A modular, multi-platform, standards-compliant XMPP library in Microsoft .NET (C#) for Open Instant Messaging - openXMPP

Rahul Agarwal and John C. Linford
December 13, 2005

INTRODUCTION

Instant Messaging popularity has risen phenomenally. Desktop IM systems are now available on most mobile devices, but popular networks and chat-clients like AIM, MSN Messenger and Yahoo Messenger are propriety and have no interoperability. Furthermore, these protocols are often limited to basic chat functionality and cannot be easily extended to add new features.

The Extensible Messaging and Presence Protocol (XMPP), (created initially by the Jabber[1] community) are now formalized by IETF and provide "open Extensible Markup Language (XML) protocols for near-real-time messaging, presence, and request-response services" (RFC3920[2]).

An XMPP based service uses a typical client-server model over a TCP connection. The clients can be registered on different servers and, if gateways are used, can communicate with non-XMPP clients (Figure 1). XMPP servers can route and store client data (i.e. undeliverable messages and rosters (contact lists).

```
C1----S1---S2---C3
       |
C2----+--G1===FN1===FC1
```

```
The symbols are as follows:

C1, C2, C3 = XMPP clients
S1, S2 = XMPP servers
G1 = A gateway that
translates between XMPP and
the protocol(s) used on a
foreign (non-XMPP) messaging
network
FN1 = A foreign messaging
network
FC1 = A client on a foreign
messaging network
```

Figure 1: XMPP Overview

XMPP defines both server-to-server and client-to-server communication. This project is strictly limited to client-to-server communication. Within this scope, we achieved full client compliance as defined in the above RFCs. We will provide the basic features of message exchange with other users, exchange of presence information (RFC 3921, Section 1.2), and the basic communication features of RFC 3920. The openXMPP library will be made available for further development to the open source community.

[1] http://www.jabber.org (Accessed September 18, 2005)
[2] RFC 3920 http://www.xmpp.org/specs/rfc3920.html (Accessed September 18, 2005)

GOALS

The goals of this project are:

RFCs
RFC 3920: XMPP Core: Defines core features like XML Streams, XML Stanzas, SASL and TLS.
RFC 3921: XMPP IM: Defines XMPP extensions for basic instant messaging and presence.

1. Create an open source .NET library implementing basic instant messaging services as described in the XMPP (version 1.0) RFC 3920 and 3921. This library will be called openXMPP and implemented using C#. Similar libraries (agsXMPP[3] and Jabber-Net[4]) exist; however they do not offer a clean design and deviate from the RFC specifications in critical areas. These two libraries will however serve as a good reference point. openXMPP will meet the following criteria:
    a. Strict compliance with RFC standards,
    b. Full support for basic XMPP functionality with optional functionality implemented as much as time allows,
    c. A modular design to support multiple platforms (e.g. Windows Mobile),
    d. Clean, highly-structured and well-documented design,

2. Using openXMPP develop a console application to demonstrate XMPP capabilities. For example, connections to Google Talk[5] should be possible.

3. Using openXMPP develop a desktop application that enables familiar instant messaging functions such as sign-in, maintaining roster and chats with friends. The desktop client will also permit new account registration.

DESIGN

The openXMPP library has been designed to make it easy to understand and reuse. openXMPP implements functions on top the transport layer in the protocol stack (Figure 2).

The XMPP protocol makes use of both synchronous and asynchronous communications over the same network connection. We achieved this functionality with pipelined stages which can block the calling thread if necessary. To make openXMPP more versatile, system-specific components (such as the network interface) are implemented abstractly and can be dynamically loaded by the implementing application. XMPP protocol elements (*stanzas* or *stream elements*) are generated and parsed by factory classes and sent and received through a stream class. For example, the following statements send a chat stanza:

[3] http://www.ag-software.de/index.php?option=content&task=view&id=72&Itemid=103 (Accessed September 18, 2005)
[4] http://www.jabberstudio.org/projects/jabber-net/project/view.php (Accessed September 18, 2005)
[5] Google Talk http://www.google.com/talk/developer.html (Accessed September 18, 2005)
[9] RFC 3920: XMPP Core. http://www.xmpp.org/specs/rfc3920.html. (Accessed 24 October 2005.)

```
Session session = new Session(tcpClient);
...
session.SendStanza(StanzaFactory.GetChatStanza(
“juliet@example.com/balcony”, “romeo@example.com”, “en”,
“Wherefore art thou, Romeo?”, “Subject”, “0abc020b02bb”);
```

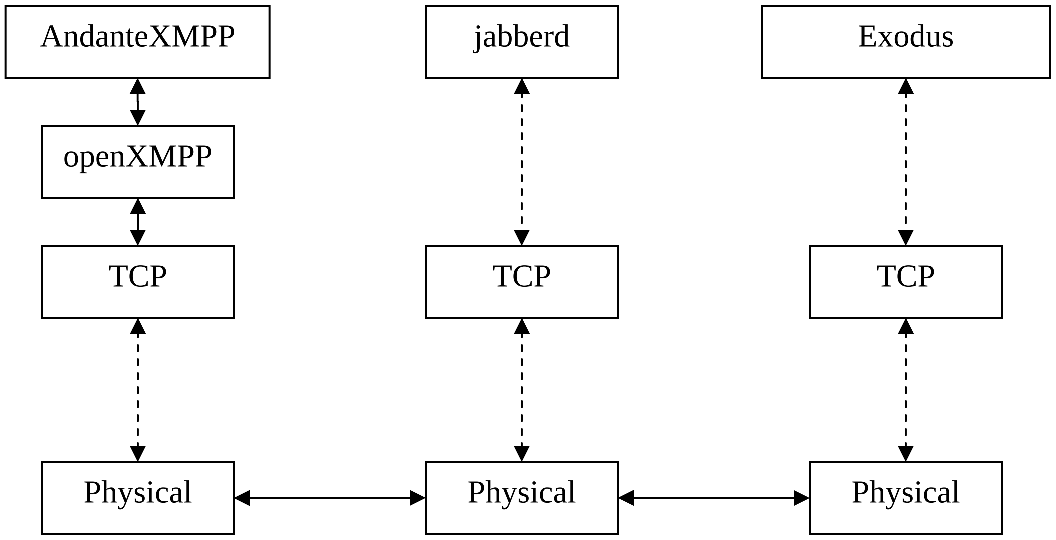


Figure 2: openXMPP and clients relative to the protocol stack and the server and other clients

The openXMPP library is 3601 lines of code containing 1379 statements in 19 classes. Our entire project (openXMPP, two network modules, a console application and graphical application) is 5825 lines of code containing 2304 statements and 29 classes.

Session Establishment

The session establishment process is described in Figure 3. The process begins with a client in the ‘Offline’ state. At this point an unsecured TCP connection is initiated with the host. If this is successful the ‘Connected’ state is reached. If the host requires TLS, the client transitions to the ‘Starting TLS’ state. If TLS is successfully established, the client transitions back to ’Connected’. If TLS fails, the client proceeds to ‘Disconnecting’ and closes the TCP connection and then goes ‘Offline’. When the server initiates SASL authentication, the client transitions to the ‘Starting SASL’ state. Once the host authenticates the user, the ‘Stating Session’ state is reached where the server/client may bind a resource and must send a “start session” tag. Upon completion the ‘Logged In’ state is reached and messages can now be exchanged. If a user wishes to disconnect, the ‘Disconnecting’ state is entered, closing the TCP connection and going ‘Offline.’

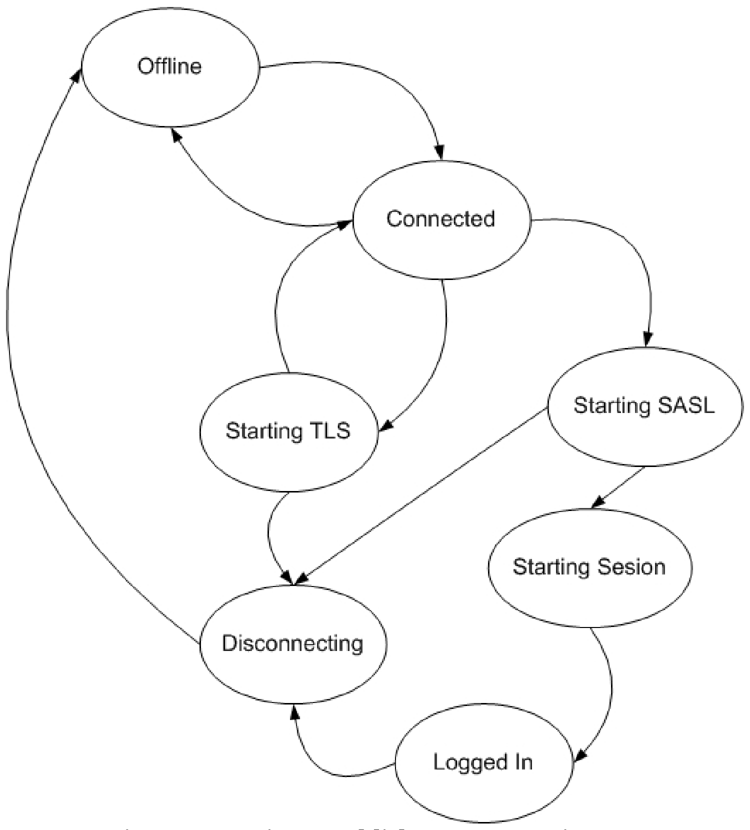


Figure 3: Session Establishment State Diagram

Synchronous vs. Asynchronous communication

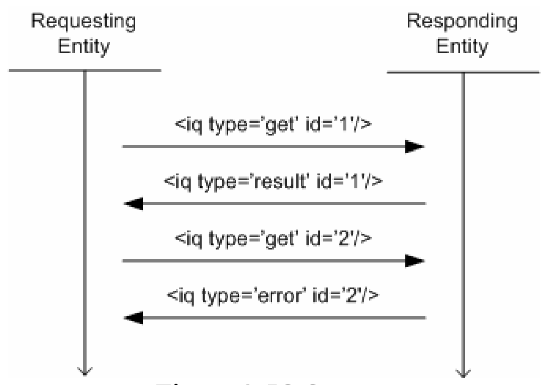


Figure 4: IQ Stanzas

The XMPP protocol is asynchronous with two exceptions: opening a session and performing information queries with IQ stanzas. In the case of opening the session, the client program must wait until the session is established and appropriate resources bound before general communication can be performed. IQ stanzas are sent as request/response with each query uniquely identified by a client-generated ID string (Figure 4). To implement this, we associated a System.Threading.ManualResetEvent and a millisecond timeout with synchronous operations. When a blocking method is called, the calling thread is blocked with WaitOne() until a condition is satisfied or a timeout occurs. Stanzas intended for the blocked thread are queued until the thread is unblocked. Stanzas associated with the openXMPP.Session object are still sent and received as necessary. Figure 5 diagrams a call to the blocking Open() method. Figure 6 diagrams a call to the non-blocking SendStanza() method.

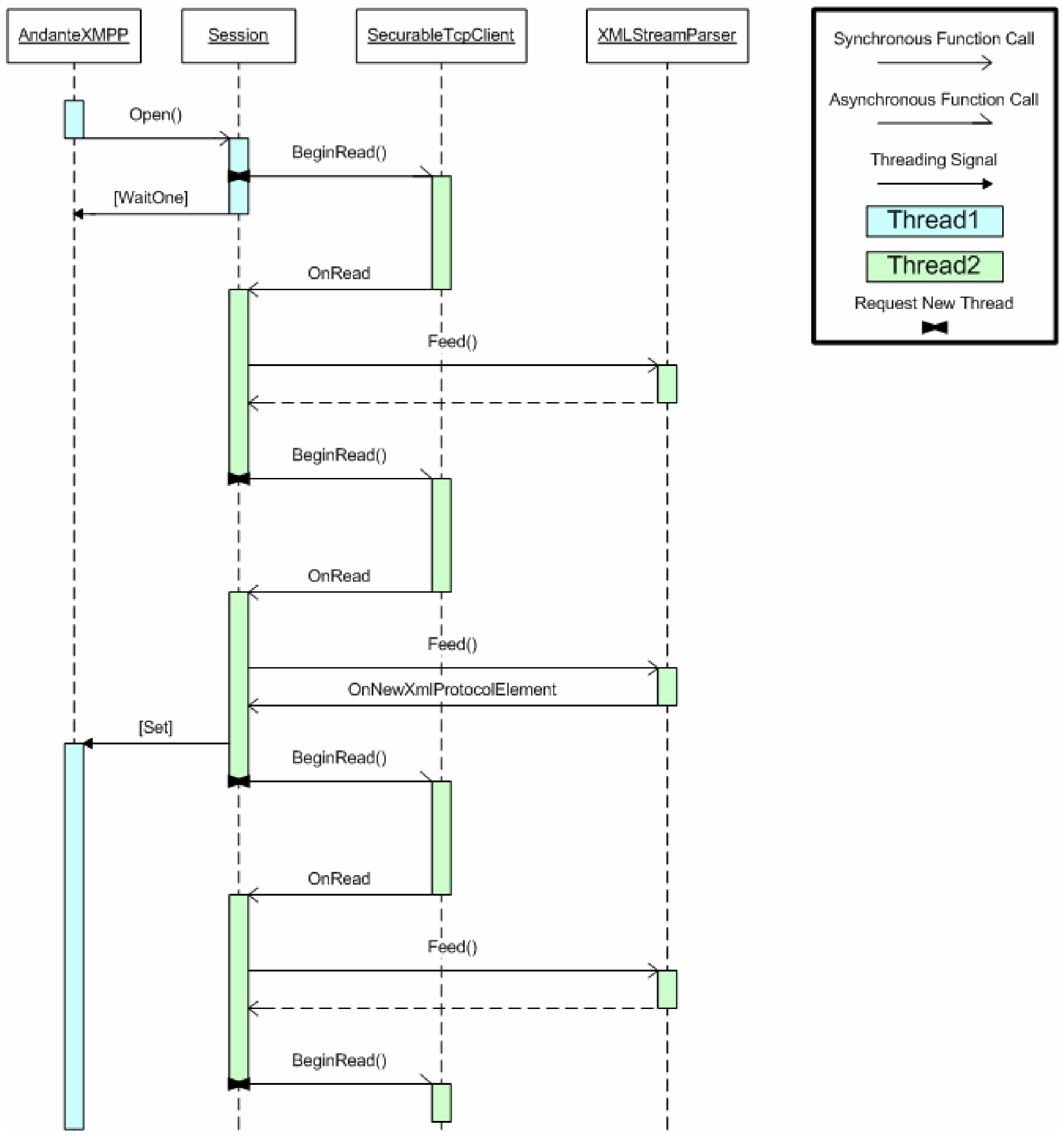


Figure 5: Blocking Method Call

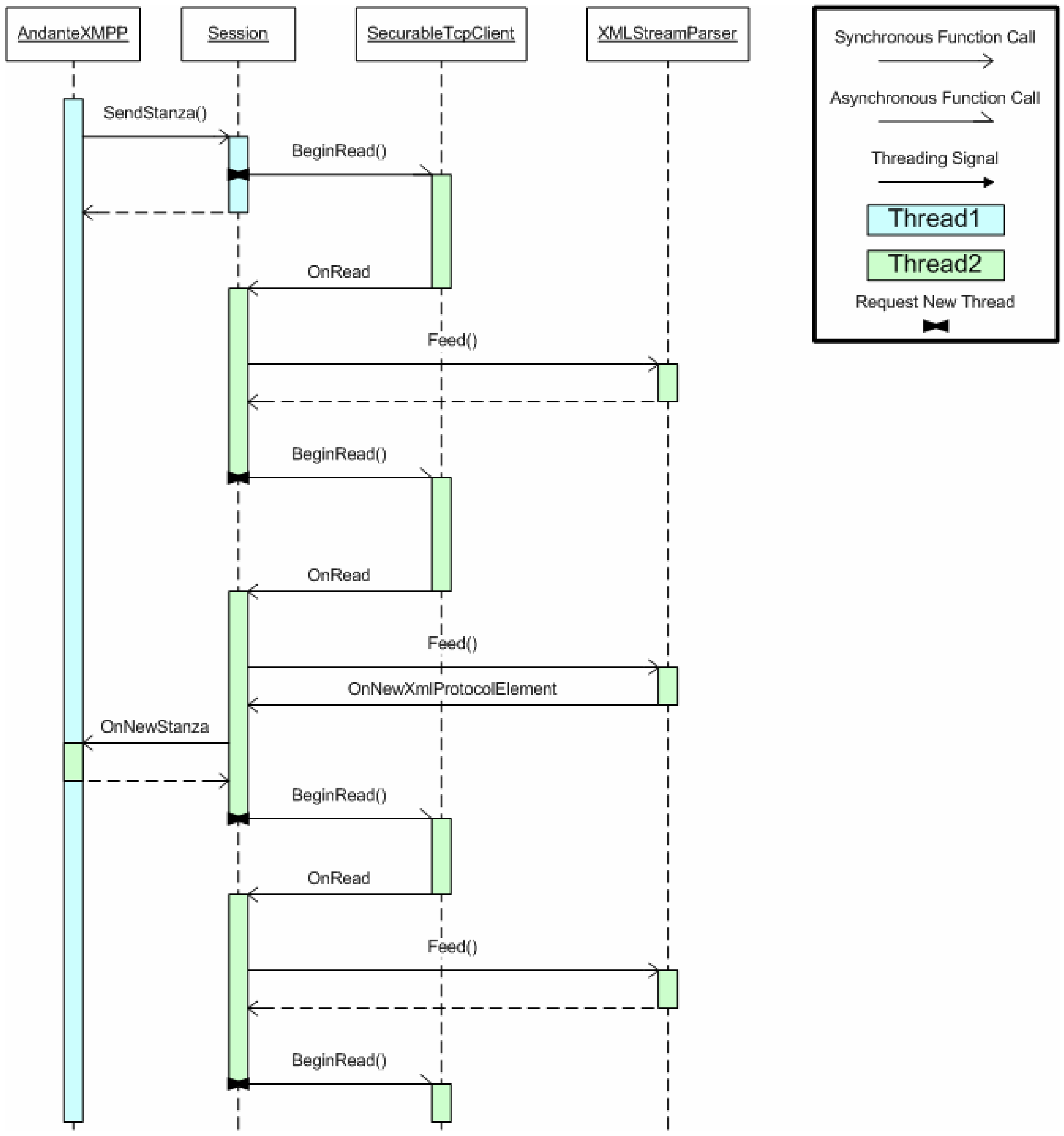


Figure 6: Non-blocking Method Call

Session Construction

All communication with an XMPP server is done through *sessions* which are authenticated *streams*. Session construction may involve establishing TLS security, negotiating an authentication mechanism, creating a new user account, and binding resources. Stanzas may not be sent until a session is established.

1. Open connection
   a. Instantiate `openXMPP.Stream` and call `Open` to initialize TLS and SASL security mechanisms and perform user authentication.

   Example stream[9] (C is client, S is server):

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

S: <?xml version='1.0'?>
   <stream:stream
     from='example.com'
     id='someid'
     xmlns='jabber:client'
     xmlns:stream='http://etherx.jabber.org/streams'
     version='1.0'>

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

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

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

   ... Client and server negotiate TLS and reopen stream ...

```
S: <stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>

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

   ... Authentication is performed via SASL DIGEST-MD5 ...

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

... Stanzas are sent ...

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

2. Broadcast presence
    a. Call `openXMPP.StanzaFactory.GetPresenceStanza` with appropriate arguments to generate a new presence stanza.
    b. Call `Stream.SendStanza` to send this stanza.
3. Retrieve your roster
    a. Call `openXMPP.StanzaFactory.GetRosterStanza` with appropriate arguments to generate a new roster stanza instructing the server to return the roster in an I/Q query.
    b. Call `Stream.SendStanza` to send this stanza.
    c. Receive an `IQStanza` from `openXMPP.Stream` containing a query element populated with the roster members as item elements.

Chat/Messaging Sequence Description

1. Send Message
    a. Get message body and recipient from user
    b. Call `openXMPP.StanzaFactory.GetChatMessageStanza` to generate a new message stanza.
    c. Call `Stream.SendStanza` to send this stanza.

    Example stanzas[10]:

```
<message
    to='romeo@example.net'
    from='juliet@example.com/balcony'
    type='chat'
    xml:lang='en'>
  <body>Wherefore art thou, Romeo?</body>
</message>

<message
    to='romeo@example.net/orchard'
    from='juliet@example.com/balcony'
    type='chat'
    xml:lang='en'>
  <body>Art thou not Romeo, and a Montague?</body>
  <thread>e0ffe42b28561960c6b12b944a092794b9683a38</thread>
</message>
```

2. Receive a Message
    a. Parse received XML into new `openXMPP.Stanza` object and raise an `openXMPP.Stream.OnReceive` event.
    b. Call `StanzaFactory.GetStanza` to determine type of stanza and generate appropriate class.
    c. Return stanza instance to client for handling

[10] RFC 3921: XMPP Messaging. http://www.xmpp.org/specs/rfc3921.html. (Accessed 23 October 2005.)

Roster/Subscription Management Sequence Description

1. Add a subscription
   a. Call `openXMPP.StanzaFactory.GetRosterStanza` to get a new stanza requesting a subscription.
   b. Call `Stream.Send` to send this stanza.

   Example stanza
   ```
   <iq from='juliet@example.com/balcony'
       type='set'
       id='roster_2'>
     <query xmlns='jabber:iq:roster'>
       <item jid='nurse@example.com'
             name='Nurse'>
         <group>Servants</group>
       </item>
     </query>
   </iq>
   ```
2. Delete a subscription
   a. Call `openXMPP.StanzaFactory.GetDeleteFriendStanza` to get a new stanza revoking a subscription.
   b. Call `Stream.SendStanza` to send this stanza.
3. Authorizing/Denying a requested subscription
   a. Call `openXMPP.StanzaFactory.GetReplySubscriptionRequestStanza` to get a new stanza authorizing or denying the subscription.
   b. Call `Stream.Send` to send this stanza.

   Example stanza:
   ```
   <presence to='contact@example.org' type='subscribe'/>
   ```

Setting Presence Information

1. Call `openXMPP.StanzaFactory.GetPresenceStanza` with appropriate arguments to generate a new stanza for the desired presence status.
2. Call `Stream.Send` to send this stanza.

Example stanza
```
<presence xml:lang='en'>
  <show>dnd</show>
  <status>Wooing Juliet</status>
  <status xml:lang='cz'>Ja dvo&#x0159;&#x00ED;m Juliet</status>
</presence>
```

Log-off Sequence Description

Call `openXMPP.Stream.Close` to send `</stream:stream>` and close the connection.

DELIVERABLES

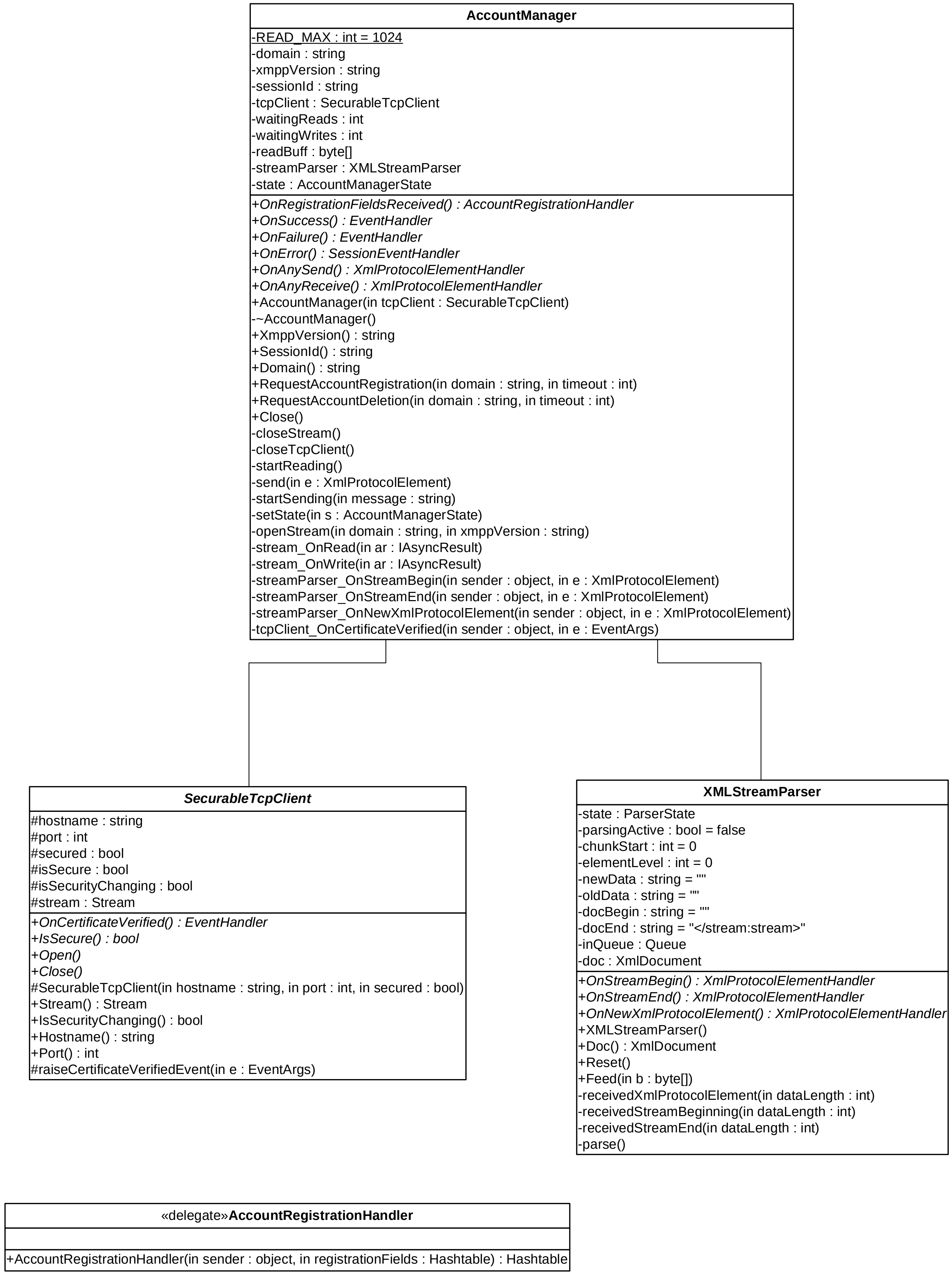
1. openXMPP library: All functionality required by RFC 3920 and RFC 3920 has been implemented, as well as several optional features. openXMPP provides the following (elements marked with a * are specified optional in RFC):
    a. XML streams
    b. XML stanzas
    c. TLS stream encryption
    d. SASL authentication
    e. Resource binding
    f. Roster and subscription management
    g. Internationalization (*)
    h. Conversation threads (*)
    i. Directed presence information (*)
    j. New account registration (*)
2. AndanteXMPP: A windows desktop client has been implemented. Some features are:
    a. Fully-functional chat client providing familiar chat capabilities such as message sending and receiving and presence management
    b. Interoperability with RFC compliant XMPP servers (GoogleTalk)
    c. Graphical roster and subscription management
    d. Register new user accounts
3. Console Client: A raw-XML chat program has been completed which allows the user to send and receive raw stanzas and perform basic tasks. Commands currently supported by the console client:
    a. roster: shows roster
    b. add: add a new friend. add <friend@example.com>, <Name> [, <group>]
    c. delete: delete a friend. delete friend@example.com
    d. chat: send a message. chat <friend@example.com>, <body> [, <body language>]
    e. chats: send a message with a subject. chats <friend@example.com>, <subject>, <body> [, <subject language>][, <body language]
    f. presence: broadcasts presence. presence <show> [, <status>, <status language>]
    g. exit: close console
4. Library Documentation: In order to provide a clean and well-documented library we have maintained a complete set of UML class diagrams describing openXMPP. The diagrams describe the properties and methods of the 30 classes and their relationships. We have also kept through code documentation through C# XML Comments which could be used to generate a complete set of library documentation with a tool such as nDoc[16].

[16] NDoc Code Documentation Generator for .NET. http://ndoc.sourceforge.net/

FUTURE WORK

Several optional features of XMPP, such as subscription blocking, have not been implemented due to time constraints. openXMPP's highly-structured design will allow interested parties to easily add this functionality.

In our first statement of project deliverables we listed a PocketPC chat client called PocketXMPP. Unfortunately, the .NET Compact Framework lacks critical namespaces and classes required to provide basic RFC functionality. Specifically, the System.Reflection namespace, required for effective SASL authentication, is nonexistent in both .NET CF 1.1 and .NET CF 2.0. The System.Security namespace, required for TLS security, is only partially implemented. Fortunately, the design of openXMPP permits application-specific modules to be used in place of modules requiring these namespaces. The AgsXMPP library has demonstrated that SASL can be successfully implemented on .NET CF, but a TLS-secured network stream object has yet to be developed.

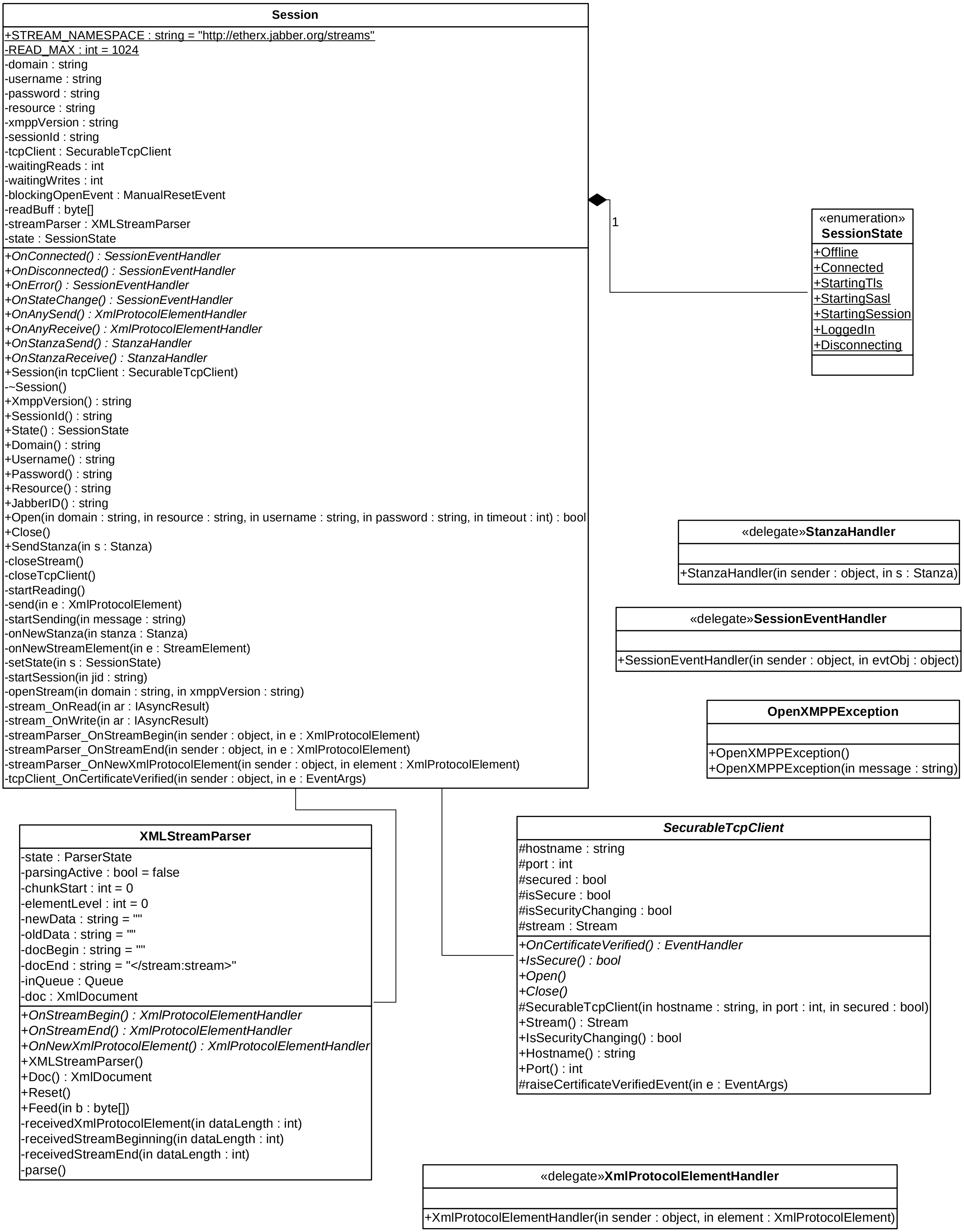


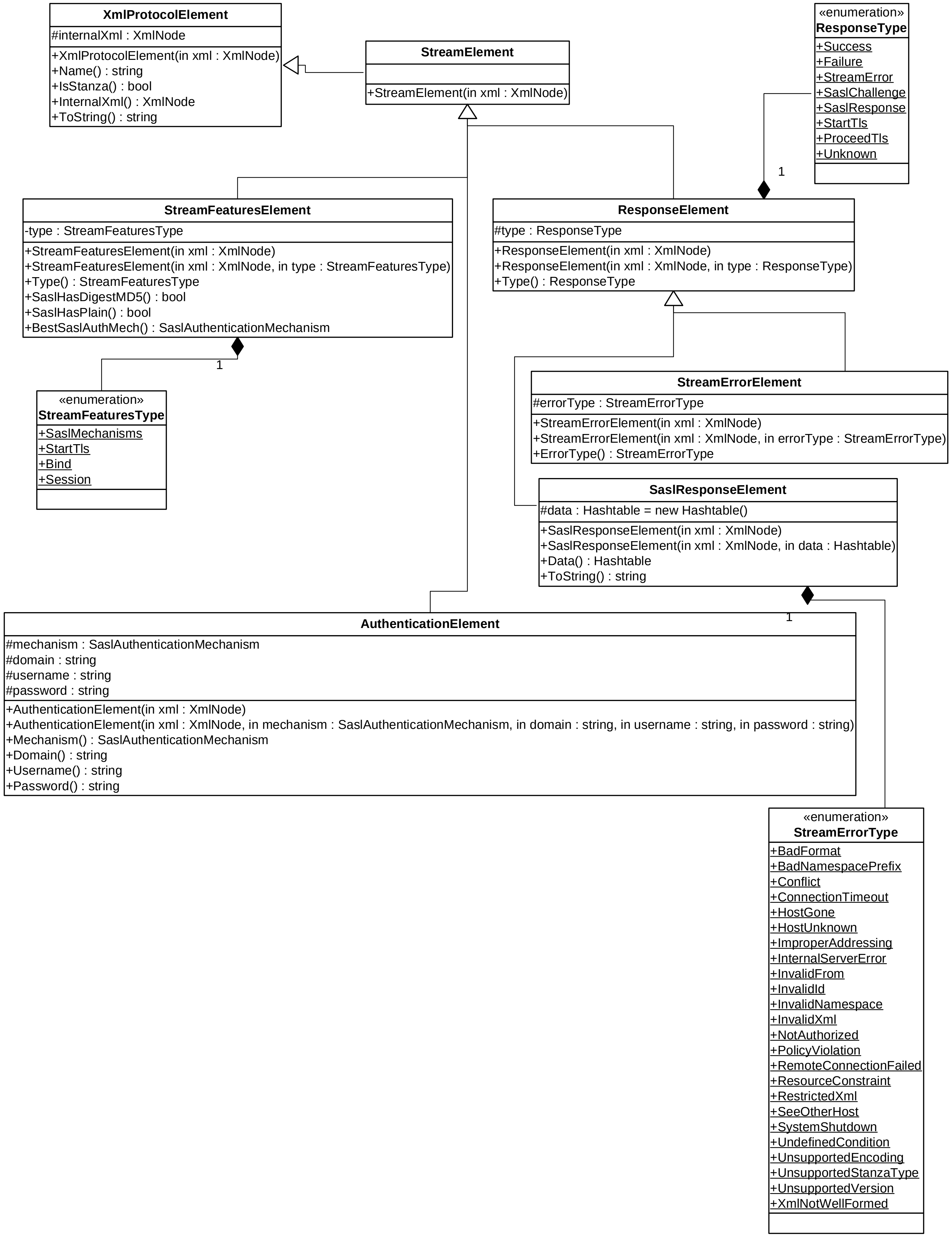
```
AccountManager
-READ_MAX : int = 1024
-domain : string
-xmppVersion : string
-sessionId : string
-tcpClient : SecurableTcpClient
-waitingReads : int
-waitingWrites : int
-readBuff : byte[]
-streamParser : XMLStreamParser
-state : AccountManagerState
+OnRegistrationFieldsReceived() : AccountRegistrationHandler
+OnSuccess() : EventHandler
+OnFailure() : EventHandler
+OnError() : SessionEventHandler
+OnAnySend() : XmlProtocolElementHandler
+OnAnyReceive() : XmlProtocolElementHandler
+AccountManager(in tcpClient : SecurableTcpClient)
-~AccountManager()
+XmppVersion() : string
+SessionId() : string
+Domain() : string
+RequestAccountRegistration(in domain : string, in timeout : int)
+RequestAccountDeletion(in domain : string, in timeout : int)
+Close()
-closeStream()
-closeTcpClient()
-startReading()
-send(in e : XmlProtocolElement)
-startSending(in message : string)
-setState(in s : AccountManagerState)
-openStream(in domain : string, in xmppVersion : string)
-stream_OnRead(in ar : IAsyncResult)
-stream_OnWrite(in ar : IAsyncResult)
-streamParser_OnStreamBegin(in sender : object, in e : XmlProtocolElement)
-streamParser_OnStreamEnd(in sender : object, in e : XmlProtocolElement)
-streamParser_OnNewXmlProtocolElement(in sender : object, in e : XmlProtocolElement)
-tcpClient_OnCertificateVerified(in sender : object, in e : EventArgs)
```

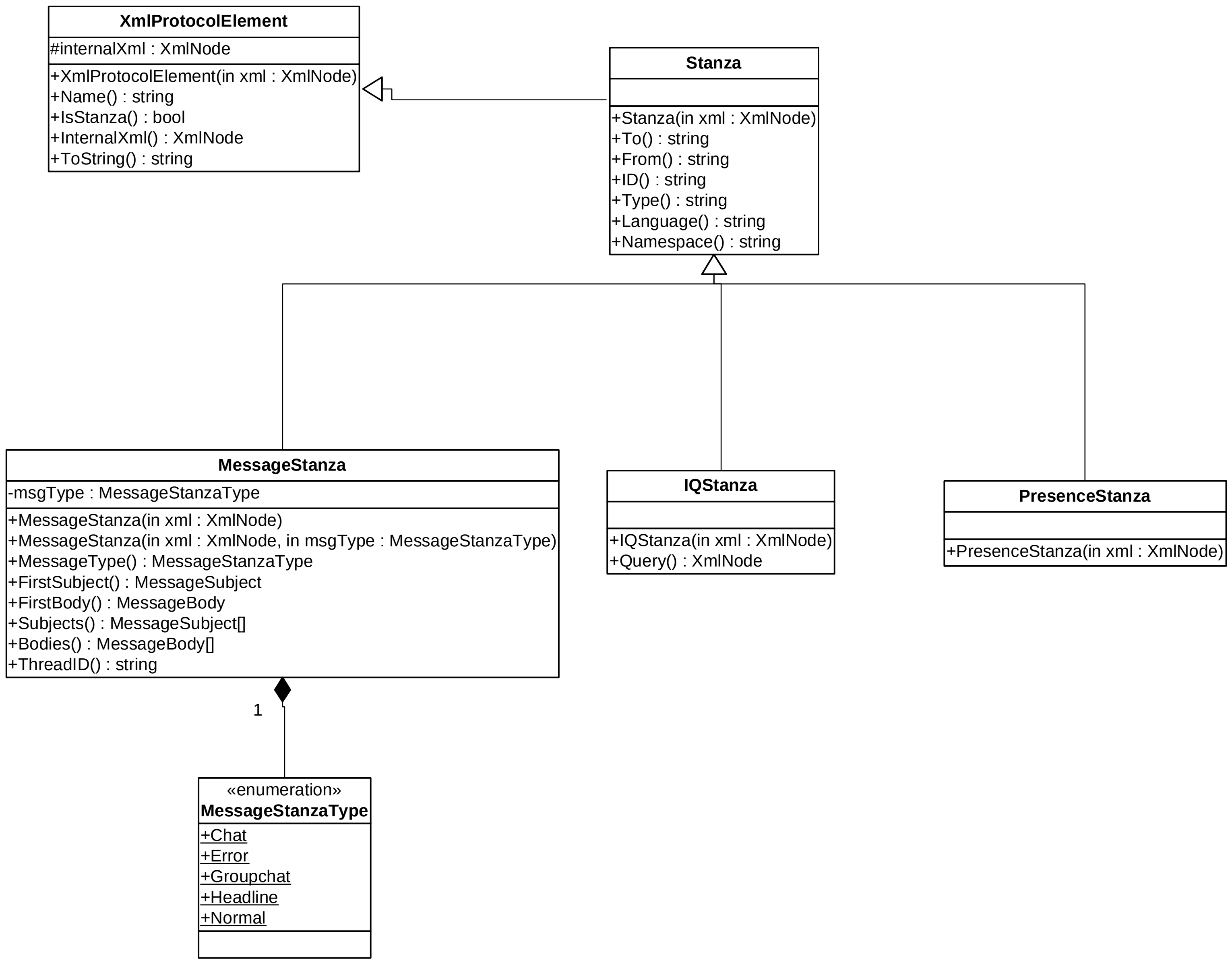
```
SecurableTcpClient
#hostname : string
#port : int
#secured : bool
#isSecure : bool
#isSecurityChanging : bool
#stream : Stream
+OnCertificateVerified() : EventHandler
+IsSecure() : bool
+Open()
+Close()
#SecurableTcpClient(in hostname : string, in port : int, in secured : bool)
+Stream() : Stream
+IsSecurityChanging() : bool
+Hostname() : string
+Port() : int
#raiseCertificateVerifiedEvent(in e : EventArgs)
```

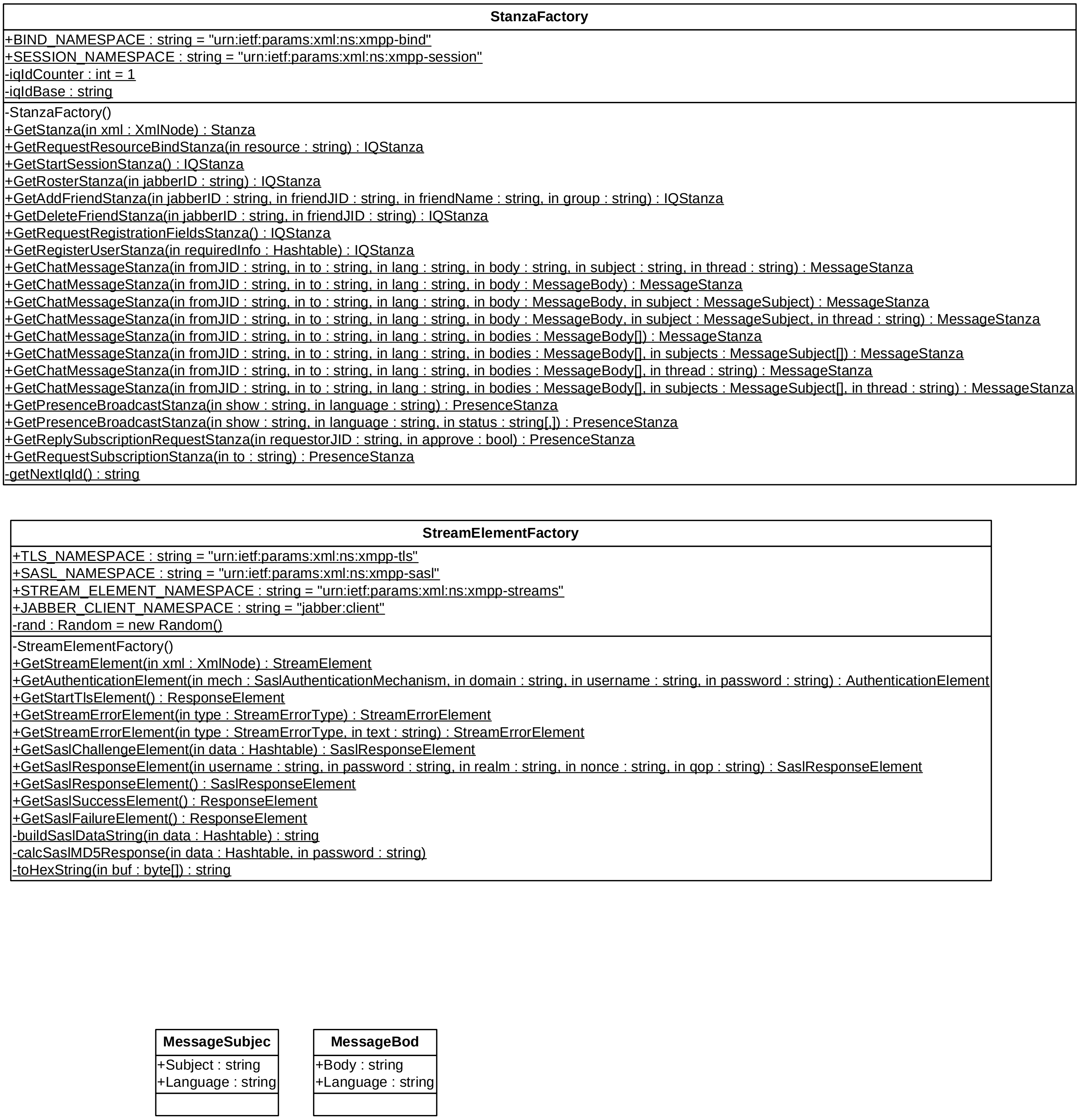
```
XMLStreamParser
-state : ParserState
-parsingActive : bool = false
-chunkStart : int = 0
-elementLevel : int = 0
-newData : string = ""
-oldData : string = ""
-docBegin : string = ""
-docEnd : string = "</stream:stream>"
-inQueue : Queue
-doc : XmlDocument
+OnStreamBegin() : XmlProtocolElementHandler
+OnStreamEnd() : XmlProtocolElementHandler
+OnNewXmlProtocolElement() : XmlProtocolElementHandler
+XMLStreamParser()
+Doc() : XmlDocument
+Reset()
+Feed(in b : byte[])
-receivedXmlProtocolElement(in dataLength : int)
-receivedStreamBeginning(in dataLength : int)
-receivedStreamEnd(in dataLength : int)
-parse()
```

```
«delegate»AccountRegistrationHandler
+AccountRegistrationHandler(in sender : object, in registrationFields : Hashtable) : Hashtable
```


Session
+STREAM_NAMESPACE : string = "http://etherx.jabber.org/streams"
-READ_MAX : int = 1024
-domain : string
-username : string
-password : string
-resource : string
-xmppVersion : string
-sessionId : string
-tcpClient : SecurableTcpClient
-waitingReads : int
-waitingWrites : int
-blockingOpenEvent : ManualResetEvent
-readBuff : byte[]
-streamParser : XMLStreamParser
-state : SessionState
+OnConnected() : SessionEventHandler
+OnDisconnected() : SessionEventHandler
+OnError() : SessionEventHandler
+OnStateChange() : SessionEventHandler
+OnAnySend() : XmlProtocolElementHandler
+OnAnyReceive() : XmlProtocolElementHandler
+OnStanzaSend() : StanzaHandler
+OnStanzaReceive() : StanzaHandler
+Session(in tcpClient : SecurableTcpClient)
-~Session()
+XmppVersion() : string
+SessionId() : string
+State() : SessionState
+Domain() : string
+Username() : string
+Password() : string
+Resource() : string
+JabberID() : string
+Open(in domain : string, in resource : string, in username : string, in password : string, in timeout : int) : bool
+Close()
+SendStanza(in s : Stanza)
-closeStream()
-closeTcpClient()
-startReading()
-send(in e : XmlProtocolElement)
-startSending(in message : string)
-onNewStanza(in stanza : Stanza)
-onNewStreamElement(in e : StreamElement)
-setState(in s : SessionState)
-startSession(in jid : string)
-openStream(in domain : string, in xmppVersion : string)
-stream_OnRead(in ar : IAsyncResult)
-stream_OnWrite(in ar : IAsyncResult)
-streamParser_OnStreamBegin(in sender : object, in e : XmlProtocolElement)
-streamParser_OnStreamEnd(in sender : object, in e : XmlProtocolElement)
-streamParser_OnNewXmlProtocolElement(in sender : object, in element : XmlProtocolElement)
-tcpClient_OnCertificateVerified(in sender : object, in e : EventArgs)
1
«enumeration»
SessionState
+Offline
+Connected
+StartingTls
+StartingSasl
+StartingSession
+LoggedIn
+Disconnecting
«delegate»StanzaHandler
+StanzaHandler(in sender : object, in s : Stanza)
«delegate»SessionEventHandler
+SessionEventHandler(in sender : object, in evtObj : object)
OpenXMPPException
+OpenXMPPException()
+OpenXMPPException(in message : string)
XMLStreamParser
-state : ParserState
-parsingActive : bool = false
-chunkStart : int = 0
-elementLevel : int = 0
-newData : string = ""
-oldData : string = ""
-docBegin : string = ""
-docEnd : string = "</stream:stream>"
-inQueue : Queue
-doc : XmlDocument
+OnStreamBegin() : XmlProtocolElementHandler
+OnStreamEnd() : XmlProtocolElementHandler
+OnNewXmlProtocolElement() : XmlProtocolElementHandler
+XMLStreamParser()
+Doc() : XmlDocument
+Reset()
+Feed(in b : byte[])
-receivedXmlProtocolElement(in dataLength : int)
-receivedStreamBeginning(in dataLength : int)
-receivedStreamEnd(in dataLength : int)
-parse()
SecurableTcpClient
#hostname : string
#port : int
#secured : bool
#isSecure : bool
#isSecurityChanging : bool
#stream : Stream
+OnCertificateVerified() : EventHandler
+IsSecure() : bool
+Open()
+Close()
#SecurableTcpClient(in hostname : string, in port : int, in secured : bool)
+Stream() : Stream
+IsSecurityChanging() : bool
+Hostname() : string
+Port() : int
#raiseCertificateVerifiedEvent(in e : EventArgs)
«delegate»XmlProtocolElementHandler
+XmlProtocolElementHandler(in sender : object, in element : XmlProtocolElement)


XmlProtocolElement
#internalXml : XmlNode
+XmlProtocolElement(in xml : XmlNode)
+Name() : string
+IsStanza() : bool
+InternalXml() : XmlNode
+ToString() : string
StreamElement
+StreamElement(in xml : XmlNode)
«enumeration»
ResponseType
+Success
+Failure
+StreamError
+SaslChallenge
+SaslResponse
+StartTls
+ProceedTls
+Unknown
1
StreamFeaturesElement
-type : StreamFeaturesType
+StreamFeaturesElement(in xml : XmlNode)
+StreamFeaturesElement(in xml : XmlNode, in type : StreamFeaturesType)
+Type() : StreamFeaturesType
+SaslHasDigestMD5() : bool
+SaslHasPlain() : bool
+BestSaslAuthMech() : SaslAuthenticationMechanism
ResponseElement
#type : ResponseType
+ResponseElement(in xml : XmlNode)
+ResponseElement(in xml : XmlNode, in type : ResponseType)
+Type() : ResponseType
1
«enumeration»
StreamFeaturesType
+SaslMechanisms
+StartTls
+Bind
+Session
StreamErrorElement
#errorType : StreamErrorType
+StreamErrorElement(in xml : XmlNode)
+StreamErrorElement(in xml : XmlNode, in errorType : StreamErrorType)
+ErrorType() : StreamErrorType
SaslResponseElement
#data : Hashtable = new Hashtable()
+SaslResponseElement(in xml : XmlNode)
+SaslResponseElement(in xml : XmlNode, in data : Hashtable)
+Data() : Hashtable
+ToString() : string
1
AuthenticationElement
#mechanism : SaslAuthenticationMechanism
#domain : string
#username : string
#password : string
+AuthenticationElement(in xml : XmlNode)
+AuthenticationElement(in xml : XmlNode, in mechanism : SaslAuthenticationMechanism, in domain : string, in username : string, in password : string)
+Mechanism() : SaslAuthenticationMechanism
+Domain() : string
+Username() : string
+Password() : string
«enumeration»
StreamErrorType
+BadFormat
+BadNamespacePrefix
+Conflict
+ConnectionTimeout
+HostGone
+HostUnknown
+ImproperAddressing
+InternalServerError
+InvalidFrom
+InvalidId
+InvalidNamespace
+InvalidXml
+NotAuthorized
+PolicyViolation
+RemoteConnectionFailed
+ResourceConstraint
+RestrictedXml
+SeeOtherHost
+SystemShutdown
+UndefinedCondition
+UnsupportedEncoding
+UnsupportedStanzaType
+UnsupportedVersion
+XmlNotWellFormed


XmlProtocolElement
#internalXml : XmlNode
+XmlProtocolElement(in xml : XmlNode)
+Name() : string
+IsStanza() : bool
+InternalXml() : XmlNode
+ToString() : string
Stanza
+Stanza(in xml : XmlNode)
+To() : string
+From() : string
+ID() : string
+Type() : string
+Language() : string
+Namespace() : string
MessageStanza
-msgType : MessageStanzaType
+MessageStanza(in xml : XmlNode)
+MessageStanza(in xml : XmlNode, in msgType : MessageStanzaType)
+MessageType() : MessageStanzaType
+FirstSubject() : MessageSubject
+FirstBody() : MessageBody
+Subjects() : MessageSubject[]
+Bodies() : MessageBody[]
+ThreadID() : string
IQStanza
+IQStanza(in xml : XmlNode)
+Query() : XmlNode
PresenceStanza
+PresenceStanza(in xml : XmlNode)
1
«enumeration»
MessageStanzaType
+Chat
+Error
+Groupchat
+Headline
+Normal



StanzaFactory

```
+BIND_NAMESPACE : string = "urn:ietf:params:xml:ns:xmpp-bind"
+SESSION_NAMESPACE : string = "urn:ietf:params:xml:ns:xmpp-session"
-iqIdCounter : int = 1
-iqIdBase : string
-----
-StanzaFactory()
+GetStanza(in xml : XmlNode) : Stanza
+GetRequestResourceBindStanza(in resource : string) : IQStanza
+GetStartSessionStanza() : IQStanza
+GetRosterStanza(in jabberID : string) : IQStanza
+GetAddFriendStanza(in jabberID : string, in friendJID : string, in friendName : string, in group : string) : IQStanza
+GetDeleteFriendStanza(in jabberID : string, in friendJID : string) : IQStanza
+GetRequestRegistrationFieldsStanza() : IQStanza
+GetRegisterUserStanza(in requiredInfo : Hashtable) : IQStanza
+GetChatMessageStanza(in fromJID : string, in to : string, in lang : string, in body : string, in subject : string, in thread : string) : MessageStanza
+GetChatMessageStanza(in fromJID : string, in to : string, in lang : string, in body : MessageBody) : MessageStanza
+GetChatMessageStanza(in fromJID : string, in to : string, in lang : string, in body : MessageBody, in subject : MessageSubject) : MessageStanza
+GetChatMessageStanza(in fromJID : string, in to : string, in lang : string, in body : MessageBody, in subject : MessageSubject, in thread : string) : MessageStanza
+GetChatMessageStanza(in fromJID : string, in to : string, in lang : string, in bodies : MessageBody[]) : MessageStanza
+GetChatMessageStanza(in fromJID : string, in to : string, in lang : string, in bodies : MessageBody[], in subjects : MessageSubject[]) : MessageStanza
+GetChatMessageStanza(in fromJID : string, in to : string, in lang : string, in bodies : MessageBody[], in thread : string) : MessageStanza
+GetChatMessageStanza(in fromJID : string, in to : string, in lang : string, in bodies : MessageBody[], in subjects : MessageSubject[], in thread : string) : MessageStanza
+GetPresenceBroadcastStanza(in show : string, in language : string) : PresenceStanza
+GetPresenceBroadcastStanza(in show : string, in language : string, in status : string[,]) : PresenceStanza
+GetReplySubscriptionRequestStanza(in requestorJID : string, in approve : bool) : PresenceStanza
+GetRequestSubscriptionStanza(in to : string) : PresenceStanza
-getNextIqId() : string
```

StreamElementFactory

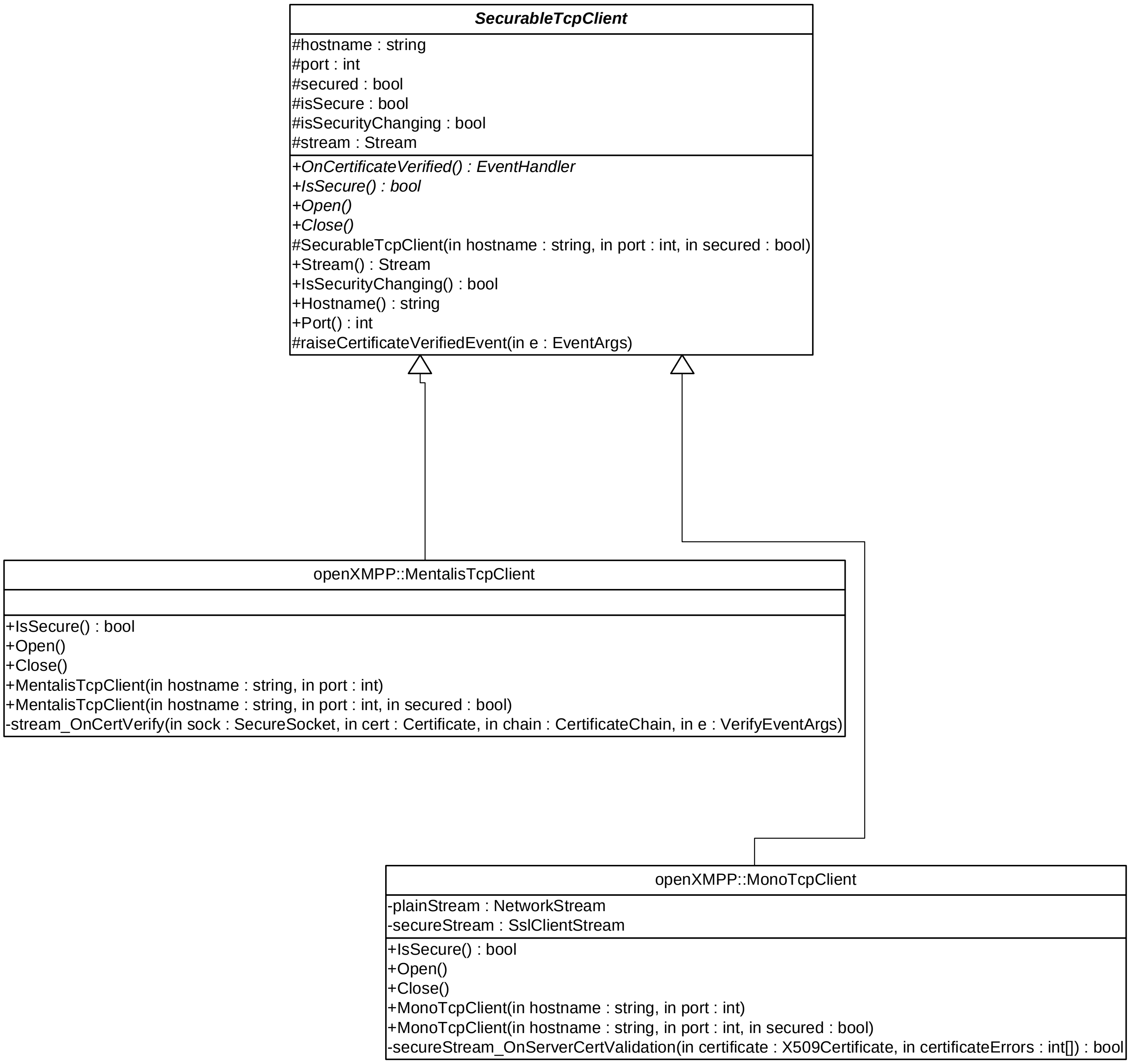
```
+TLS_NAMESPACE : string = "urn:ietf:params:xml:ns:xmpp-tls"
+SASL_NAMESPACE : string = "urn:ietf:params:xml:ns:xmpp-sasl"
+STREAM_ELEMENT_NAMESPACE : string = "urn:ietf:params:xml:ns:xmpp-streams"
+JABBER_CLIENT_NAMESPACE : string = "jabber:client"
-rand : Random = new Random()
-----
-StreamElementFactory()
+GetStreamElement(in xml : XmlNode) : StreamElement
+GetAuthenticationElement(in mech : SaslAuthenticationMechanism, in domain : string, in username : string, in password : string) : AuthenticationElement
+GetStartTlsElement() : ResponseElement
+GetStreamErrorElement(in type : StreamErrorType) : StreamErrorElement
+GetStreamErrorElement(in type : StreamErrorType, in text : string) : StreamErrorElement
+GetSaslChallengeElement(in data : Hashtable) : SaslResponseElement
+GetSaslResponseElement(in username : string, in password : string, in realm : string, in nonce : string, in qop : string) : SaslResponseElement
+GetSaslResponseElement() : SaslResponseElement
+GetSaslSuccessElement() : ResponseElement
+GetSaslFailureElement() : ResponseElement
-buildSaslDataString(in data : Hashtable) : string
-calcSaslMD5Response(in data : Hashtable, in password : string)
-toHexString(in buf : byte[]) : string
```

MessageSubjec

```
+Subject : string
+Language : string
```

MessageBod

```
+Body : string
+Language : string
```



SecurableTcpClient
#hostname : string
#port : int
#secured : bool
#isSecure : bool
#isSecurityChanging : bool
#stream : Stream
+OnCertificateVerified() : EventHandler
+IsSecure() : bool
+Open()
+Close()
#SecurableTcpClient(in hostname : string, in port : int, in secured : bool)
+Stream() : Stream
+IsSecurityChanging() : bool
+Hostname() : string
+Port() : int
#raiseCertificateVerifiedEvent(in e : EventArgs)
openXMPP::MentalisTcpClient
+IsSecure() : bool
+Open()
+Close()
+MentalisTcpClient(in hostname : string, in port : int)
+MentalisTcpClient(in hostname : string, in port : int, in secured : bool)
-stream_OnCertVerify(in sock : SecureSocket, in cert : Certificate, in chain : CertificateChain, in e : VerifyEventArgs)
openXMPP::MonoTcpClient
-plainStream : NetworkStream
-secureStream : SslClientStream
+IsSecure() : bool
+Open()
+Close()
+MonoTcpClient(in hostname : string, in port : int)
+MonoTcpClient(in hostname : string, in port : int, in secured : bool)
-secureStream_OnServerCertValidation(in certificate : X509Certificate, in certificateErrors : int[]) : bool