

www.nj4x.com

NJ4X API – library of access to MetaTrader server for Java and .Net

Guideline

Gerasimenko R. A.
2009 - 2012

Table of Contents

Table of Contents	2
Terminology	2
Overview	3
NJ4X API applying guideline	5
Sequence Diagrams	7
NJ4X program initialization	8
NJ4X program finishing	8
NJ4X Terminal Server	10
Addition 1: list of NJ4X API methods	11

Terminology

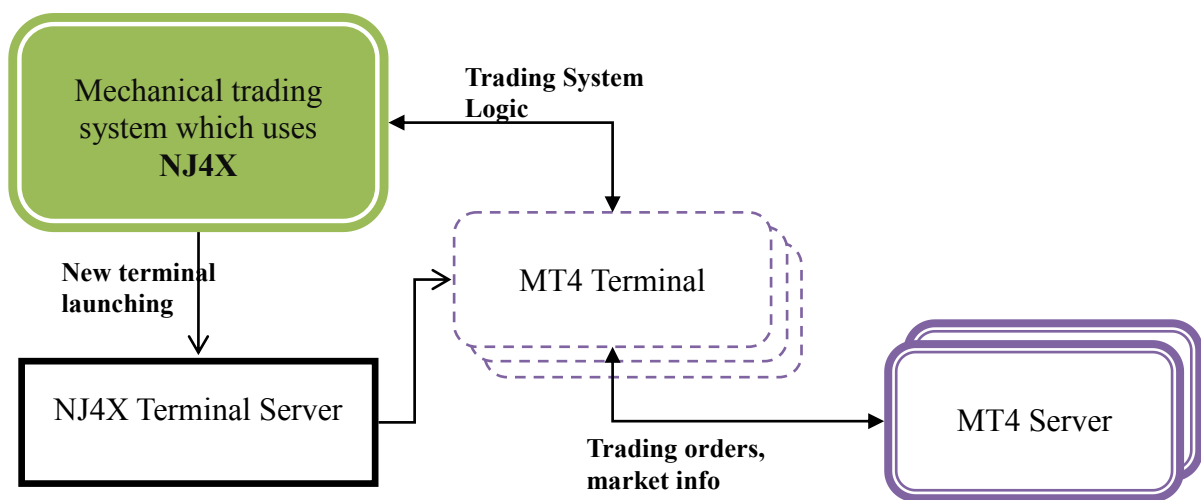
Term	Description
Advisor (EA)	Advisor (Expert Advisor) is mechanical trading system (MTS)
NJ4X	Java and .Net interface for access to MT4 terminal , previous name – JFX API
NJ4X Server	Socket server (part of Java and .Net interface for access to MT4 terminal), which receives connections from MT4 terminal and initializes NJ4X strategies
NJ4X Strategy	Trading strategy, realized via NJ4X API library. Usually it is presented by java-class, inherited from <i>com.jfx.strategy.Strategy</i> or .Net object, inherited from <i>nj4x.Strategy</i> .
MT4 Server	MetaQuotes Software Corp. trading server. MT4 server receives trading orders and other requests from trading terminals for further execution
MT4 Terminal	Client MetaTrader 4 terminal is a trader workplace, which allows to work on Forex, CFD and Futures financial markets. Java and .Net access interface (NJ4X) launches MT4 terminal in background mode and uses it for access to MT4 Servers
MQL4	Programming language by MetaQuotes Software Corp for development of advisors , which interact with MT4 trading servers .
NJ4X Terminal Server	Utility that launches MT4 terminals in background mode on request from mechanical trading system, which uses the NJ4X library.

Overview

This document presents MetaTrader access library for Java and .Net – NJ4X API. It is a toolkit for mechanical trading system developers, which provides access to MetaTrader servers by the use of standard MT4 terminal.

NJ4X also allows to avoid applying of MQL4 language. Building mechanical trading systems (advisors) is available in the most convenient development environments.

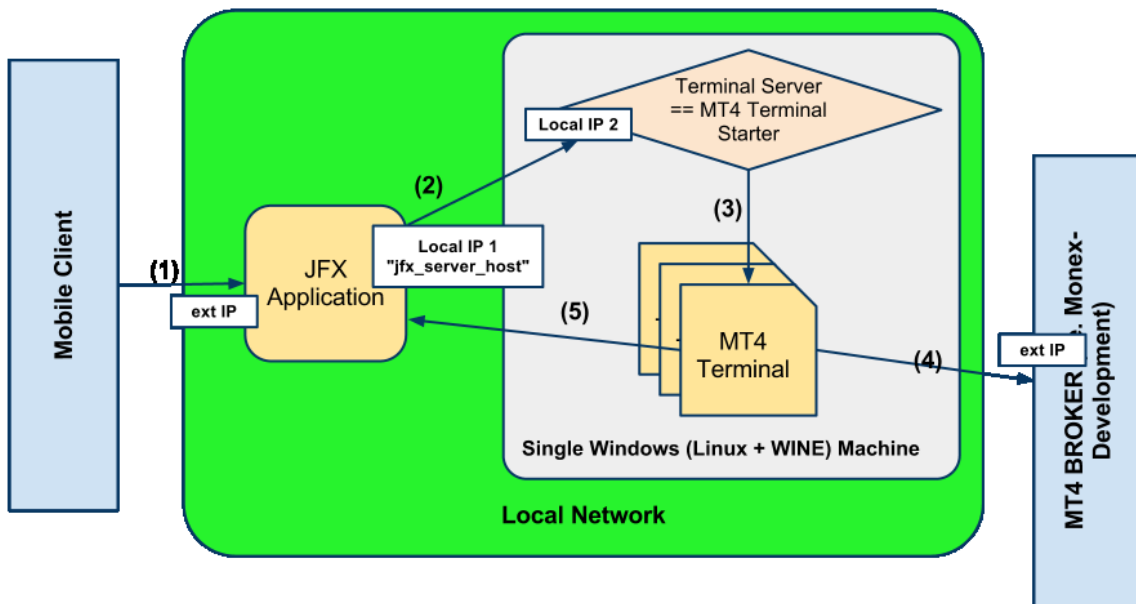
Role of MT4 terminal is reduced to mediation between MT4 Server and Java or .Net program, which uses the NJ4X library:



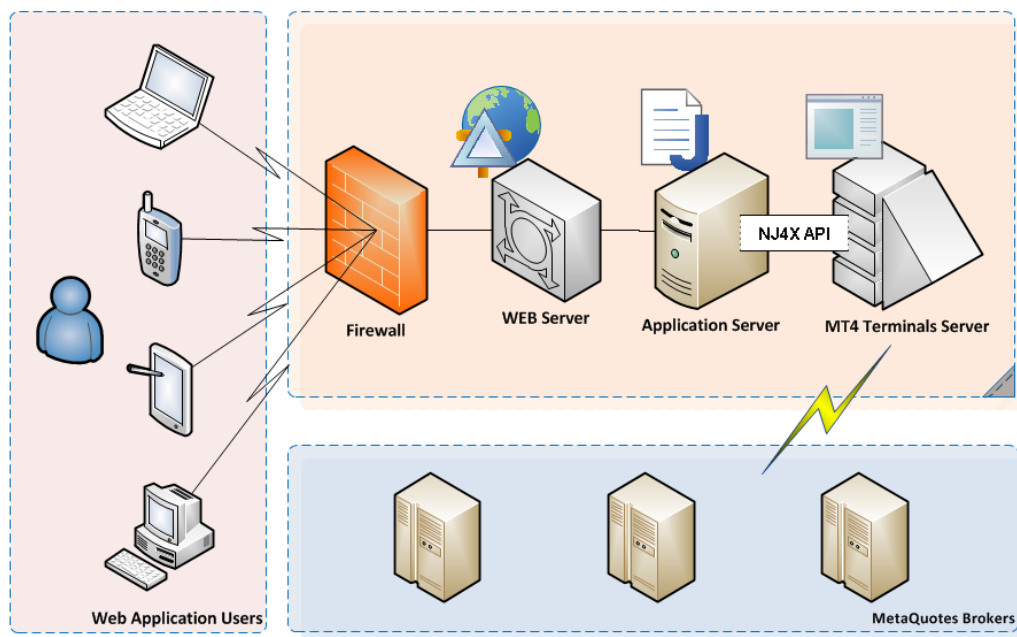
Using the NJ4X library, single Java or .Net program can work with several MT4 servers simultaneously. There are some other important aspects:

- Java/.Net advisors are developed without any restrictions about development environment (IntelliJ IDEA, MS VS2010, NetBeans and Eclipse are available)
- It makes possible automated administration of several trading accounts simultaneously.
- Compound trading systems can be well structured. MQL isn't relevant to that purpose.

Scheme of network connections in typical NJ4X library applying case:



- (1) Client connects to JFX Application and initiates connect(...) method
- (2) JFX Application connects to the "Terminal Server" asking to start MT4 Terminal
- (3) "Terminal Server" starts MT4 Terminal
- (4) MT4 Terminal connects to Broker and (if succeeded) connects back to JFX Application (5)



NJ4X API applying guideline

NJ4X library consists of four parts:

- **jfx.ex4** –NJ4X advisor for MT4 terminal
- **mt4if.dll** – communication library for connection between MT4 terminal and Java / .Net program.
- Library itself: of Java classes (**jfx.jar**) and .Net (**nj4x.dll**).
- NJ4X Terminal Server – program, which launches MT4 terminals in background mode on request from mechanical trading system, which uses NJ4X library.

There are two ways of NJ4X applying:

1. **MT4 terminal operates:** you start the MT4 terminal and manually drop **jfx.ex4** advisor to the chart. Then Expert Advisor connects to your Java or .Net program (see the Sequence Diagrams).
2. **Java / .Net program operates:** you make a Java or .Net program, which connects to broker using given MT4 account details. It's just as simple as

```
demoNJ4X.connect("127.0.0.1", 7788, Broker.AlpariUS_Demo, "982388",  
"gtx8TCo");
```

NJ4X Terminal Server launches MT4 terminal (terminal.exe) in background mode so that your mechanical trading strategy could execute its trading logic. Terminal NJ4X Server places on your computer (in user's home catalogue) a separate directory structure for each MT4 terminal (e.g. c:\Users\brian\jfx_terminals or C:\Documents and Settings\brian\jfx_terminals).

To start using NJ4X library in your Java application, you have to create your own class, inheriting it from *com.jfx.strategy.Strategy* and redefine the *coordinate()* method, e.g.

```
public class MyStrategy extends com.nj4x.strategy.Strategy {
    public void init(String symbol, int period, StrategyRunner strategyRunner) {
        super.init(symbol, period, strategyRunner);
        //
        //current orders loading, restoration to the previous launching moment.
        //
    }

    public void deinit() {
        // releasing owned resources (e.g opened files closing) when leaving the advisor
    }

    public void coordinate() {
        // here can be placed the logic of mechanical trading system
        /* use any API method: accountBalance, accountCompany, accountCredit, accountCurrency, accountEquity,
        accountFreeMargin, accountMargin, accountName, accountNumber, accountProfit, comment, day, dayOfWeek, dayOfYear,
        getLastError, getTickCount, hour, iAC, iAD, iADX, iAlligator, iAO, iATR, iBands, iBars, iBarShift, iBearsPower, iBullsPower,
        iBWMFI, iCCl, iClose, iCustom, iDeMarker, iEnvelopes, iForce, iFractals, iGator, iHigh, iHighest, iLow, iLowest, iMA, iMACD,
        iMFI, iMomentum, iOBV, iOpen, iOsMA, iRSI, iRVI, iSAR, isConnected, isDemo, iStdDev, isTesting, iStochastic,
        isTradeContextBusy, isVisualMode, iTime, iVolume, iWPR, marketInfo, minute, month, objectCreate, objectCreate, objectCreate,
        objectDelete, objectGet, objectGetFiboDescription, objectSet, objectSetFiboDescription, objectSetText, objectsTotal, objectType,
        orderClose, orderCloseBy, orderClosePrice, orderCloseTime, orderComment, orderCommission, orderDelete, orderExpiration,
        orderLots, orderMagicNumber, orderModify, orderOpenPrice, orderOpenTime, orderPrint, orderProfit, orderSelect, orderSend,
        ordersHistoryTotal, orderStopLoss, ordersTotal, orderSwap, orderSymbol, orderTakeProfit, orderTicket, orderType, print,
        refreshRates, seconds, timeCurrent, year
        */
    }
}
```

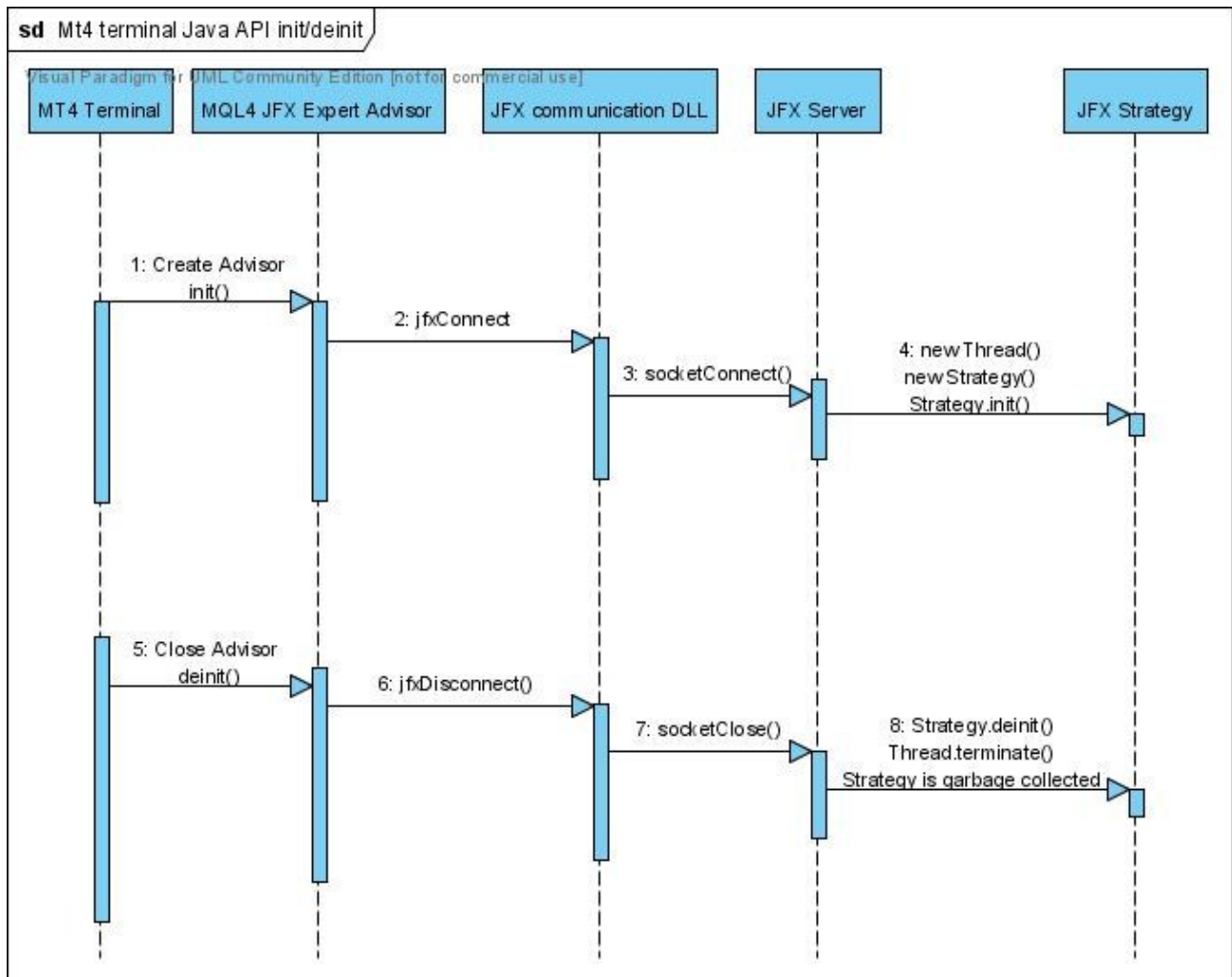
In .Net program you can use the *nj4x.Strategy* class straightway, e.g. (C#):

```
// Create strategy
var mt4 = new nj4x.Strategy();
// Connect to the Terminal Server
mt4.Connect(
    termSrvHost: "127.0.0.1",
    termSrvPort: 7788,
    mt4Server: new Broker(ConfigurationManager.AppSettings["broker"]),
    mt4User: ConfigurationManager.AppSettings["account"],
    mt4Password: ConfigurationManager.AppSettings["password"]
);
// Use API methods ...
Console.WriteLine(String.Format("Account {0}", mt4.AccountNumber()));
Console.WriteLine(String.Format("Equity {0}", mt4.AccountEquity()));
//
Console.ReadLine();
```

Sequence Diagrams

Java or .Net advisor class name is transmitted to NJ4X by MT4 terminal. The NJ4X library automatically creates new copy of this class and starts trading information exchange.

Example of events sequence diagram at the beginning and at the end of NJ4X application work is given below:



NJ4X program initialization

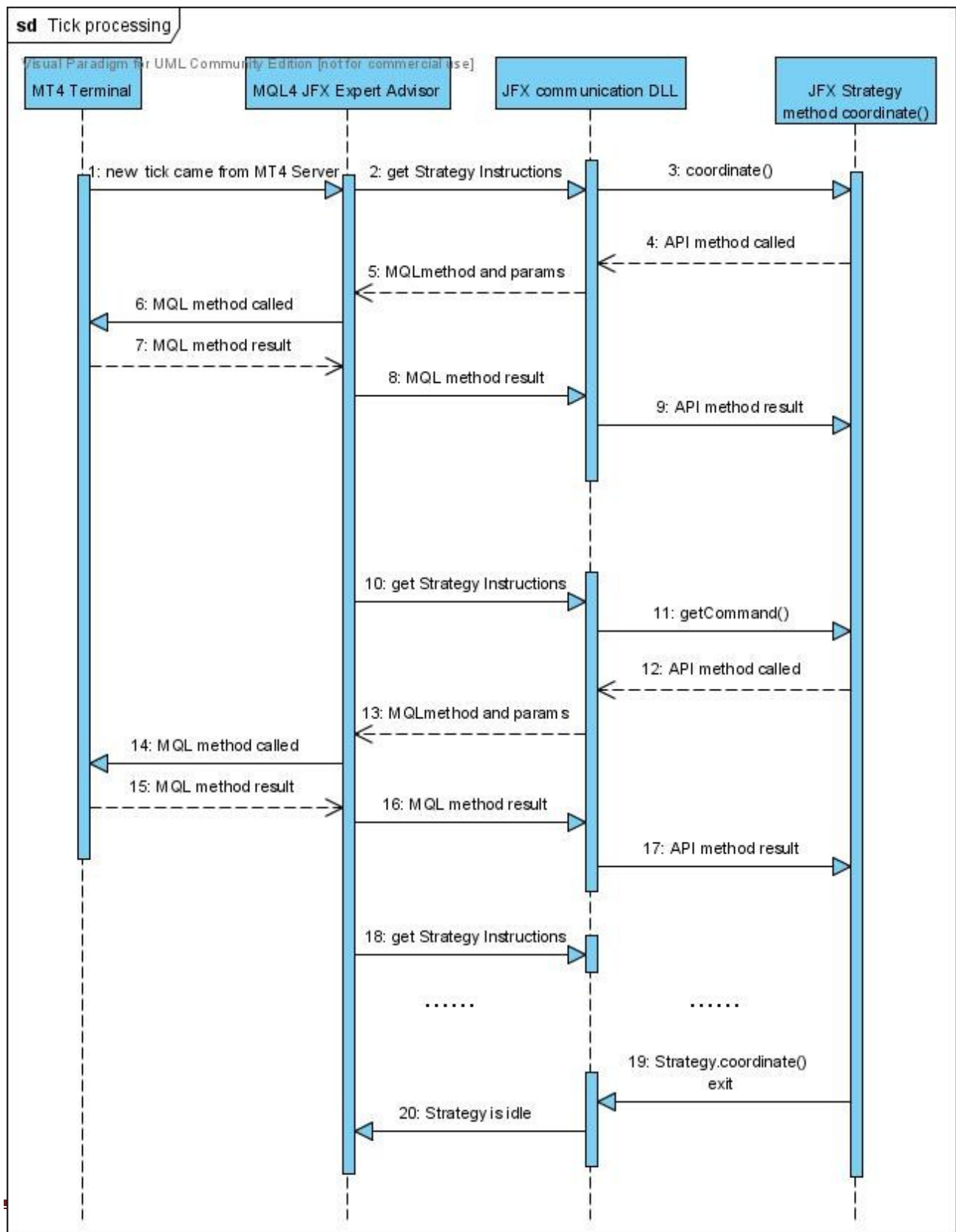
- 1: Create Advisor – at the launch of standard NJ4X (jfx.ex4) advisor, MT4 terminal calls his init() method
- 2: NJ4XConnect – advisor's init() method connects to Java or .Net NJ4X program via communication library MT4IF.DLL (which is a part of NJ4X)
- 3: socketConnect – MT4IF.DLL library establishes a TCP/IP connection with NJ4X Server (which launches automatically at the moment of appealing to com.jfx.net.JFXServer.getInstance() or to nj4x.Net.NJ4XServer.Instance in NJ4X program) and returns session identifier to jfx.ex4 advisor.
- 4: new Thread/Strategy – during connection, NJ4X Server creates new copy of Java or .Net MTS class.

NJ4X program finishing

- 5: Close Advisor – when MT4 terminal deletes jfx.ex4 advisor from chart of chosen tool, special advisor method – deinit() – is called.
- 6: NJ4XDisconnect() - inside deinit() method occurs closing of TCP/IP connection with program's NJ4X Server via communication library MT4IF.DLL (which is a part of NJ4X).
- 7: socketClose() - MT4IF.DLL library closes TCP/IP connection with NJ4X program and disposes session identifier.
- 8: Strategy.deinit() – copy of Java or .Net class released.

If NJ4X program finishing occurs because of loss of network connection with MT4 terminal, communication library MT4IF.DLL will try to restore the connection automatically. Henceforth everything will be done by standard NJ4X program initialization algorithm.

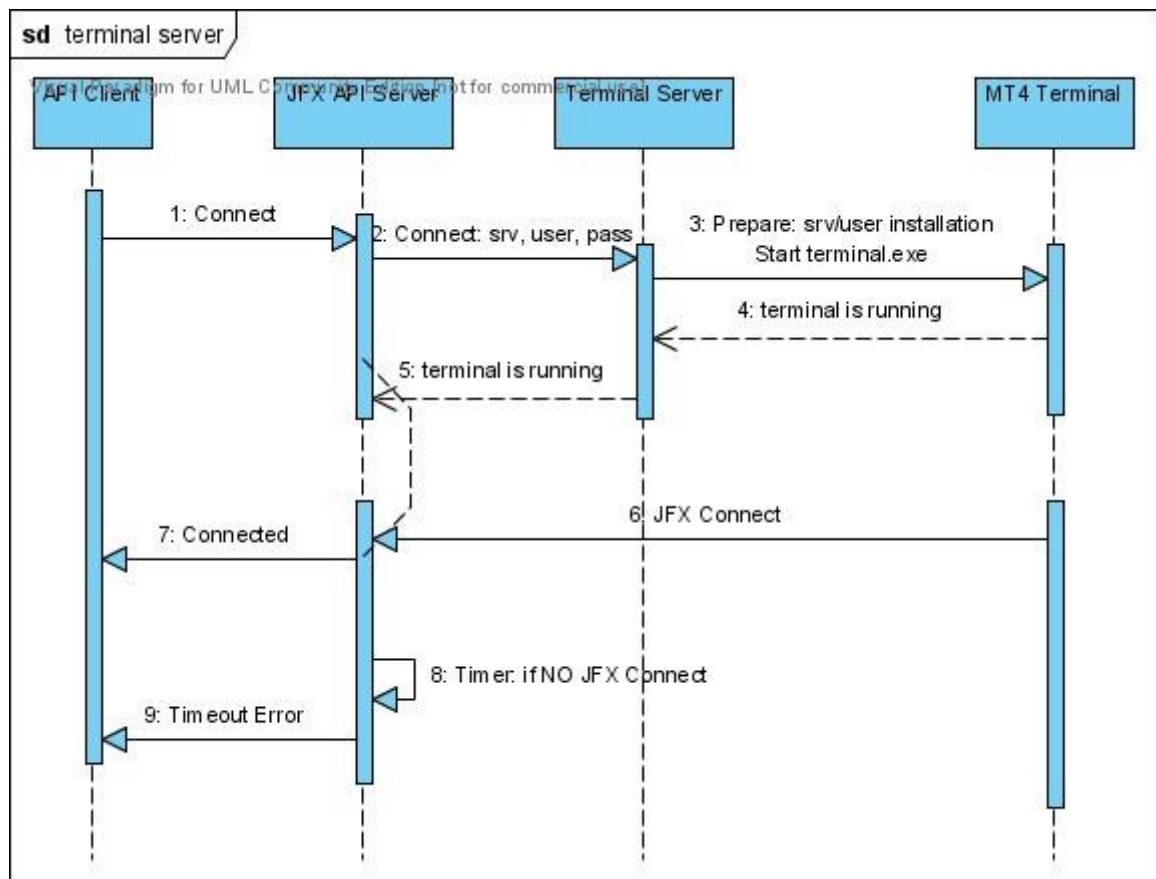
After NJ4X program initialization executes processing of trading signals and sending operating actions from trading system to MT4 server:



NJ4X Terminal Server

NJ4X Terminal Server conducts the MT4 terminal launching in background mode if receiving operating signals from NJ4X program, e.g.

```
public class NJ4XExample extends com.jfx.strategy.Strategy{...
    connect("127.0.0.1" /*terminal server adress*/, 7788 /*port*/,
        Broker.AlpariUK_Demo /*broker*/, "1107623" /*user*/, "w4ylog" /*password*/);
}
```



Addition 1: list of NJ4X API methods

Method	Description
<u>Marketinfo(String, MarketInfo)</u>	
<u>MarketinfoModeTime(String)</u>	Returns the last incoming tick time (last known server time).
<u>ToDate(Double)</u>	Converts MT4 terminal date to DateTime
<u>iAD(String, Timeframe, Int32)</u>	Calculates the Accumulation/Distribution indicator and returns its value
<u>iBearsPower(String, Timeframe, Int32, AppliedPrice, Int32)</u>	Calculates the Bears Power indicator and returns its value
<u>iAC(String, Timeframe, Int32)</u>	Calculates the Bill Williams' Accelerator/Decelerator oscillator
<u>iAlligator(String, Timeframe, Int32, Int32, Int32, Int32, Int32, Int32, MovingAverageMethod, AppliedPrice, GatorMode, Int32)</u>	Calculates the Bill Williams' Alligator and returns its value
<u>iAO(String, Timeframe, Int32)</u>	Calculates the Bill Williams' Awesome oscillator and returns its value
<u>iBWMFI(String, Timeframe, Int32)</u>	Calculates the Bill Williams Market Facilitation index and returns its value
<u>iBullsPower(String, Timeframe, Int32, AppliedPrice, Int32)</u>	Calculates the Bulls Power indicator and returns its value
<u>iCCI(String, Timeframe, Int32, AppliedPrice, Int32)</u>	Calculates the Commodity channel index and returns its value
<u>iDeMarker(String, Timeframe, Int32, Int32)</u>	Calculates the DeMarker indicator and returns its value
<u>iEnvelopes(String, Timeframe, Int32, AppliedPrice, Int32)</u>	Calculates the Envelopes indicator and returns its value

<u>MovingAverageMethod, Int32, AppliedPrice, Double, BandsIndicatorLines, Int32)</u>	value
<u>iForce(String, Timeframe, Int32, MovingAverageMethod, AppliedPrice, Int32)</u>	Calculates the Force index and returns its value
<u>iFractals(String, Timeframe, BandsIndicatorLines, Int32)</u>	Calculates the Fractals and returns its value
<u>iIchimoku(String, Timeframe, Int32, Int32, Int32, IchimokuSource, Int32)</u>	Calculates the Ichimoku Kinko Hyo and returns its value
<u>iATR(String, Timeframe, Int32, Int32)</u>	Calculates the Indicator of the average true range and returns its value
<u>iWPR(String, Timeframe, Int32, Int32)</u>	Calculates the Larry William's percent range indicator and returns its value
<u>iMomentum(String, Timeframe, Int32, AppliedPrice, Int32)</u>	Calculates the Momentum indicator and returns its value
<u>iMFI(String, Timeframe, Int32, Int32)</u>	Calculates the Money flow index and returns its value
<u>iADX(String, Timeframe, Int32, AppliedPrice, ADXIndicatorLines, Int32)</u>	Calculates the Movement directional index and returns its value
<u>iBands(String, Timeframe, Int32, Int32, Int32, AppliedPrice, BandsIndicatorLines, Int32)</u>	Calculates the Movement directional index and returns its value
<u>iMA(String, Timeframe, Int32, Int32, MovingAverageMethod, AppliedPrice, Int32)</u>	Calculates the Moving average indicator and returns its value
<u>iOsMA(String, Timeframe, Int32, Int32, Int32, AppliedPrice, Int32)</u>	Calculates the Moving Average of Oscillator and returns its value
<u>iMACD(String, Timeframe, Int32, Int32, Int32, AppliedPrice, MACDIndicatorLines, Int32)</u>	Calculates the Moving averages convergence/divergence and returns its value
<u>iOBV(String, Timeframe, AppliedPrice, Int32)</u>	Calculates the On Balance Volume indicator and returns its value
<u>iSAR(String, Timeframe, Double, Double,</u>	Calculates the Parabolic Stop and Reverse system

<u>Int32)</u>	and returns its value
<u>iRSI(String, Timeframe, Int32, AppliedPrice, Int32)</u>	Calculates the Relative strength index and returns its value
<u>iRVI(String, Timeframe, Int32, MACDIndicatorLines, Int32)</u>	Calculates the Relative Vigor index and returns its value
<u>iCustom(String, Timeframe, String, Int32, Int32)</u>	Calculates the specified custom indicator and returns its value
<u>iStdDev(String, Timeframe, Int32, Int32, MovingAverageMethod, AppliedPrice, Int32)</u>	Calculates the Standard Deviation indicator and returns its value
<u>iStochastic(String, Timeframe, Int32, Int32, Int32, MovingAverageMethod, Int32, MACDIndicatorLines, Int32)</u>	Calculates the Stochastic oscillator and returns its value
<u>AccountFreeMarginMode()</u>	Calculation mode of free margin allowed to open positions on the current account
<u>ObjectSetText(String, String, Int32, String, Color)</u>	Changes the object description
<u>ObjectSet(String, ObjectProperty, Double)</u>	Changes the value of the specified object property
<u>OrderCloseBy(Int32, Int32, Color)</u>	Closes an opened order by another opposite opened order
<u>OrderClose(Int32, Double, Double, Int32, Color)</u>	Closes opened order
<u>ObjectCreate(String, ObjectType, Int32, DateTime, Double, DateTime, Double, DateTime, Double)</u>	Creation of an object with the specified name, type and initial coordinates in the specified window
<u>ObjectCreate(String, ObjectType, Int32, DateTime, Double)</u>	Creation of an object with the specified name, type and initial coordinates in the specified window
<u>ObjectCreate(String, ObjectType, Int32, DateTime, Double, DateTime, Double)</u>	Creation of an object with the specified name, type and initial coordinates in the specified window
<u>GlobalVariablesDeleteAll(String)</u>	Deletes global variables

<u>ObjectDelete(String)</u>	Deletes object having the specified name
<u>OrderDelete(Int32, Color)</u>	Deletes previously opened pending order
<u>GlobalVariableDel(String)</u>	Deletes the global variable
<u>Alert(String)</u>	Displays a dialog box containing the user-defined data
<u>PlaySound(String)</u>	Function plays a sound file
<u>WindowBarsPerChart()</u>	Function returns the amount of bars visible on the chart
<u>iGator(String, Timeframe, Int32, Int32, Int32, Int32, Int32, Int32, MovingAverageMethod, AppliedPrice, BandsIndicatorLines, Int32)</u>	Gator oscillator calculation
<u>WindowFind(String)</u>	If indicator with name was found, the function returns the window index containing this specified indicator, otherwise it returns -1
<u>OrderModify(Int32, Double, Double, Double, DateTime, Color)</u>	Modification of characteristics for the previously opened position or pending orders
<u>Print(String)</u>	Prints a message to the experts log
<u>OrderPrint()</u>	Prints information about the selected order in the log in the following format: ticket number; open time; trade operation; amount of lots; open price; Stop Loss; Take Profit; close time; close price; commission; swap; profit; comment; magic number; pending order expiration date
<u>WindowRedraw()</u>	Redraws the current chart forcibly
<u>RefreshRates()</u>	Refreshing of data in pre-defined variables and series arrays
<u>ObjectsDeleteAll(Int32, Int32)</u>	Removes all objects of the specified type and in the specified sub-window of the chart

<u>ObjectDescription(String)</u>	Return object description
<u>Symbol()</u>	Returns a text string with the name of the current financial instrument
<u>OrderLots()</u>	Returns amount of lots for the selected order
<u>OrderMagicNumber()</u>	Returns an identifying (magic) number for the currently selected order
<u>AccountBalance()</u>	Returns balance value of the current account (the amount of money on the account)
<u>OrderCommission()</u>	Returns calculated commission for the currently selected order
<u>TerminalName()</u>	Returns client terminal name
<u>OrderClosePrice()</u>	Returns close price for the currently selected order
<u>OrderCloseTime()</u>	Returns close time for the currently selected order
<u>iClose(String, Timeframe, Int32)</u>	Returns Close value for the bar of indicated symbol with timeframe and shift
<u>OrderComment()</u>	Returns comment for the selected order
<u>WindowsTotal()</u>	Returns count of indicator windows on the chart (including main chart)
<u>AccountCredit()</u>	Returns credit value of the current account
<u>AccountCurrency()</u>	Returns currency name of the current account
<u>AccountEquity()</u>	Returns equity value of the current account
<u>OrderExpiration()</u>	Returns expiration date for the selected pending order

<u>AccountFreeMarginCheck(String, TradeOperation, Double)</u>	Returns free margin that remains after the specified position has been opened at the current price on the current account
<u>AccountFreeMargin()</u>	Returns free margin value of the current account
<u>iHigh(String, Timeframe, Int32)</u>	Returns High value for the bar of indicated symbol with timeframe and shift
<u>AccountLeverage()</u>	Returns leverage of the current account
<u>iLow(String, Timeframe, Int32)</u>	Returns Low value for the bar of indicated symbol with timeframe and shift
<u>AccountMargin()</u>	Returns margin value of the current account
<u>OrdersTotal()</u>	Returns market and pending orders count
<u>WindowPriceMax(Int32)</u>	Returns maximal value of the vertical scale of the specified subwindow of the current chart (0-main chart window, the indicators' subwindows are numbered starting from 1)
<u>WindowPriceMin(Int32)</u>	Returns minimal value of the vertical scale of the specified subwindow of the current chart (0-main chart window, the indicators' subwindows are numbered starting from 1)
<u>WindowExpertName()</u>	Returns name of the executed expert, script, custom indicator, or library, depending on the MQL4 program, from which this function has been called
<u>OrderOpenPrice()</u>	Returns open price for the currently selected order
<u>OrderOpenTime()</u>	Returns open time for the currently selected order
<u>iOpen(String, Timeframe, Int32)</u>	Returns Open value for the bar of indicated symbol with timeframe and shift

<u>OrderType()</u>	Returns order operation type for the currently selected order
<u>AccountProfit()</u>	Returns profit value of the current account
<u>OrderStopLoss()</u>	Returns stop loss value for the currently selected order
<u>OrderSwap()</u>	Returns swap value for the currently selected order
<u>OrderTakeProfit()</u>	Returns take profit value for the currently selected order
<u>Period()</u>	Returns the amount of minutes determining the used period (chart timeframe)
<u>Seconds()</u>	Returns the amount of seconds elapsed from the beginning of the current minute of the last known server time by the moment of the program start (this value will not change within the time of the program execution)
<u>AccountCompany()</u>	Returns the brokerage company name where the current account was registered
<u>AccountStopoutMode()</u>	Returns the calculation mode for the Stop Out level
<u>UninitializeReason()</u>	Returns the code of the uninitialization reason for the experts, custom indicators, and scripts
<u>AccountServer()</u>	Returns the connected server name
<u>AccountName()</u>	Returns the current account name
<u>Day()</u>	Returns the current day of the month, i.e., the day of month of the last known server time
<u>DayOfYear()</u>	Returns the current day of the year (1 means 1 January,...,365(6) does 31 December), i.e., the day of year of the last known server time

<u>Minute()</u>	Returns the current minute (0,1,2,...59) of the last known server time by the moment of the program start (this value will not change within the time of the program execution)
<u>Month()</u>	Returns the current month as number (1-January,2,3,4,5,6,7,8,9,10,11,12), i.e., the number of month of the last known server time
<u>Year()</u>	Returns the current year, i.e., the year of the last known server time
<u>DayOfWeek()</u>	Returns the current zero-based day of the week (0-Sunday,1,2,3,4,5,6) of the last known server time
<u>TerminalPath()</u>	Returns the directory, from which the client terminal was launched
<u>Hour()</u>	Returns the hour (0,1,2,...23) of the last known server time by the moment of the program start (this value will not change within the time of the program execution)
<u>TimeCurrent()</u>	Returns the last known server time (time of incoming of the latest quote) as number of seconds elapsed from 00:00 January 1, 1970
<u>TerminalCompany()</u>	Returns the name of company owning the client terminal
<u>OrderProfit()</u>	Returns the net profit value (without swaps or commissions) for the selected order
<u>iBars(String, Timeframe)</u>	Returns the number of bars on the specified chart
<u>OrdersHistoryTotal()</u>	Returns the number of closed orders in the account history loaded into the terminal
<u>AccountNumber()</u>	Returns the number of the current account

<u>OrderSymbol()</u>	Returns the order symbol value for selected order
<u>WindowPriceOnDropped()</u>	Returns the price part of the chart point where expert or script was dropped
<u>iLowest(String, Timeframe, Series, Int32, Int32)</u>	Returns the shift of the least value over a specific number of periods depending on type
<u>iHighest(String, Timeframe, Series, Int32, Int32)</u>	Returns the shift of the maximum value over a specific number of periods depending on type
<u>WindowHandle(String, Timeframe)</u>	Returns the system window handler containing the given chart
<u>WindowTimeOnDropped()</u>	Returns the time part of the chart point where expert or script was dropped
<u>WindowXOnDropped()</u>	Returns the value at X axis in pixels for the chart window client area point at which the expert or script was dropped
<u>WindowYOnDropped()</u>	Returns the value at Y axis in pixels for the chart window client area point at which the expert or script was dropped
<u>GlobalVariableGet(String)</u>	Returns the value of an existing global variable or 0 if an error occurs
<u>AccountStopoutLevel()</u>	Returns the value of the Stop Out level
<u>iVolume(String, Timeframe, Int32)</u>	Returns Tick Volume value for the bar of indicated symbol with timeframe and shift
<u>OrderTicket()</u>	Returns ticket number for the currently selected order
<u>iTime(String, Timeframe, Int32)</u>	Returns Time value for the bar of indicated symbol with timeframe and shift

<u>ObjectsTotal(ObjectType)</u>	Returns total amount of objects of the specified type in the chart
<u>IsTradeContextBusy()</u>	Returns TRUE if a thread for trading is occupied by another expert advisor, otherwise returns FALSE
<u>IsExpertEnabled()</u>	Returns TRUE if expert advisors are enabled for running, otherwise returns FALSE
<u>IsOptimization()</u>	Returns TRUE if expert runs in the strategy tester optimization mode, otherwise returns FALSE
<u>IsTesting()</u>	Returns TRUE if expert runs in the testing mode, otherwise returns FALSE
<u>WindowIsVisible(Int32)</u>	Returns TRUE if the chart subwindow is visible, otherwise returns FALSE
<u>IsLibrariesAllowed()</u>	Returns TRUE if the expert can call library function, otherwise returns FALSE
<u>IsTradeAllowed()</u>	Returns TRUE if the expert is allowed to trade and a thread for trading is not occupied, otherwise returns FALSE
<u>IsVisualMode()</u>	Returns TRUE if the expert is tested with checked 'Visual Mode' button, otherwise returns FALSE
<u>IsDemo()</u>	Returns TRUE if the expert runs on a demo account, otherwise returns FALSE
<u>IsDllsAllowed()</u>	Returns TRUE if the function DLL call is allowed for the expert, otherwise returns FALSE
<u>GlobalVariableCheck(String)</u>	Returns TRUE if the global variable exists, otherwise, returns FALSE
<u>IsStopped()</u>	Returns TRUE if the program (an expert or a script) has been commanded to stop its operation, otherwise returns FALSE

<u>WindowOnDropped()</u>	Returns window index where expert, custom indicator or script was dropped
<u>WindowScreenShot(String, Int32, Int32, Int32, Int32, Int32)</u>	Saves current chart screen shot as a GIF file
<u>ObjectFind(String)</u>	Search for an object having the specified name
<u>iBarShift(String, Timeframe, DateTime, Boolean)</u>	Search for bar by open time
<u>GlobalVariableSet(String, Double)</u>	Sets a new value of the global variable
<u>GlobalVariableSetOnCondition(String, Double, Double)</u>	Sets the new value of the existing global variable if the current value equals to the third parameter check_value
<u>ObjectSetFiboDescription(String, Int32, String)</u>	The function assigns a new description to a level of a Fibonacci object
<u>ObjectGetShiftByValue(String, Double)</u>	The function calculates and returns bar index (shift related to the current bar) for the given price
<u>ObjectGetValueByShift(String, Int32)</u>	The function calculates and returns the price value for the specified bar (shift related to the current bar)
<u>ObjectMove(String, Int32, DateTime, Double)</u>	The function moves an object coordinate in the chart
<u>Comment(String)</u>	The function outputs the comment defined by the user in the left top corner of the chart
<u>WindowFirstVisibleBar()</u>	The function returns the first visible bar number in the current chart window
<u>GetLastError()</u>	The function returns the last occurred error, then the value of special last_error variable where the last error code is stored will be zeroized
<u>ObjectGetFiboDescription(String, Int32)</u>	The function returns the level description of a Fibonacci object

<u>GlobalVariableName(Int32)</u>	The function returns the name of a global variable by its index in the list of global variables
<u>ObjectName(Int32)</u>	The function returns the object name by its index in the objects list
<u>ObjectType(String)</u>	The function returns the object type value
<u>IsConnected()</u>	The function returns the status of the main connection between client terminal and server that performs data pumping
<u>GlobalVariablesTotal()</u>	The function returns the total count of global variables
<u>ObjectGet(String, ObjectProperty)</u>	The function returns the value of the specified object property
<u>OrderSelect(Int32, SelectionType, SelectionPool)</u>	The function selects an order for further processing
<u>HideTestIndicators(Boolean)</u>	The function sets a flag hiding indicators called by the Expert Advisor
<u>GetTickCount()</u>	The GetTickCount() function retrieves the number of milliseconds that have elapsed since the system was started
<u>OrderSend(String, TradeOperation, Double, Double, Int32, Double, Double, String, Int32, DateTime, Color)</u>	The main function used to open a position or place a pending order
<u>MessageBox(String, String, Int32)</u>	The MessageBox function creates, displays, and operates message box