SXS Transport Adaptor Programming Guide

Overview

The Simple XML Socket Transport Adaptor (hereinafter SXS adaptor) is intended for communication with FIXEdge from third-part applications. The main goal of this adaptor is a quick and easy building of the network applications that communicate using XML messages in format of Simple XML Socket Protocol.

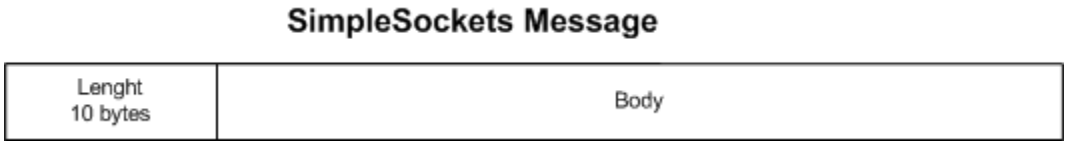
SXS Protocol

The Simple XML Socket Protocol (hereinafter SXS protocol) is intended for communication with the FIX Server from client applications. SXS is not high-performance, but it exposes additional reliability features.

SXS Message

Message in SXS protocol has the following format:

[Header [Package length (10 bytes)]] [message body]



The package length is calculated as header length plus body length. Each message begins with a field which indicates package length in the text form. Then XML tags indicate message type. The length value is represented as a string. The string must be expanded up to 10 characters and filled with '\0' (ASCII NUL character) - if entire message length is 125 bytes, the message header will be "125\0\0\0\0\0\0\0\0".

SXS protocol supports the following message types:

- LR - Logon Request message
- LA - Logon Acknowledgement message
- SM - Status message
- HQ - Heartbeat query message
- HR - Heartbeat response message
- CM - Command message
- SR - Shutdown Request message
- SA - Shutdown Acknowledgement message
- FM - FIX message

SXS protocol supports the following XML tags in SXS messages:

- T - message type. It is required for all messages. For values, refer to "Message types".
- SEQNUM - sequence number of a message. Integer format.
- NOTIFY - may be presented in "SM" messages for client notification (For example, server sends it before disconnecting). Tag value is the line with message.
- REJECT - is for Status messages (SM) rejecting an action (Logon with ClientID which is already used by someone else, invalid message format, unidentified tag etc.) Tag value is the line with the reject reason.
- VERSION - SXS protocol version. It is required for "LR" message.
- CLIENTID - client identifier. It is required for "LR" message.
- HEARTBEAT - turning on / off of HEARTBEAT message sent by server. It is optional. It can be presented in messages of the "LR" and "CM" types. Valid values: ON / OFF.

- FIX - contains FIX message. It is required for "FM" messages. Can be presented in "LA". One message can contain only one entry of this tag. The tag must finish the message. The tag contents may be a sequence of any characters including zeros or control characters.
- MSG - contains SXS message. It is optional. It can be presented in "SM" messages containing REJECT tag. One message must contain only one entry of this tag finishing the message. The tag contents may be a sequence of any characters including zeros or control characters.

Note: Tags are case-sensitive and all characters must be capitalized.

Message type\Tag	T	SEQNUM	NOTIFY	REJECT	VERSION	CLIENTID	HEARTBEAT	FIX	MSG
LR	Required	Required			Required	Required	Optional		
LA	Required	Required						Optional	
SM	Required	Required	Optional	Optional					Optional
HQ	Required	Required							
HR	Required	Required							
CM	Required	Required					Optional		
SR	Required	Required	Optional						
SA	Required	Required							
FM	Required	Required						Required	

Restrictions:

1. One message cannot contain several tags with the same name.
2. Tag's text field cannot contain other tags.
3. MSG and FIX tags, if present in a message, must finish the message. A message cannot concurrently contain both these tags (MSG and FIX).

Message examples:

- Heartbeat messages

  19..... <T>HQ</T>

  and response on it:

  19..... <T>HR</T>
- LogOn Request message

  97.....<T>LR</T><VERSION>1.0</VERSION><CLIENTID>CLIENT-001</CLIENTID><HEARTBEAT>OFF</HEARTBEAT>

  reply from the server, if username is authenticated and other parameters are valid:

  xxxx <T>LA</T><FIX>LastRecivedMessage</FIX>

  otherwise:

  xxxx <T>SM</T><REJECT>Wrong message type.</REJECT><MSG>XXXXXXX</MSG>

  where "xxxx" is the length of the whole message and "XXXXXXX" - is original client's message.
- Shutdown Request message

  19..... <T>SR</T>
- Command Message

  45..... <T>CM</T><HEARTBEAT>OFF</HEARTBEAT>
- FIX message

  xxxx <T>FM</T><FIX>LastReceivedMessage</FIX>

  where "xxxx" is the length of the whole message.

SXS Session

Each Client must implement support of Heartbeat algorithm. Client must process messages of type **HQ** and generate reply messages **HR**. By means of this messages SXS Server checks the Client presence on network. If server does not receive **HQ** response in expected time it disconnects Client forcedly.

⚠ **Note**: Server also supports process of messages with type **HQ**, and generates **HR** response. So Client may check Server's presence in network it any time.

Client Operation Scenario

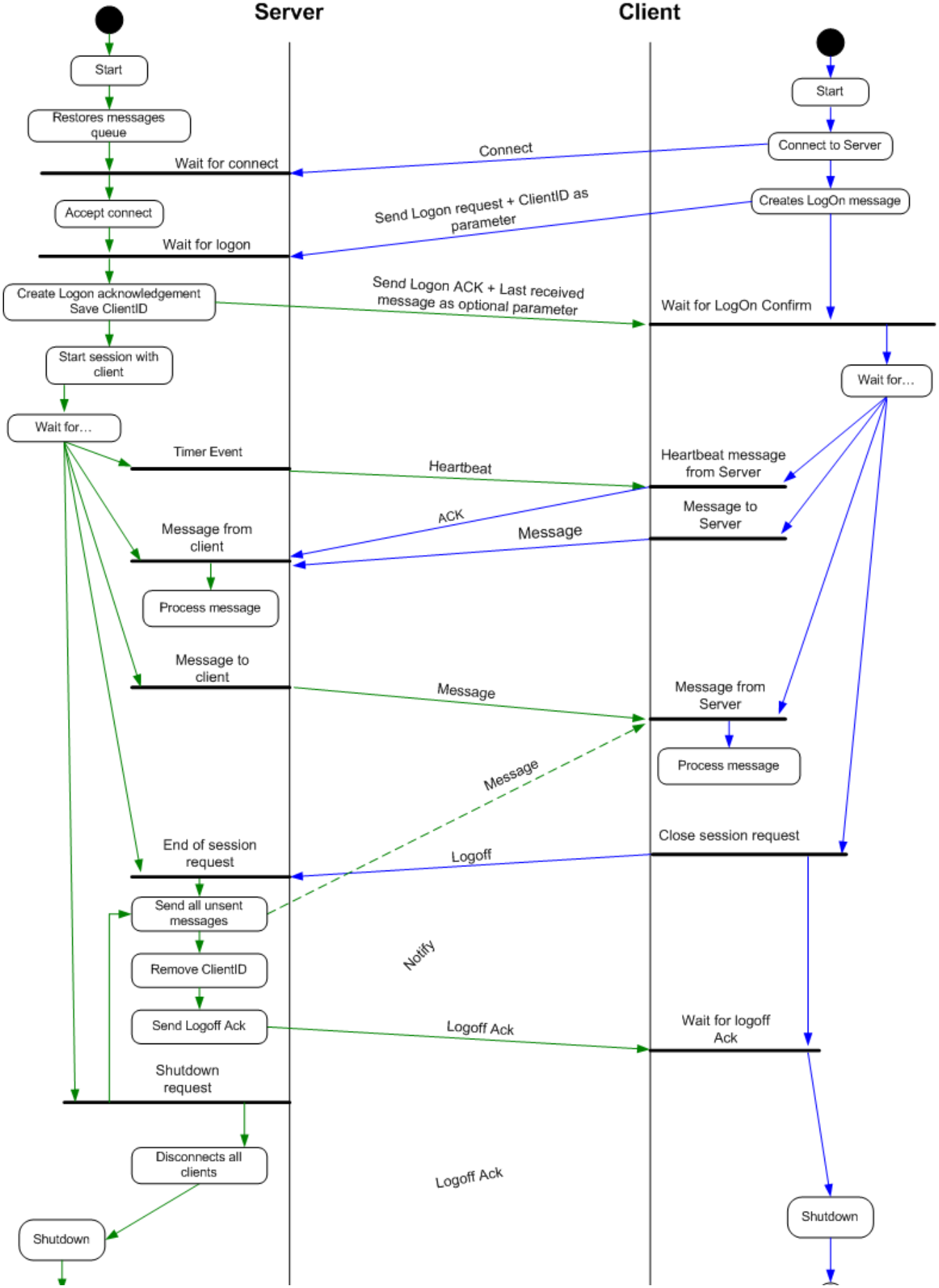
1. Establishing a session.
    a. Client connects to the server.
    b. Client sends Logon message, which indicates protocol version, its unique identifier (which it can get from server administrator) and, optionally, whether Heartbeat message sending is initially turned on (Later on this setting can be changed by sending Command message (CM) indicating new values for Heartbeat sending).
    c. Server checks that Client with the same ID does not have connection at the same time. If Client with the same ID already have been connected, Server generates reject message and sends it to Client.
    d. Server verifies the uniqueness of the identifier. If this identifier is already used, then client's logon is rejected and a message with notification and resend reason is sent to the client. On the second logon with the same identifier, the client also receives the last FIX message received by the server from this client in previous sessions.
    e. If the client had sent a message (or several messages) before login request (LR), then, in reply, it gets a Status message (SM) consisting of the line "Wrong message type. The type of first message in session must be LR" inside the REJECT tag and its own sent message inside the MSG tag.
2. Receiving / sending messages
    a. Each message is sent via this protocol (FIX-message, command, changing current setting of the server, various notifications, logon and logoff requests, etc.) must be formed according to the rules specified in the section "Message Format".
3. Waiting mode
    a. Waiting for messages from the server
    b. If Heartbeat Request (HR) is received, echo-reply **must be** sent back. Otherwise Client will be disconnected forcedly after time interval that depends from HEARTBEAT option. If HEARTBEAT=ON, time-wait interval is equal to 3 x HeartbeatTime. If HEARTBEAT=OFF, time-wait interval is equal to TestRequest interval. Both intervals are configured via SimpleSockets.properties file and equal to 30 seconds by default.
    c. Other messages are handled at the client's discretion.
4. Shutting down client's workstation
    a. A message of the type "SR" - Shutdown Request - is sent to the server. The user can identify logoff reason in the < NOTIFY > optional tag.
    b. The server replies the message "SA" - Shutdown Acknowledgement - after receiving which the client closes the socket.

⚠ **Note**: By default, sending HEARTBEAT messages to the client is turned off on the server (because this mechanism is optional). It is turned on only on the client's request.

Server Operation Scenario

1. Starting up.
    a. After the start-up, the server switches to the waiting mode.
2. Receiving messages
    a. Message header indicating its length is read for the socket.
    b. The rest of the message is read from the socket.
    c. The message is processed and depending on its type actions are carried out:
        i. **Logon Request (LR)**
            1. Server is checked for the presence of ClientID protocol version in the message body. If either of these parameters is absent, then the server forms and sends a SM message containing REJECT tag with the reject reason and MSG tag with send request.
            2. Support of the protocol version is checked. If the version is not supported, then the server sends a SM message, containing REJECT tag with the reason "Unsupported Protocol Version" and MSG tag with send request.
            3. Received ClientID is checked for the presence in the global identifiers list. If it is not found, then SM message, containing REJECT tag with the reason "Unknown ClientID" and MSG tag with sent request, is returned to the client.
            4. The server checks whether there is session descriptor for this ClientID.
                a. In case of session's activity, SM message is returned. Message contains REJECT tag with the reason "ClientID is duplicated by another application" and MSG tag with original client's message. If the session is not active, a LA message containing logon acknowledgement and the last received FIX message (if the server received FIX messages during the previous session) is sent to the client. If Heartbeat mechanism is not active for existent session, Server sends the Test Request message (HR) and waits TestRequest time interval for Client's response. If Client doesn't respond during this interval, Server disconnects it forcedly.
                b. If the session descriptor is absent, then new session descriptor is created and LA message without parameters is sent to the client and the transport level notifies the server about the activity of the client with given ClientID.
            5. If HEARTBEAT tag is presented in the message, the server sets parameters of sending Heartbeat messages to the client according to the request.
        ii. **FIX message (FM)**
            1. Counter of sent Heartbeat messages is reset.
            2. FIX message (the text part of FIX tag) is selected and stored in the session descriptor as the last received one.
            3. FIX message is sent to FIXServer for further processing.
        iii. **Command message (CM)**
            1. Counter of sent Heartbeat messages is reset.
            2. The message is processed depending on the received data
3. Heartbeat message.

    a. On the event from timer, the server scans the list of connected clients and sends HEARTBEAT message to those with Heartbeat flag set, after which it increments the counter value. If for the moment of sending the message a client's counter has exceeded the maximum (3 messages), then this client is considered to go offline and session with this client is broken.
    b. On the receipt of any message from the client, the counter of sent messages is reset.
4. Sending messages
    a. **If a session with the client is established**
        i. messages are sent in the same manner as described in step 2 of "Client operation scenario".
    b. **If there is no connection with the client**
        i. If a message for sending to a client has been received but there is no session established with the client, then the transport level notifies the server about the fact that the client is unavailable at the moment.
5. Disconnecting
    a. Normal mode. The server sends Shut down acknowledgement (SA) to all connected clients, after which it closes clients' connections.
    b. Abnormal mode. For example, there is power cut. On the next start-up, it acts as in the normal mode.


Server
Client
Start
Restores messages queue
Wait for connect
Accept connect
Wait for logon
Create Logon acknowledgement Save ClientID
Start session with client
Wait for…
Timer Event
Message from client
Process message
Message to client
End of session request
Send all unsent messages
Remove ClientID
Send Logoff Ack
Shutdown request
Disconnects all clients
Shutdown
Start
Connect to Server
Creates LogOn message
Wait for LogOn Confirm
Wait for…
Heartbeat message from Server
Message to Server
Message from Server
Process message
Close session request
Wait for logoff Ack
Shutdown
Connect
Send Logon request + ClientID as parameter
Send Logon ACK + Last received message as optional parameter
Heartbeat
ACK
Message
Message
Message
Logoff
Notify
Logoff Ack
Logoff Ack

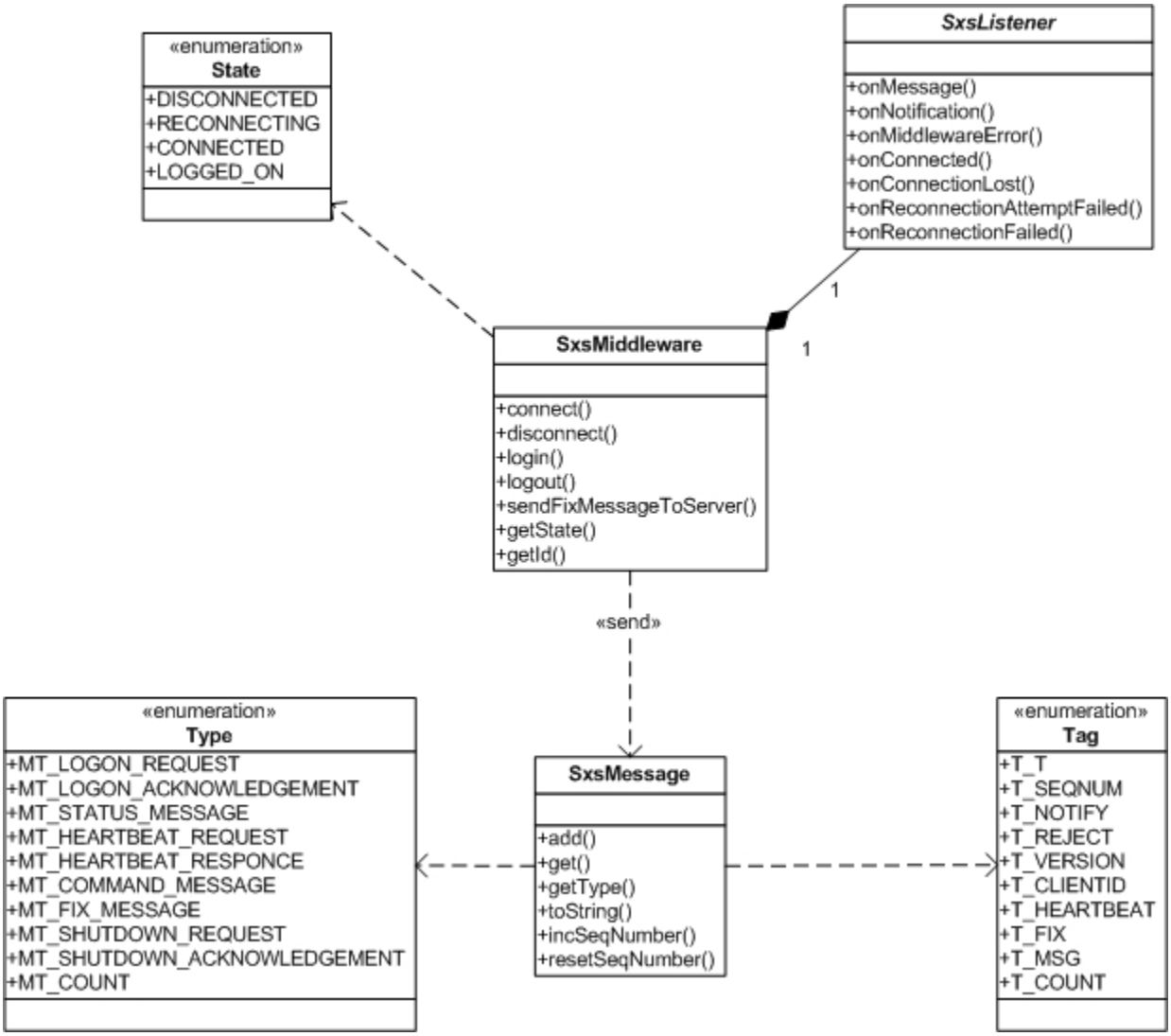
Middleware

To simplify creation of the client application B2BITS provides middleware for the following languages: Java - SXSLibraryJava, C/C++ - SxsLibraryCpp and platfrom .Net - SXSLibraryNet. Middleware encapsulates SXS message encoding, Heartbeat algorithm, etc.

SxsLibraryCpp

The SXSLibraryCpp consists from the following components:

- SxsMiddleware - the primary component of the SxsLibraryCpp. SxsMiddleware encapsulates the SXS session and simplifies the connection establishing and closing, SXS message sending.
- SxsMessage - encapsulates SXS message.
- SxsListener - event's subscriber that handles various SXS session's events. Implemented by the client application.



SxsListener methods are called in the following situations:

- onMessage - new SXS message with type FM was received
- onNotification - SXSLibrary notifies about changes during execution
- onMiddlewareError - error was fired during execution
- onConnected - SXSLibrary establishes session, SXS session may send and receive SXS messages
- onConnectionLost - SXSLibrary closes established connection

It's easy to implement SXS client (based on SXSLibraryCpp) in the following way:

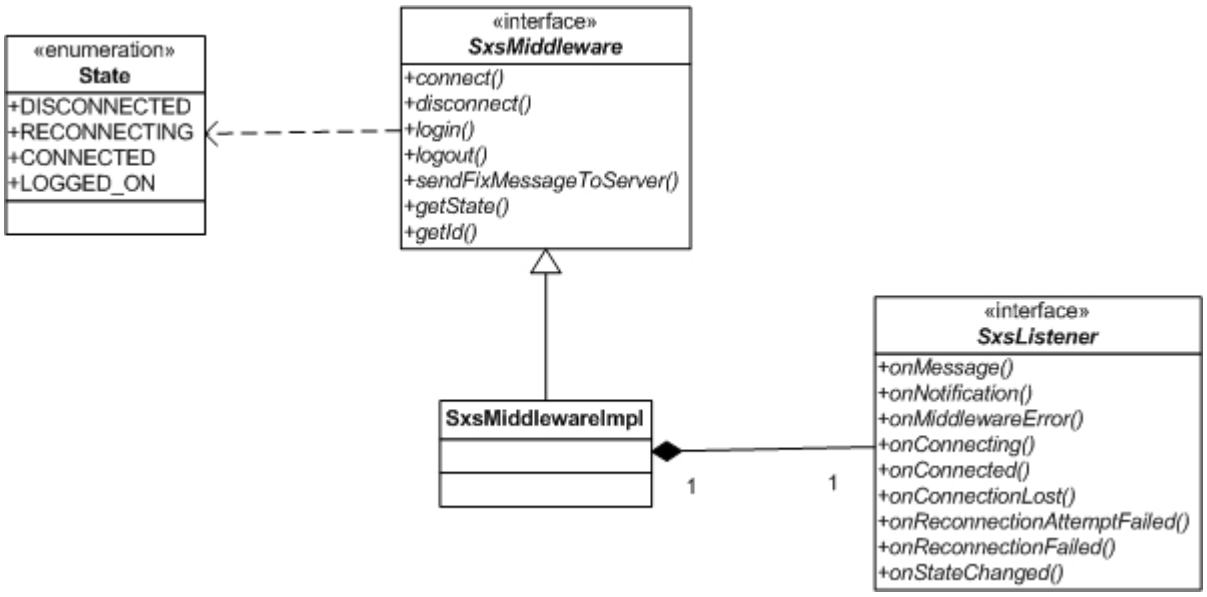
1. Implement SxsListener that will handle SXS session's events
2. Create SxsMiddleware object and call connect() method to bind SxsMiddleware with custom SxsListener and create connection.
3. Call SxsMiddleware::login() to establish SXS session

4. When session is established (SxsListener::OnConnected() was called) client application may send messages and proceed received
5. Call SxsMiddleware::logout() and SxsMiddleware::disconnect() to close established SXS session

.Net

The SXSLibraryNet consists from the following components:

- SxsMiddlewareImpl - the primary component of the SxsLibraryCpp, implements SxsMiddleware interface. SxsMiddlewareImpl encapsulates the SXS session and simplifies the connection establishing and closing, SXS message sending.
- SxsListener - event's subscriber that handles various SXS session's events. Implemented by the client application.



It's easy to implement SXS client (based on SXSLibraryNet) in the following way:

1. Implement SxsListener that will handles SXS session's events
2. Create SxsMiddlewareImpl object and call connect() method to bind SxsMiddleware with custom SxsListener and create connection.
3. Call SxsMiddleware::login() to establish SXS session
4. When session established (SxsListener::OnConnected() was called) client application may send messages and proceed received
5. Call SxsMiddleware::logout() and SxsMiddleware::disconnect() to close established SXS session

Client application sample

The sample SXS client application is represented below. It connects to the passed host:port, establishes connection, sends application message and closes connection.

C/C++ client

The sample SXS client application on C/C++ is represented below. It connects to the port 8129 on local machine, establishes connection, sends application messages and closes connection.

```
/**
* $Revision: 1.4.2.1 $
*
* (c) B2BITS 2006. B2BITS is an abbreviation of B2BITS corporation.
*
* This software is for the use of the paying client of B2BITS (which may be
* a corporation, business area, business unit or single use subject to
* licence terms) to whom it was delivered (the "Licensee")and no other party,
* and any use beyond this business area is contrary to the terms of the licence grant.
*
* The Licensee acknowledges and agrees that the Software and Documentation
* (the "Confidential Information") is confidential and proprietary to
* the Licensor and the Licensee hereby agrees to use the Confidential
* Information only as permitted by the full licence agreement betweeen
* the two parties, to maintain the confidentiality of the Confidential
* Information and not to disclose the confidential information, or any part
* thereof, to any other person, firm or corporation. The Licensee
* acknowledges that disclosure of the Confidential Information may give rise
* to an irreparable injury to the Licensor in-adequately compensable in
* damages. Accordingly the Licensor may seek (without the posting of any
```

```
* bond or other security) injunctive relief against the breach of the forgoing
* undertaking of confidentiality and non-disclosure, in addition to any other
* legal remedies which may be available, and the licensee consents to the
* obtaining of such injunctive relief. All of the undertakings and
* obligations relating to confidentiality and non-disclosure, whether
* contained in this section or elsewhere in this agreement, shall survive
* the termination or expiration of this agreement for a period of five (5)
* years.
*
* The Licensor agrees that any information or data received from the Licensee
* in connection with the performance of the support agreement relating to this
* software shall be confidential, will be used only in connection with the
* performance of the Licensor's obligations hereunder, and will not be
* disclosed to third parties, including contractors, without the Licensor's
* express permission in writing.
*
* Information regarding the software may be provided to the Licensee's outside
* auditors and attorneys only to the extent required by their respective
* functions.
*/#include "B2BITS_FixMessages.h"
#include "B2BITS_Exception.h"
#include "B2BITS_FIXFields.h"
#include "B2BITS_FIXGroup.h"
#include "B2BITS_FIXMessage.h"
#include "B2BITS_FIXMsgFactory.h"
#include "B2BITS_FIXMsgProcessor.h" #include "B2BITS_Mutex.h"
#include "B2BITS_Guard.h"
#include "B2BITS_Condition.h"#include "SxsMiddleware.h"
#include "SxsListener.h"#include <iostream>using namespace com::b2bits::FixServer::middleware;
using namespace std;namespace {struct ConnectionListener : public SxsListener
{
virtual void onMessage( const std::string& id, const std::string& rawFixMessage )
{
cout << endl << "Incoming message: " << rawFixMessage << endl;
}virtual void onNotification( const std::string& id, const std::string& notification )
{
cout << endl << "Notification: " << notification << endl;
} virtual void onMiddlewareError( const std::string& id, const std::string& reason )
{
cout << endl << "Error: " << reason << endl;
} virtual void onConnected( const std::string& id )
{
cout << endl << "Event: CONNECTED" << endl;
Utils::Guard< System::Mutex > l(lock_);
cond_.signal();
} virtual void onConnectionLost( const std::string& id, const std::string& reason )
{
cout << endl << "Event: CONNECTION LOST. Reason: " << reason << endl;
} virtual void onReconnectionAttemptFailed( const std::string& id )
{
} virtual void onReconnectionFailed( const std::string& id )
{
}
bool wait4connect( const SxsMiddleware& sxs, int milliseconds )
{
Utils::Guard< System::Mutex > l(lock_);
if( CONNECTED == sxs.getState() )
{ return true; }
cond_.timedWait( &lock_, milliseconds );
return ( CONNECTED == sxs.getState() );
}private: System::Mutex lock_;
System::Condition cond_;
}; } int main( int argc, char** argv )
{
try
{
// initialize FIX Engine
Engine::FixMessages::init();
// create SXS Middleware
SxsMiddleware sxs;
// connection parameters:
```

```
string clientid = "SxsClient.Net";
// host, where the FIX Edge is started
string hostname = "localhost";
// port, where FIX Edge SXS adaptor is lstening
// refer to the 'TransportLayer.SXSAdaptorDLL.listenPort' property
int port = 8129;
int reconnectionAttempts = 10;
int delay = 10;
// create connection listener
ConnectionListener listener;
// establish connection
sxs.connect(clientid, hostname, port, reconnectionAttempts, delay, &listener );
// wait while connection is established or failed (milliseconds)
int wait4confirm = 10000;
if( !listener.wait4connect(sxs, wait4confirm) )
{ throw logic_error("Cannot connect. Please check connection parameters.");}
// login
sxs.login(clientid, wait4confirm);
// create FIX Message "News"
try
{
Engine::FIXMessage* pNews = Engine::FIXMsgFactory::singleton()->newSkel( Engine::FIX40, "B" );
pNews->setSender(clientid);
pNews->setTarget("FIXEdge");
pNews->set( FIXField::Urgency, "Urgent" );
pNews->set( FIXField::OrigTime, "20060101-09:11:11");
pNews->set( FIXField::LinesOfText, 4 );
Engine::FIXGroup* pGroup = pNews->getGroup(FIXField::LinesOfText);
pGroup->set( FIXField::Text, "The 1st news line;", 0 );
pGroup->set( FIXField::Text, "The 1st news line;", 1 );
pGroup->set( FIXField::Text, "The 1st news line;", 2 );
pGroup->set( FIXField::Text, "The 1st news line;", 3 );
pNews->set( 9001, "This is a custom field" );
// send message to FIX Edge
{
int size(0);
const char* raw = pNews->toRaw(&size);
sxs.sendFixMessageToServer( string(raw, size) );
}
Engine::FIXMessage::release(pNews);
Engine::FIXMessage* pOrder = Engine::FIXMsgFactory::singleton()->newSkel( Engine::FIX44, "D" );
pOrder->setSender(clientid);
pOrder->setTarget("FIXEdge");
pOrder->set( FIXField::ClOrdID, "123" );
pOrder->set( FIXField::Symbol, "IBM" );
pOrder->set( FIXField::Side, "1" );
pOrder->set( FIXField::TransactTime, "20060101-09:11:11");
pOrder->set( FIXField::OrderQty, 100 );
pOrder->set( FIXField::OrdType, "1" );
// send message to FIX Edge
{
int size(0);
const char* raw = pOrder->toRaw(&size);
sxs.sendFixMessageToServer( string(raw, size) );
}
Engine::FIXMessage::release(pOrder);
}
catch( const Utils::Exception& ex )
{ cerr << endl << "Error : " << ex.what() << endl; } // logout from FIX Edge
sxs.logout(clientid, "Manual logout", wait4confirm);
// disconnect from FIX Edge
sxs.disconnect();
// stop FIX Engine
Engine::FixMessages::destroy();
}
catch( const Utils::Exception& ex )
{ cerr << "Error : " << ex.what() << endl; }
catch( const std::exception& ex )
{ cerr << "Error : " << ex.what() << endl; }
return 0;
}
```

.Net client

The sample SXS client application on C/C++ is represented below. It connects to the port 8129 on local machine, establishes connection, sends application messages and closes connection.

```
/*
* $Revision: 1.4.2.1 $
*
* (c) B2BITS 2007. B2BITS is an abbreviation of
* Business to Business Information Technology Services Ltd with
* its registered offices located at 2nd floor Lynton House,
* 7-12 Tavistock square, London, WC1H 9BQ. Under number 4221585.
* "Licensor" shall mean B2BITS.
*
* The Licensee acknowledges and agrees that the Software and Documentation
* (the "Confidential Information") is confidential and proprietary to
* the Licensor and the Licensee hereby agrees to use the Confidential
* Information only as permitted by the full licence agreement betweeen
* the two parties, to maintain the confidentiality of the Confidential
* Information and not to disclose the confidential information, or any part
* thereof, to any other person, firm or corporation. The Licensee
* acknowledges that disclosure of the Confidential Information may give rise
* to an irreparable injury to the Licensor in-adequately compensable in
* damages. Accordingly the Licensor may seek (without the posting of any
* bond or other security) injunctive relief against the breach of the forgoing
* undertaking of confidentiality and non-disclosure, in addition to any other
* legal remedies which may be available, and the licensee consents to the
* obtaining of such injunctive relief. All of the undertakings and
* obligations relating to confidentiality and non-disclosure, whether
* contained in this section or elsewhere in this agreement, shall survive
* the termination or expiration of this agreement for a period of five (5)
* years.
*
* The Licensor agrees that any information or data received from the Licensee
* in connection with the performance of the support agreement relating to this
* software shall be confidential, will be used only in connection with the
* performance of the Licensor's obligations hereunder, and will not be
* disclosed to third parties.
*
* Information regarding the software may be provided to the Licensee's outside
* auditors and attorneys only to the extent required by their respective
* functions.
*/
using System;
using System.Threading;
using com.b2bits.FixServer.middleware; namespace SxsClient.Net
{
/// <summary>
/// SXS Connection Listener
/// </summary>
class ConnectionListener : SxsListener
{
private object semaphore_ = new object(); #region SxsListener Memberspublic void onReconnectionAttemptFailed
(string id){} public void onConnectionLost(string id, string reason){} public void onMiddlewareError(string id,
string reason){} public void onNotification(string id, string notification){} public void onConnecting( String
id ){} public void onStateChanged( String id, State state ){} public void onConnected(string id) {
lock( semaphore_ )
{
Monitor.Pulse(semaphore_);
}
} public void onMessage(string id, string rawFixMessage) {
com.b2bits.FIXAntenna.FixMessage flatmsg = com.b2bits.FIXAntenna.FixMessage.Parse(rawFixMessage);
switch( flatmsg.MsgType )
{
case "D":
com.b2bits.FIXAntenna.Decorator.FIX44.NewOrderSingle order =
new com.b2bits.FIXAntenna.Decorator.FIX44.NewOrderSingle(flatmsg);
```

```
Console.WriteLine("\nNew order received for Client with ID '" + id + "':\n" + rawFixMessage);
break;
case "B":
com.b2bits.FIXAntenna.Decorator.FIX40.News news =
new com.b2bits.FIXAntenna.Decorator.FIX40.News(flatmsg);
Console.WriteLine("\nNews received for Client with ID '" + id + "':\n" + rawFixMessage);
break;
default:
Console.WriteLine("\nNew message received for Client with ID '" + id + "':\n" + rawFixMessage);
break;
}
} public void onReconnectionFailed(string id){} #endregion public bool wait4connect( SxsMiddleware sxs, int
milliseconds ) {
lock( semaphore_ )
{
if( State.CONNECTED == sxs.getState() )
{
return true;
}
Monitor.Wait(semaphore_, milliseconds);
}
return ( State.CONNECTED == sxs.getState() );
} }/// <summary>
/// Sample client. Connects to FIX Edge via SXS using SXS middleware.
/// </summary>
class SxsClient
{
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main(string[] args)
{
try{
// initialize FIX Engine
com.b2bits.FIXAntenna.FixMessages engine = com.b2bits.FIXAntenna.FixMessages.Create("engine.properties");
// create SXS Middleware
SxsMiddleware sxs = new SxsMiddlewareImpl();
// connection parameters:
string clientid = "SxsClient.Net";
// host, where the FIX Edge is started
string hostname = "localhost";
// port, where FIX Edge SXS adaptor is lstening
// refer to the 'TransportLayer.SXSAdaptorDLL.listenPort' property
int port = 8129;
int reconnectionAttempts = 10;
int delay = 10;
// create connection listener
ConnectionListener listener = new ConnectionListener();
try{
// establish connection
sxs.connect(clientid, hostname, port, reconnectionAttempts, delay, listener );
// wait while connection is established or failed
int wait4confirm = 1000;
if( !listener.wait4connect(sxs, wait4confirm) )
{
throw new Exception("Cannot connect. Please check connection parameters.");
}
// login
sxs.login(clientid, wait4confirm);
// create FIX Message "News" using object model
com.b2bits.FIXAntenna.Decorator.FIX40.News news = new com.b2bits.FIXAntenna.Decorator.FIX40.News();
// set various fields
news.MessageHeader.SenderCompID = clientid;
news.MessageHeader.TargetCompID = "FIXEdge";
news.Urgency = "Urgent";
news.OrigTime = DateTime.Now.ToUniversalTime().ToString("yyyyMMdd-HH:mm:ss");
news.NoLinesOfText = 4;
news.LinesOfText[0].Text = "The 1st news line;";
news.LinesOfText[1].Text = "The 2nd news line;";
news.LinesOfText[2].Text = "The 3rd news line;";
```

```
news.LinesOfText[3].Text = "The 4th news line.";
news.SetField( 9001, "This is a custom field" );
// send message to FIX Edge
sxs.sendFixMessageToServer( news.FlatMessage.ToString() ); // create FIX Message "New Order Single" usign
object model com.b2bits.FIXAntenna.Decorator.FIX44.NewOrderSingle order = new com.b2bits.FIXAntenna.Decorator.
FIX44.NewOrderSingle();
order.MessageHeader.SenderCompID = clientid;
order.MessageHeader.TargetCompID = "FIXEdge";
order.ClOrdID = "123";
order.Symbol = "IBM";
order.Side = "1";
order.TransactTime = DateTime.Now.ToUniversalTime().ToString("yyyyMMdd-HH:mm:ss");
order.OrderQty = 100;
order.OrdType = "1";
// send message to FIX Edge
sxs.sendFixMessageToServer( order.FlatMessage.ToString() );
// logout from FIX Edge
sxs.logout(clientid, "Manual logout", wait4confirm);
}
catch( Exception ex )
{
Console.WriteLine("Error: " + ex.Message );
}
// disconnect from FIX Edge
sxs.disconnect();
// stop FIX Engine
engine.Stop();
}
catch( Exception ex )
{
Console.WriteLine("Error: " + ex.Message );
}
}
}
}
```