646, September 2006 (TXT).

[NAMEPREP]	Hoffman, P. and M. Blanchet, " Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN) ," RFC 3491, March 2003 (TXT).
[PLAIN]	Zeilenga, K., " The PLAIN Simple Authentication and Security Layer (SASL) Mechanism ," RFC 4616, August 2006 (TXT).
[RANDOM]	Eastlake, D., Schiller, J., and S. Crocker, " Randomness Requirements for Security ," BCP 106, RFC 4086, June 2005 (TXT).
[SASL]	Melnikov, A. and K. Zeilenga, " Simple Authentication and Security Layer (SASL) ," RFC 4422, June 2006 (TXT).
[STRINGPREP]	Hoffman, P. and M. Blanchet, " Preparation of Internationalized Strings ("stringprep") ," RFC 3454, December 2002 (TXT).
[TCP]	Postel, J., " Transmission Control Protocol ," STD 7, RFC 793, September 1981 (TXT).
[TERMS]	Bradner, S. , " Key words for use in RFCs to Indicate Requirement Levels ," BCP 14, RFC 2119, March 1997 (TXT , HTML , XML).
[TLS]	Dierks, T. and E. Rescorla, " The Transport Layer Security (TLS) Protocol Version 1.2 ," RFC 5246, August 2008 (TXT).
[UCS2]	International Organization for Standardization, "Information Technology - Universal Multiple-octet coded Character Set (UCS) - Amendment 2: UCS Transformation Format 8 (UTF-8)," ISO Standard 10646-1 Addendum 2, October 1996.
[UNICODE]	The Unicode Consortium, "The Unicode Standard, Version 3.2.0," 2000. The Unicode Standard, Version 3.2.0 is defined by The Unicode Standard, Version 3.0 (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), as amended by the Unicode Standard Annex #27: Unicode 3.1 (http://www.unicode.org/reports/tr27/) and by the Unicode Standard Annex #28: Unicode 3.2 (http://www.unicode.org/reports/tr28/).
[UTF-8]	Yergeau, F., " UTF-8, a transformation format of ISO 10646 ," STD 63, RFC 3629, November 2003 (TXT).
[UUID]	Leach, P. , Mealling, M. , and R. Salz , " A Universally Unique Identifier (UUID) URN Namespace ," RFC 4122, July 2005 (TXT , HTML , XML).
[URI]	Berners-Lee, T. , Fielding, R. , and L. Masinter , " Uniform Resource Identifier (URI): Generic Syntax ," STD 66, RFC 3986, January 2005 (TXT , HTML , XML).
[X509]	Housley, R., Polk, W., Ford, W., and D. Solo, " Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile ," RFC 3280, April 2002 (TXT).

[X509-SRV]	Santesson, S., " Internet X.509 Public Key Infrastructure Subject Alternative Name for Expression of Service Name ," RFC 4985, August 2007 (TXT).
[XML]	Paoli, J., Maler, E., Sperberg-McQueen, C., Yergeau, F., and T. Bray, " Extensible Markup Language (XML) 1.0 (Fourth Edition) ," World Wide Web Consortium Recommendation REC-xml-20060816, August 2006 (HTML).
[XML-NAMES]	Layman, A., Hollander, D., Tobin, R., and T. Bray, " Namespaces in XML 1.1 (Second Edition) ," World Wide Web Consortium Recommendation REC-xml-names11-20060816, August 2006 (HTML).

17.2. Informative References

[TOC](#)

[ACAP]	Newman, C. and J. Myers , " ACAP -- Application Configuration Access Protocol ," RFC 2244, November 1997 (TXT).
[ANONYMOUS]	Zeilenga, K. , " Anonymous Simple Authentication and Security Layer (SASL) Mechanism ," RFC 4505, June 2006 (TXT).
[ASN.1]	CCITT, "Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1)," 1988.
[CHANNEL]	Williams, N. , " On the Use of Channel Bindings to Secure Channels ," RFC 5056, November 2007 (TXT).
[DIGEST-MD5]	Leach, P. and C. Newman , " Using Digest Authentication as a SASL Mechanism ," RFC 2831, May 2000 (TXT).
[DNSSEC]	Arends, R. , Austein, R. , Larson, M. , Massey, D. , and S. Rose , " DNS Security Introduction and Requirements ," RFC 4033, March 2005 (TXT).
[DNS-TXT]	Rosenbaum, R. , " Using the Domain Name System To Store Arbitrary String Attributes ," RFC 1464, May 1993 (TXT).
[DOS]	Handley, M. , Rescorla, E. , and IAB, " Internet Denial-of-Service Considerations ," RFC 4732, December 2006 (TXT).
[GSS-API]	Linn, J. , " Generic Security Service Application Program Interface Version 2, Update 1 ," RFC 2743, January 2000 (TXT).
[HTTP]	Fielding, R. , Gettys, J. , Mogul, J. , Frystyk, H. , Masinter, L. , Leach, P. , and T. Berners-Lee , " Hypertext Transfer Protocol -- HTTP/1.1 ," RFC 2616, June 1999 (TXT , PS , PDF , HTML , XML).
[IMAP]	Crispin, M. , " INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1 ," RFC 3501, March 2003 (TXT).
[IMP-REQS]	

	Day, M., Aggarwal, S., and J. Vincent, "Instant Messaging / Presence Protocol Requirements," RFC 2779, February 2000 (TXT).
[IRI]	Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)," RFC 3987, January 2005 (TXT).
[LINKLOCAL]	Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses," RFC 3927, May 2005 (TXT).
[MAILBOXES]	Crocker, D., "MAILBOX NAMES FOR COMMON SERVICES, ROLES AND FUNCTIONS," RFC 2142, May 1997 (TXT, HTML, XML).
[POP3]	Myers, J. and M. Rose, "Post Office Protocol - Version 3," STD 53, RFC 1939, May 1996 (TXT).
[PUNYCODE]	Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)," RFC 3492, March 2003 (TXT).
[RFC3920]	Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 3920, October 2004 (TXT, HTML, XML).
[RFC3921]	Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," RFC 3921, October 2004 (TXT, HTML, XML).
[rfc3921bis]	Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," draft-saintandre-rfc3921bis-08 (work in progress), March 2009 (TXT).
[SMTP]	Klensin, J., "Simple Mail Transfer Protocol," RFC 2821, April 2001 (TXT).
[URN-OID]	Mealling, M., "A URN Namespace of Object Identifiers," RFC 3061, February 2001 (TXT).
[USINGTLS]	Newman, C., "Using TLS with IMAP, POP3 and ACAP," RFC 2595, June 1999 (TXT).
[XEP-0001]	Saint-Andre, P., "XMPP Extension Protocols," XSF XEP 0001, January 2008.
[XEP-0045]	Saint-Andre, P., "Multi-User Chat," XSF XEP 0045, July 2007.
[XEP-0060]	Millard, P., Saint-Andre, P., and R. Meijer, "Publish-Subscribe," XSF XEP 0060, September 2007.
[XEP-0071]	Saint-Andre, P., "XHTML-IM," XSF XEP 0071, September 2007.
[XEP-0077]	Saint-Andre, P., "In-Band Registration," XSF XEP 0077, January 2006.
[XEP-0124]	Paterson, I., Smith, D., and P. Saint-Andre, "Bidirectional-streams Over Synchronous HTTP (BOSH)," XSF XEP 0124, February 2007.
[XEP-0156]	Hildebrand, J. and P. Saint-Andre, "Discovering Alternative XMPP Connection Methods," XSF XEP 0156, June 2007.

[XEP-0165]	Saint-Andre, P. , " Best Practices to Prevent JID Mimicking ," XSF XEP 0165, July 2007.
[XEP-0174]	Saint-Andre, P. , " Link-Local Messaging ," XSF XEP 0174, September 2007.
[XEP-0175]	Saint-Andre, P. , " Best Practices for Use of SASL ANONYMOUS ," XSF XEP 0175, September 2006.
[XEP-0178]	Saint-Andre, P. and P. Millard , " Best Practices for Use of SASL EXTERNAL with Certificates ," XSF XEP 0178, February 2007.
[XEP-0205]	Saint-Andre, P. , " Best Practices to Discourage Denial of Service Attacks ," XSF XEP 0205, July 2007.
[XEP-0206]	Paterson, I. , " XMPP Over BOSH ," XSF XEP 0206, June 2007.
[XEP-0220]	Saint-Andre, P. and J. Miller , " Server Dialback ," XSF XEP 0220, October 2008.
[XEP-0246]	Saint-Andre, P. , " End-to-End XML Streams ," XSF XEP 0246, June 2008.
[XML-FRAG]	Grosso, P. and D. Veillard, " XML Fragment Interchange ," World Wide Web Consortium CR CR-xml-fragment-20010212, February 2001 (HTML).
[XML-REG]	Mealling, M., " The IETF XML Registry ," BCP 81, RFC 3688, January 2004 (TXT).
[XML-SCHEMA]	Thompson, H., Maloney, M., Mendelsohn, N., and D. Beech, " XML Schema Part 1: Structures Second Edition ," World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004 (HTML).
[XMPP-URI]	Saint-Andre, P., " Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP) ," RFC 5122, February 2008 (TXT).

Appendix A. Nodeprep

[TOC](#)

A.1. Introduction

[TOC](#)

This appendix defines the "Nodeprep" profile of stringprep. As such, it specifies processing rules that will enable users to enter internationalized node identifiers in the Extensible Messaging and Presence Protocol (XMPP) and have the highest chance of getting the content of the strings correct. (An XMPP node identifier is the optional portion of an XMPP address that precedes an XMPP domain identifier and the '@' separator; it is often but not exclusively

associated with an instant messaging username.) These processing rules are intended only for XMPP node identifiers and are not intended for arbitrary text or any other aspect of an XMPP address. This profile defines the following, as required by [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#):

- *The intended applicability of the profile: internationalized node identifiers within XMPP
- *The character repertoire that is the input and output to stringprep: Unicode 3.2, specified in Section 2 of this Appendix
- *The mappings used: specified in Section 3
- *The Unicode normalization used: specified in Section 4
- *The characters that are prohibited as output: specified in Section 5
- *Bidirectional character handling: specified in Section 6

A.2. Character Repertoire

[TOC](#)

This profile uses Unicode 3.2 with the list of unassigned code points being Table A.1, both defined in Appendix A of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

A.3. Mapping

[TOC](#)

This profile specifies mapping using the following tables from [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#):

Table B.1

Table B.2

[TOC](#)

A.4. Normalization

This profile specifies the use of Unicode normalization form KC, as described in [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

A.5. Prohibited Output

[TOC](#)

This profile specifies the prohibition of using the following tables from [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

Table C.1.1

Table C.1.2

Table C.2.1

Table C.2.2

Table C.3

Table C.4

Table C.5

Table C.6

Table C.7

Table C.8

Table C.9

In addition, the following additional Unicode characters are also prohibited:

U+0022 (QUOTATION MARK), i.e., "

U+0026 (AMPERSAND), i.e., &

U+0027 (APOSTROPHE), i.e., '

U+002F (SOLIDUS), i.e., /

U+003A (COLON), i.e., :

U+003C (LESS-THAN SIGN), i.e., <

U+003E (GREATER-THAN SIGN), i.e., >

U+0040 (COMMERCIAL AT), i.e., @

A.6. Bidirectional Characters

[TOC](#)

This profile specifies checking bidirectional strings, as described in Section 6 of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

A.7. Notes

[TOC](#)

Because the additional characters prohibited by Nodeprep are prohibited after normalization, an implementation MUST NOT enable a human user to input any Unicode code point whose decomposition includes those characters; such code points include but are not necessarily limited to the following (refer to [\[UNICODE\] \(The Unicode Consortium, "The Unicode Standard, Version 3.2.0," 2000.\)](#) for complete information).

- *U+2100 (ACCOUNT OF)
- *U+2101 (ADDRESSED TO THE SUBJECT)
- *U+2105 (CARE OF)
- *U+2106 (CADA UNA)
- *U+226E (NOT LESS-THAN)
- *U+226F (NOT GREATER-THAN)

*U+2A74 (DOUBLE COLON EQUAL)

*U+FE13 (SMALL COLON)

*U+FE60 (SMALL AMPERSAND)

*U+FE64 (SMALL LESS-THAN SIGN)

*U+FE65 (SMALL GREATER-THAN SIGN)

*U+FE6B (SMALL COMMERCIAL AT)

*U+FF02 (FULLWIDTH QUOTATION MARK)

*U+FF06 (FULLWIDTH AMPERSAND)

*U+FF07 (FULLWIDTH APOSTROPHE)

*U+FF0F (FULLWIDTH SOLIDUS)

*U+FF1A (FULLWIDTH COLON)

*U+FF1C (FULLWIDTH LESS-THAN SIGN)

*U+FF1E (FULLWIDTH GREATER-THAN SIGN)

*U+FF20 (FULLWIDTH COMMERCIAL AT)

Appendix B. Resourceprep

[TOC](#)

B.1. Introduction

[TOC](#)

This appendix defines the "Resourceprep" profile of stringprep. As such, it specifies processing rules that will enable users to enter internationalized resource identifiers in the Extensible Messaging and Presence Protocol (XMPP) and have the highest chance of getting the content of the strings correct. (An XMPP resource identifier is the optional portion of an XMPP address that follows an XMPP domain identifier and the '/' separator.) These processing rules are intended only for XMPP resource identifiers and are not intended for arbitrary text or any other aspect of an XMPP address.

This profile defines the following, as required by [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#):

- *The intended applicability of the profile: internationalized resource identifiers within XMPP
- *The character repertoire that is the input and output to stringprep: Unicode 3.2, specified in Section 2 of this Appendix
- *The mappings used: specified in Section 3
- *The Unicode normalization used: specified in Section 4
- *The characters that are prohibited as output: specified in Section 5
- *Bidirectional character handling: specified in Section 6

B.2. Character Repertoire

[TOC](#)

This profile uses Unicode 3.2 with the list of unassigned code points being Table A.1, both defined in Appendix A of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

B.3. Mapping

[TOC](#)

This profile specifies mapping using the following tables from [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#):

| Table B.1

B.4. Normalization

[TOC](#)

This profile specifies the use of Unicode normalization form KC, as described in [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

B.5. Prohibited Output

[TOC](#)

This profile specifies the prohibition of using the following tables from [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

Table C.1.2

Table C.2.1

Table C.2.2

Table C.3

Table C.4

Table C.5

Table C.6

Table C.7

Table C.8

Table C.9

B.6. Bidirectional Characters

[TOC](#)

This profile specifies checking bidirectional strings, as described in Section 6 of [\[STRINGPREP\] \(Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings \("stringprep"\)," December 2002.\)](#).

Appendix C. XML Schemas

[TOC](#)

Because validation of XML streams and stanzas is optional, the following XML schemas are provided for descriptive purposes only. These schemas are not normative.

The following schemas formally define various XML namespaces used in the core XMPP protocols, in conformance with [\[XML-SCHEMA\] \(Thompson, H., Maloney, M., Mendelsohn, N., and D. Beech, "XML Schema Part 1: Structures Second Edition," October 2004.\)](#). For schemas defining the 'jabber:client' and 'jabber:server' namespaces, refer to [\[rfc3921bis\]](#)

[\(Saint-Andre, P., "Extensible Messaging and Presence Protocol \(XMPP\): Instant Messaging and Presence," March 2009.\)](#).

[TOC](#)

C.1. Streams Namespace

```

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

<xs:schema
  xmlns:xs='http://www.w3.org/2001/XMLSchema'
  targetNamespace='http://etherx.jabber.org/streams'
  xmlns='http://etherx.jabber.org/streams'
  elementFormDefault='unqualified'>

  <xs:import namespace='jabber:client' />
  <xs:import namespace='jabber:server' />
  <xs:import namespace='urn:ietf:params:xml:ns:xmpp-sasl' />
  <xs:import namespace='urn:ietf:params:xml:ns:xmpp-streams' />
  <xs:import namespace='urn:ietf:params:xml:ns:xmpp-tls' />

  <xs:element name='stream'>
    <xs:complexType>
      <xs:sequence xmlns:client='jabber:client'
        xmlns:server='jabber:server'
        <xs:element ref='features' minOccurs='0' maxOccurs='1' />
        <xs:any namespace='urn:ietf:params:xml:ns:xmpp-tls'
          minOccurs='0'
          maxOccurs='unbounded' />
        <xs:any namespace='urn:ietf:params:xml:ns:xmpp-sasl'
          minOccurs='0'
          maxOccurs='unbounded' />
        <xs:choice minOccurs='0' maxOccurs='1'>
          <xs:choice minOccurs='0' maxOccurs='unbounded'>
            <xs:element ref='client:message' />
            <xs:element ref='client:presence' />
            <xs:element ref='client:iq' />
          </xs:choice>
          <xs:choice minOccurs='0' maxOccurs='unbounded'>
            <xs:element ref='server:message' />
            <xs:element ref='server:presence' />
            <xs:element ref='server:iq' />
            <xs:element ref='db:result' />
            <xs:element ref='db:verify' />
          </xs:choice>
        </xs:choice>
        <xs:element ref='error' minOccurs='0' maxOccurs='1' />
      </xs:sequence>
      <xs:attribute name='from' type='xs:string' use='optional' />
      <xs:attribute name='id' type='xs:string' use='optional' />
      <xs:attribute name='to' type='xs:string' use='optional' />
      <xs:attribute name='version' type='xs:decimal' use='optional' />
      <xs:attribute ref='xml:lang' use='optional' />
    </xs:complexType>
  </xs:element>

```

```
<xs:element name='features'>
  <xs:complexType>
    <xs:any namespace='##other' />
  </xs:complexType>
</xs:element>

<xs:element name='error'>
  <xs:complexType>
    <xs:sequence xmlns:err='urn:ietf:params:xml:ns:xmpp-streams'>
      <xs:group ref='err:streamErrorGroup' />
      <xs:element ref='err:text'
        minOccurs='0'
        maxOccurs='1' />
      <xs:any namespace='##other'
        minOccurs='0'
        maxOccurs='1' />
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```


C.2. Stream Error Namespace

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

```
<xs:schema
```

```
  xmlns:xs='http://www.w3.org/2001/XMLSchema'  
  targetNamespace='urn:ietf:params:xml:ns:xmpp-streams'  
  xmlns='urn:ietf:params:xml:ns:xmpp-streams'  
  elementFormDefault='qualified'>
```

```
  <xs:element name='bad-format' type='empty'/>  
  <xs:element name='bad-namespace-prefix' type='empty'/>  
  <xs:element name='conflict' type='empty'/>  
  <xs:element name='connection-timeout' type='empty'/>  
  <xs:element name='host-gone' type='empty'/>  
  <xs:element name='host-unknown' type='empty'/>  
  <xs:element name='improper-addressing' type='empty'/>  
  <xs:element name='internal-server-error' type='empty'/>  
  <xs:element name='invalid-from' type='empty'/>  
  <xs:element name='invalid-id' type='empty'/>  
  <xs:element name='invalid-namespace' type='empty'/>  
  <xs:element name='invalid-xml' type='empty'/>  
  <xs:element name='not-authorized' type='empty'/>  
  <xs:element name='policy-violation' type='empty'/>  
  <xs:element name='remote-connection-failed' type='empty'/>  
  <xs:element name='resource-constraint' type='empty'/>  
  <xs:element name='restricted-xml' type='empty'/>  
  <xs:element name='see-other-host' type='xs:string'/>  
  <xs:element name='system-shutdown' type='empty'/>  
  <xs:element name='undefined-condition' type='empty'/>  
  <xs:element name='unsupported-encoding' type='empty'/>  
  <xs:element name='unsupported-stanza-type' type='empty'/>  
  <xs:element name='unsupported-version' type='empty'/>  
  <xs:element name='xml-not-well-formed' type='empty'/>
```

```
<xs:group name='streamErrorGroup'>
```

```
  <xs:choice>
```

```
    <xs:element ref='bad-format'/>  
    <xs:element ref='bad-namespace-prefix'/>  
    <xs:element ref='conflict'/>  
    <xs:element ref='connection-timeout'/>  
    <xs:element ref='host-gone'/>  
    <xs:element ref='host-unknown'/>  
    <xs:element ref='improper-addressing'/>  
    <xs:element ref='internal-server-error'/>  
    <xs:element ref='invalid-from'/>  
    <xs:element ref='invalid-id'/>  
    <xs:element ref='invalid-namespace'/>  
    <xs:element ref='invalid-xml'/>  
    <xs:element ref='not-authorized'/>
```

```

    <xs:element ref='policy-violation' />
    <xs:element ref='remote-connection-failed' />
    <xs:element ref='resource-constraint' />
    <xs:element ref='restricted-xml' />
    <xs:element ref='see-other-host' />
    <xs:element ref='system-shutdown' />
    <xs:element ref='undefined-condition' />
    <xs:element ref='unsupported-encoding' />
    <xs:element ref='unsupported-stanza-type' />
    <xs:element ref='unsupported-version' />
    <xs:element ref='xml-not-well-formed' />
  </xs:choice>
</xs:group>

<xs:element name='text'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional' />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:simpleType name='empty'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value='' />
  </xs:restriction>
</xs:simpleType>

</xs:schema>

```

C.3. STARTTLS Namespace

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

<xs:schema
  xmlns:xs='http://www.w3.org/2001/XMLSchema'
  targetNamespace='urn:ietf:params:xml:ns:xmpp-tls'
  xmlns='urn:ietf:params:xml:ns:xmpp-tls'
  elementFormDefault='qualified'>

  <xs:element name='starttls'>
    <xs:complexType>
      <xs:choice minOccurs='0' maxOccurs='1'>
        <xs:element name='optional' type='empty' />
        <xs:element name='required' type='empty' />
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name='proceed' type='empty' />

  <xs:element name='failure' type='empty' />

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value='' />
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

C.4. SASL Namespace

```

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

<xs:schema
  xmlns:xs='http://www.w3.org/2001/XMLSchema'
  targetNamespace='urn:ietf:params:xml:ns:xmpp-sasl'
  xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
  elementFormDefault='qualified'>

  <xs:element name='mechanisms'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='mechanism'
          minOccurs='1'
          maxOccurs='unbounded'
          type='xs:NMTOKEN' />
        <xs:choice minOccurs='0' maxOccurs='1'>
          <xs:element name='optional' type='empty' />
          <xs:element name='required' type='empty' />
        </xs:choice>
        <xs:any namespace='##other'
          minOccurs='0'
          maxOccurs='unbounded' />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='abort' type='empty' />

  <xs:element name='auth'>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base='xs:string'>
          <xs:attribute name='mechanism'
            type='xs:NMTOKEN'
            use='required' />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

  <xs:element name='challenge' type='xs:string' />

  <xs:element name='response' type='xs:string' />

  <xs:element name='success' type='xs:string' />

  <xs:element name='failure'>
    <xs:complexType>
      <xs:sequence>

```

```

<xs:choice minOccurs='0'>
  <xs:element name='aborted' type='empty'/>
  <xs:element name='account-disabled' type='empty'/>
  <xs:element name='credentials-expired' type='empty'/>
  <xs:element name='encryption-required' type='empty'/>
  <xs:element name='incorrect-encoding' type='empty'/>
  <xs:element name='invalid-authzid' type='empty'/>
  <xs:element name='invalid-mechanism' type='empty'/>
  <xs:element name='malformed-request' type='empty'/>
  <xs:element name='mechanism-too-weak' type='empty'/>
  <xs:element name='not-authorized' type='empty'/>
  <xs:element name='temporary-auth-failure' type='empty'/>
  <xs:element name='transition-needed' type='empty'/>
</xs:choice>
<xs:element ref='text' minOccurs='0' maxOccurs='1'/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name='text'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:simpleType name='empty'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value=''/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>

```

C.5. Resource Binding Namespace


```

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

<xs:schema
  xmlns:xs='http://www.w3.org/2001/XMLSchema'
  targetNamespace='urn:ietf:params:xml:ns:xmpp-bind'
  xmlns='urn:ietf:params:xml:ns:xmpp-bind'
  elementFormDefault='qualified'>

  <xs:element name='bind'>
    <xs:complexType>
      <xs:choice>
        <xs:choice>
          <xs:element name='resource' type='resourceType' />
          <xs:element name='jid' type='fullJIDType' />
        </xs:choice>
        <xs:choice>
          <xs:element name='optional' type='empty' />
          <xs:element name='required' type='empty' />
        </xs:choice>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name='unbind'>
    <xs:complexType>
      <xs:choice>
        <xs:choice>
          <xs:element name='resource' type='resourceType' />
        </xs:choice>
        <xs:choice>
          <xs:element name='optional' type='empty' />
          <xs:element name='required' type='empty' />
        </xs:choice>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name='fullJIDType'>
    <xs:restriction base='xs:string'>
      <xs:minLength value='8' />
      <xs:maxLength value='3071' />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name='resourceType'>
    <xs:restriction base='xs:string'>
      <xs:minLength value='1' />
      <xs:maxLength value='1023' />
    </xs:restriction>
  </xs:simpleType>

```

```
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

TOC

C.6. Stanza Error Namespace

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

```
<xs:schema
```

```
  xmlns:xs='http://www.w3.org/2001/XMLSchema'  
  targetNamespace='urn:ietf:params:xml:ns:xmpp-stanzas'  
  xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'  
  elementFormDefault='qualified'>
```

```
  <xs:element name='bad-request' type='empty'/>  
  <xs:element name='conflict' type='empty'/>  
  <xs:element name='feature-not-implemented' type='empty'/>  
  <xs:element name='forbidden' type='empty'/>  
  <xs:element name='gone' type='xs:string'/>  
  <xs:element name='internal-server-error' type='empty'/>  
  <xs:element name='item-not-found' type='empty'/>  
  <xs:element name='jid-malformed' type='empty'/>  
  <xs:element name='not-acceptable' type='empty'/>  
  <xs:element name='not-allowed' type='empty'/>  
  <xs:element name='not-authorized' type='empty'/>  
  <xs:element name='not-modified' type='empty'/>  
  <xs:element name='payment-required' type='empty'/>  
  <xs:element name='recipient-unavailable' type='empty'/>  
  <xs:element name='redirect' type='xs:string'/>  
  <xs:element name='registration-required' type='empty'/>  
  <xs:element name='remote-server-not-found' type='empty'/>  
  <xs:element name='remote-server-timeout' type='empty'/>  
  <xs:element name='resource-constraint' type='empty'/>  
  <xs:element name='service-unavailable' type='empty'/>  
  <xs:element name='subscription-required' type='empty'/>  
  <xs:element name='undefined-condition' type='empty'/>  
  <xs:element name='unexpected-request' type='empty'/>  
  <xs:element name='unknown-sender' type='empty'/>
```

```
<xs:group name='stanzaErrorGroup'>
```

```
  <xs:choice>  
    <xs:element ref='bad-request'/>  
    <xs:element ref='conflict'/>  
    <xs:element ref='feature-not-implemented'/>  
    <xs:element ref='forbidden'/>  
    <xs:element ref='gone'/>  
    <xs:element ref='internal-server-error'/>  
    <xs:element ref='item-not-found'/>  
    <xs:element ref='jid-malformed'/>  
    <xs:element ref='not-acceptable'/>  
    <xs:element ref='not-authorized'/>  
    <xs:element ref='not-allowed'/>  
    <xs:element ref='not-modified'/>  
    <xs:element ref='payment-required'/>
```

```

    <xs:element ref='recipient-unavailable'/>
    <xs:element ref='redirect'/>
    <xs:element ref='registration-required'/>
    <xs:element ref='remote-server-not-found'/>
    <xs:element ref='remote-server-timeout'/>
    <xs:element ref='resource-constraint'/>
    <xs:element ref='service-unavailable'/>
    <xs:element ref='subscription-required'/>
    <xs:element ref='undefined-condition'/>
    <xs:element ref='unexpected-request'/>
    <xs:element ref='unknown-sender'/>
  </xs:choice>
</xs:group>

<xs:element name='text'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:simpleType name='empty'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value=''/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>

```

Appendix D. Contact Addresses

[TOC](#)

Consistent with [\[MAILBOXES\] \(Crocker, D., "MAILBOX NAMES FOR COMMON SERVICES, ROLES AND FUNCTIONS," May 1997.\)](#), an organization that offers an XMPP service SHOULD provide an Internet mailbox of "XMPP" for inquiries related to that service, where the host portion of the resulting mailto URI MUST be the organization's domain, not the domain of the XMPP service itself (e.g., the XMPP service might be offered at im.example.com but the Internet mailbox would be <xmpp@example.com>).

[TOC](#)

Appendix E. Account Provisioning

Account provisioning is out of scope for this specification. Possible methods for account provisioning include account creation by a server administrator and in-band account registration using the 'jabber:iq:register' namespace as documented in [\[XEP-0077\] \(Saint-Andre, P., "In-Band Registration," January 2006.\)](#).

Appendix F. Differences From RFC 3920

[TOC](#)

Based on consensus derived from implementation and deployment experience as well as formal interoperability testing, the following substantive modifications were made from RFC 3920.

- *Corrected the ABNF syntax for JIDs to prevent zero-length node identifiers, domain identifiers, and resource identifiers.
- *Corrected the nameprep processing rules to require use of the UseSTD3ASCIIRules flag.
- *Recommended or mandated use of the 'from' and 'to' attributes on stream headers.
- *More fully specified stream closing handshake.
- *Specified recommended stream reconnection algorithm.
- *Specified return of <restricted-xml/> stream error in response to receipt of prohibited XML features.
- *Specified that TLS plus SASL PLAIN is a mandatory-to-implement technology for client-to-server connections, since implementation of SASL EXTERNAL is uncommon in XMPP clients, in part because underlying security features such as end-user X.509 certificates are not yet widely deployed.
- *Added the <account-disabled/>, <credentials-expired/>, <encryption-required/>, <malformed-request/>, and <transition-needed/> SASL error conditions to handle error flows mistakenly left out of RFC 3920 or discussed in RFC 4422 but not in RFC 2222.
- *More fully specified binding of multiple resources to the same stream.

*Added the <unknown-sender/> stanza error condition to provide appropriate handling of stanzas when multiple resources are bound to the same stream.

*Added the <not-modified/> stanza error condition to enable potential ETags usage.

*Removed unnecessary requirement for escaping of characters that map to certain predefined entities, which do not need to be escaped in XML.

*Clarified process of DNS SRV lookups and fallbacks.

*Clarified handling of SASL security layers.

*Clarified handling of stream features, regularized use of the <required/> child element, and defined use of the <optional/> child element.

*Clarified handling of data that violates the well-formedness definitions for XML 1.0 and XML namespaces.

*Specified security considerations in more detail, especially with regard to presence leaks and denial of service attacks.

*Moved historical documentation of the server dialback protocol from this specification to a separate specification maintained by the XMPP Standards Foundation.

In addition, numerous changes of an editorial nature were made in order to more fully specify and clearly explain XMPP.

Appendix G. Copying Conditions

[TOC](#)

Regarding this entire document or any portion of it, the author makes no guarantees and is not responsible for any damage resulting from its use. The author grants irrevocable permission to anyone to use, modify, and distribute it in any way that does not diminish the rights of anyone else to use, modify, and distribute it, provided that redistributed derivative works do not contain misleading author or version information. Derivative works need not be licensed under similar terms.

Index

[TOC](#)

B	
	Bare JID
C	
	Connected Resource
D	
	Domain Identifier
E	
	Entity
	Error Stanza
	Extended Content
F	
	Full JID
I	
	Initial Stream
	IQ Stanza
J	
	Jabber Identifier
M	
	Message Stanza
N	
	Node Identifier
P	
	Presence Stanza
R	
	Resource Identifier
	Response Stream
S	
	Stream ID
W	
	Whitespace Ping
X	
	XML Stanza
	XML Stream

Author's Address

[TOC](#)

	Peter Saint-Andre
	Cisco
Email:	psaintan@cisco.com