SFS Transport Adaptor Programming Guide

Overview

The Simple FIX Socket Transport Adaptor (hereinafter SFS) provides the easy and quick way for the client applications to communicate with FIXEdge using FIX protocol. The Session level of the FIX protocol requires to control various aspects: messages sequence numbering, sequence gaps resolving, session state monitoring, etc. Client application doesn't need to support FIX session level (except Logon and Logout messages) when communicates with SFS adaptor.

SFS adaptor uses the TCP sockets and works as acceptor.

SFS Messages

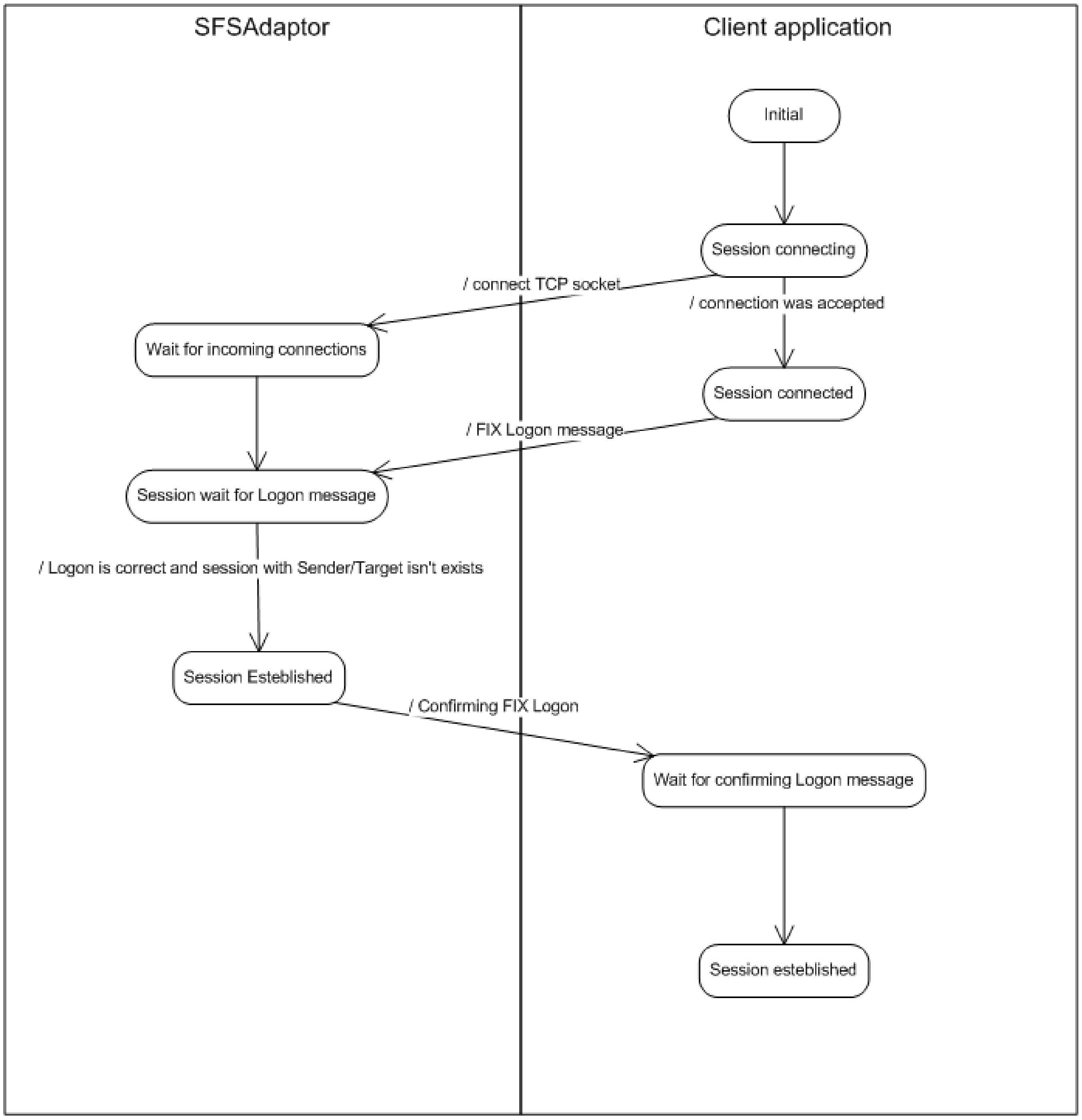
SFS adaptor supports all FIX.4x protocols. The following FIX messages can be sent to the SFS adaptor (other types of FIX messages aren't supported and handled):

- All Application-level FIX messages of the FIX.4x protocol
- FIX Logon message of the FIX.4x protocol - used to establish session between SFS adaptor and Client application
- FIX Logout message of the FIX.4x protocol - used to close session between SFS adaptor and Client application

Each SFS message should be formed according to the rules which are specified in the FIX protocol.

SFS Session

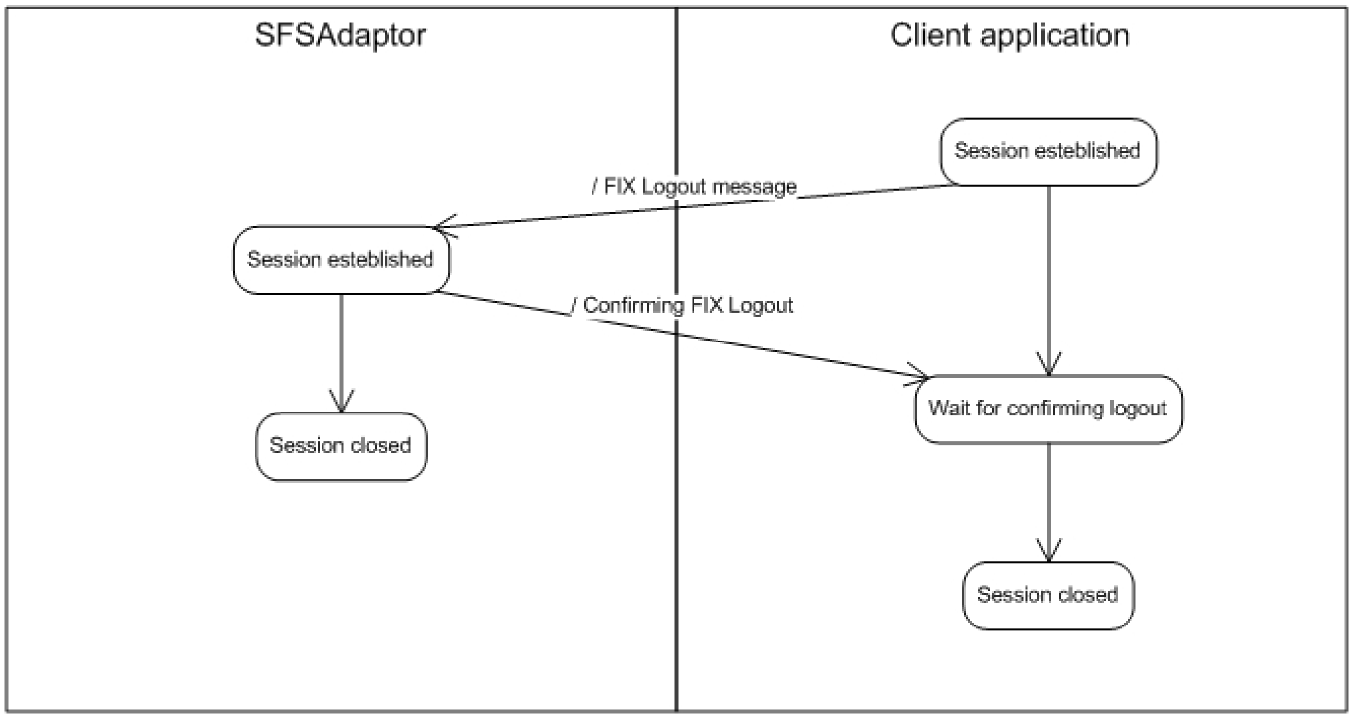
To establish new session Client Application should connect to the SFS adaptor listen socket and send FIX Logon message. SFS adaptor accepts new connection and handles received message. If Logon message is correct and SFS hasn't session with same SenderCompId and TargetCompId, SFS adaptor will send Confirming Logon message. When Client Application receives confirming FIX Logon message - session is established.


SFSAdaptor
Client application
Initial
Session connecting
/ connect TCP socket
/ connection was accepted
Wait for incoming connections
Session connected
/ FIX Logon message
Session wait for Logon message
/ Logon is correct and session with Sender/Target isn't exists
Session Esteblished
/ Confirming FIX Logon
Wait for confirming Logon message
Session esteblished

The established session closes in following situations:

- Client Application sends FIX Logout message
- Socket was closed
- Client Application sends any FIX session level message - duplicate Logon message, Heartbeat, Reject, Resend Request, Sequence Reset or Test Request
- SFS adaptor is terminating

To close connection Client Application has to send FIX Logout message and wait for the confirming Logout that SFSAdaptor will send.



Client application sample

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

```
/**
* $Revision: 1.5.2.1 $
*
* (c) B2BITS 2003. 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.
*
* 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 "iostream"
#include "string"
#include "Winsock2.h"
using namespace std;
string getMessageType(const string& msg);
int main(int argc, char* argv[])
{
if(3 != argc){
cout<< "Usage: \nSFSClient.exe hostip port"<< endl;
return 1;
}
WSADATA wsaData;
if(WSAStartup(MAKEWORD(2,2), &wsaData)) // WSAStartup failed
return 1;
// resolve host name into address
unsigned long addr = inet_addr(argv[1]);
in_addr address;
memcpy(&address, &addr, sizeof(address));
int port = atoi(argv[2]);
// connect to the SFS adaptor
SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);
if (INVALID_SOCKET == sock){
cout<< "Unable to create socket!"<< endl;
return 1;
}
struct sockaddr_in sockAddr;
memset(&sockAddr,'\0',sizeof(sockAddr));
sockAddr.sin_family = AF_INET;
sockAddr.sin_addr = address;
sockAddr.sin_port = htons(port);
if(0 != connect(sock, (struct sockaddr*) &sockAddr ,sizeof(sockAddr))){
cout<< "Unable to connect socket!"<< endl;
closesocket(sock);
return 1;
}
// send Fix Logon message
string logonMsg = "8=FIX.4.49=6635=A49=TSender56=Receiver34=152=20070219-10:12:5298=0108=3010=250";
int ret = send(sock, logonMsg.c_str(), static_cast<int>(logonMsg.size()), 0);
if(SOCKET_ERROR == ret){
cout<< "Unable to send FIX Logon message into the socket!"<< endl;
closesocket(sock);
return 1;
}
// receive Confirming FIX Logon message
char buffer[1024];
ret = ::recv(sock, buffer, 1023, 0);
if(SOCKET_ERROR == ret){
cout<< "Unable to send FIX Logon message into the socket!"<< endl;
closesocket(sock);
return 1;
}
// verify that it's Logon message
if("A" != getMessageType(string(buffer, ret))){
cout<< "The received message isn't a corfirming logon message!"<< endl;
closesocket(sock);
return 1;
}
// send Fix Order message
string orderMsg = "8=FIX.4.49=12835=D49=TSender56=Receiver34=250=3073797=Y52=20070219-10:12:
5411=9000100855=TESTB54=160=20060217-10:00:0038=400040=110=048"; ret = send(sock, orderMsg.c_str(),
static_cast<int>(orderMsg.size()), 0);
if(SOCKET_ERROR == ret){
cout<< "Unable to send FIX NewOrderSingle message into the socket!"<< endl;
closesocket(sock);
return 1;
}
// send Fix Logout message
string logoutMsg = "8=FIX.4.49=8135=549=TSender56=Receiver34=4000252=20070219-10:12:5458=Sample was
complete10=118";
ret = send(sock, logoutMsg.c_str(), static_cast<int>(logoutMsg.size()), 0);
if(SOCKET_ERROR == ret){
cout<< "Unable to send FIX NewOrderSingle message into the socket!"<< endl;
closesocket(sock);
return 1;
}
// receive Confirming FIX Logon message
ret = ::recv(sock, buffer, 1023, 0);
if(SOCKET_ERROR == ret){
cout<< "Unable to send FIX Logon message into the socket!"<< endl;
closesocket(sock);
return 1;
}
// verify that it's Logon message
if("5" != getMessageType(string(buffer, ret))){
cout<< "The received message isn't a corfirming logout message!"<< endl;
closesocket(sock);
return 1;
}
//closing socket
closesocket(sock);
return 0;
}
string getMessageType(const string& msg)
{
size_t beginPos = msg.find("8=FIX.4.4");
if(string::npos == beginPos)
return "";
size_t msgSizeBegins = msg.find("9=", beginPos);
if(string::npos == msgSizeBegins)
return "";
size_t msgSizeEnds = msg.find("", msgSizeBegins + 1);
if(string::npos == msgSizeEnds)
return "";
string msgLen = msg.substr(msgSizeBegins + 3, msgSizeEnds - msgSizeBegins - 3);
int msgSize = atoi(msgLen.c_str());
size_t msgTypeBegins = msg.find("35=", msgSizeEnds);
if((string::npos == msgTypeBegins)||(msgTypeBegins > msgSizeEnds + msgSize))
return "";
size_t msgTypeEnds = msg.find("", msgTypeBegins + 1);
if((string::npos == msgTypeEnds)||(msgTypeEnds > msgSizeEnds + msgSize))
return "";
return msg.substr(msgTypeBegins + 4, msgTypeEnds - msgTypeBegins - 4);
}
```