Programming Documentation Version 1.0

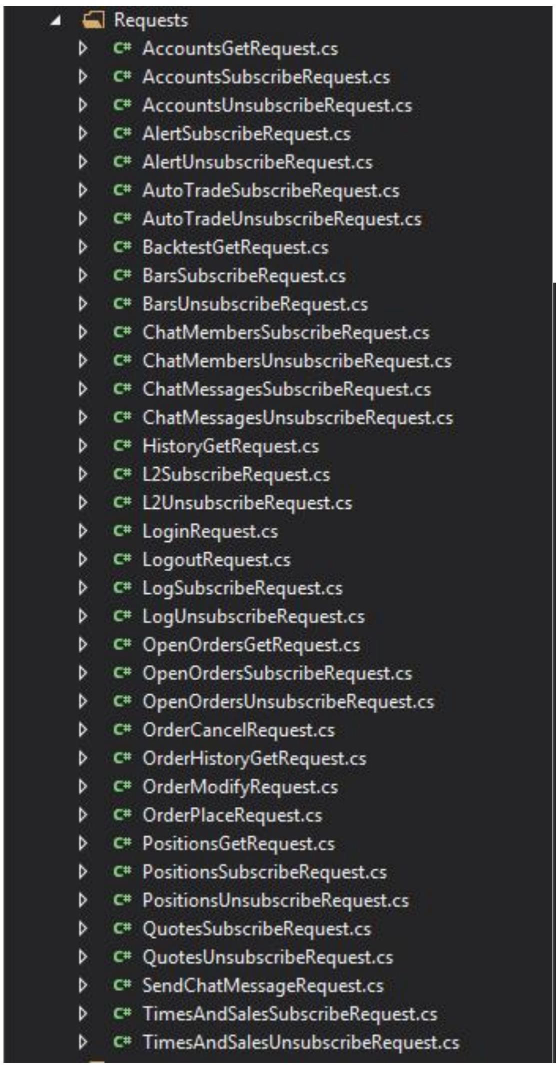
1. Server
2. Client

1. Server

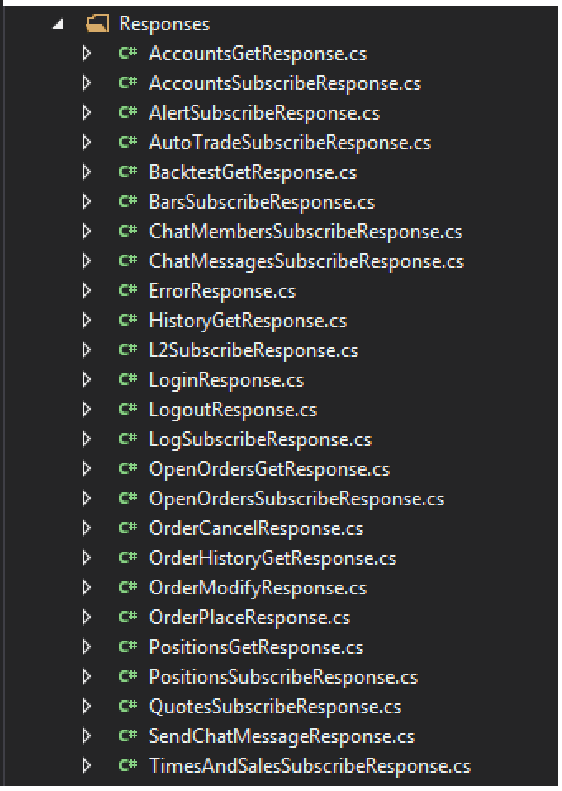
The server communicates with the M4 HTML5 web client and sends fake data for demonstration of how the client works.

The solution contains a project with several classes. Most of the classes are incoming (“Request”) and outgoing (“Response”) messages, which are the basis of the server. Also there are folders called “Types” and “Enums” with elements that are parts of messages between the client and server.

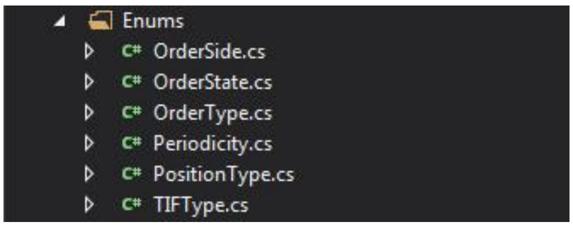
The project structure is shown below.



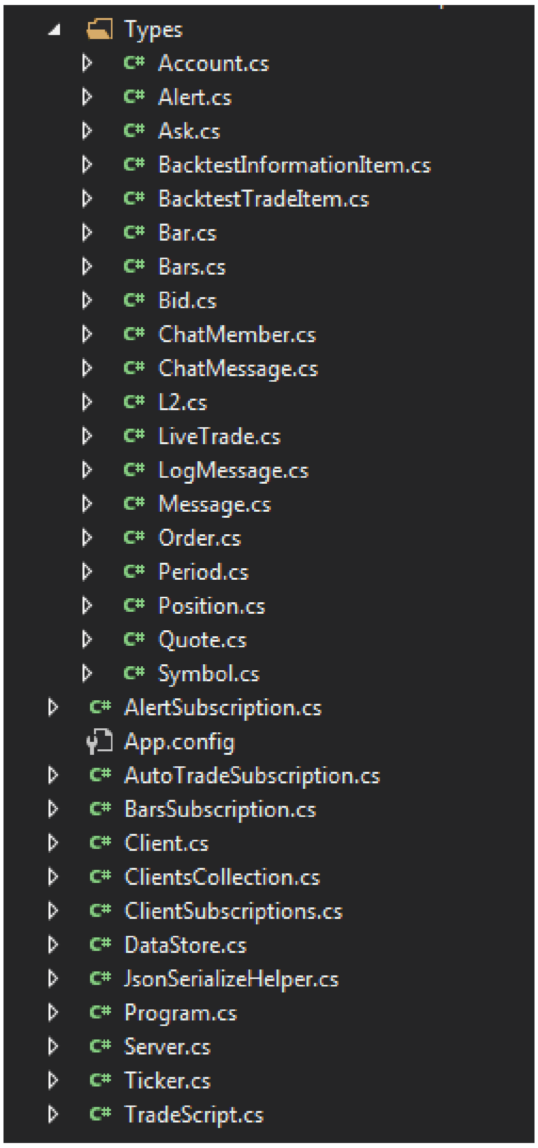
```
Requests
    AccountsGetRequest.cs
    AccountsSubscribeRequest.cs
    AccountsUnsubscribeRequest.cs
    AlertSubscribeRequest.cs
    AlertUnsubscribeRequest.cs
    AutoTradeSubscribeRequest.cs
    AutoTradeUnsubscribeRequest.cs
    BacktestGetRequest.cs
    BarsSubscribeRequest.cs
    BarsUnsubscribeRequest.cs
    ChatMembersSubscribeRequest.cs
    ChatMembersUnsubscribeRequest.cs
    ChatMessagesSubscribeRequest.cs
    ChatMessagesUnsubscribeRequest.cs
    HistoryGetRequest.cs
    L2SubscribeRequest.cs
    L2UnsubscribeRequest.cs
    LoginRequest.cs
    LogoutRequest.cs
    LogSubscribeRequest.cs
    LogUnsubscribeRequest.cs
    OpenOrdersGetRequest.cs
    OpenOrdersSubscribeRequest.cs
    OpenOrdersUnsubscribeRequest.cs
    OrderCancelRequest.cs
    OrderHistoryGetRequest.cs
    OrderModifyRequest.cs
    OrderPlaceRequest.cs
    PositionsGetRequest.cs
    PositionsSubscribeRequest.cs
    PositionsUnsubscribeRequest.cs
    QuotesSubscribeRequest.cs
    QuotesUnsubscribeRequest.cs
    SendChatMessageRequest.cs
    TimesAndSalesSubscribeRequest.cs
    TimesAndSalesUnsubscribeRequest.cs
```



```
Responses
    AccountsGetResponse.cs
    AccountsSubscribeResponse.cs
    AlertSubscribeResponse.cs
    AutoTradeSubscribeResponse.cs
    BacktestGetResponse.cs
    BarsSubscribeResponse.cs
    ChatMembersSubscribeResponse.cs
    ChatMessagesSubscribeResponse.cs
    ErrorResponse.cs
    HistoryGetResponse.cs
    L2SubscribeResponse.cs
    LoginResponse.cs
    LogoutResponse.cs
    LogSubscribeResponse.cs
    OpenOrdersGetResponse.cs
    OpenOrdersSubscribeResponse.cs
    OrderCancelResponse.cs
    OrderHistoryGetResponse.cs
    OrderModifyResponse.cs
    OrderPlaceResponse.cs
    PositionsGetResponse.cs
    PositionsSubscribeResponse.cs
    QuotesSubscribeResponse.cs
    SendChatMessageResponse.cs
    TimesAndSalesSubscribeResponse.cs
```



```
◢ Enums
    OrderSide.cs
    OrderState.cs
    OrderType.cs
    Periodicity.cs
    PositionType.cs
    TIFType.cs
```



```
◢ Types
    Account.cs
    Alert.cs
    Ask.cs
    BacktestInformationItem.cs
    BacktestTradeItem.cs
    Bar.cs
    Bars.cs
    Bid.cs
    ChatMember.cs
    ChatMessage.cs
    L2.cs
    LiveTrade.cs
    LogMessage.cs
    Message.cs
    Order.cs
    Period.cs
    Position.cs
    Quote.cs
    Symbol.cs
  AlertSubscription.cs
  App.config
  AutoTradeSubscription.cs
  BarsSubscription.cs
  Client.cs
  ClientsCollection.cs
  ClientSubscriptions.cs
  DataStore.cs
  JsonSerializeHelper.cs
  Program.cs
  Server.cs
  Ticker.cs
  TradeScript.cs
```

The server uses the SuperSocket library to establish connections with clients and the Modulus TradeScript library for Backtesting, Alerts and Autotrading.

There are several main classes that implement the server logic:

1. Server.cs
2. Ticker.cs
3. DataStore.cs
4. Message.cs (parent class for all requests and responses)
5. Client.cs with ClientSubscriptions.cs (user representation)

When the server starts, it runs Ticker which has timers that sends appropriate data to the client if subscribed. Server.cs has a list of connected clients and every client has its own subscription object. Ticker looks at the ClientSubscriptions of every Client and sends SubscribeResponse if need.

Also Server.cs has a large switch for processing every incoming message. If an incoming message is a get request, the Server processes this message and sends an appropriate response. If an incoming message is a subscribe request, it switches to the appropriate flag ON in ClientSubscriptions for the current client.

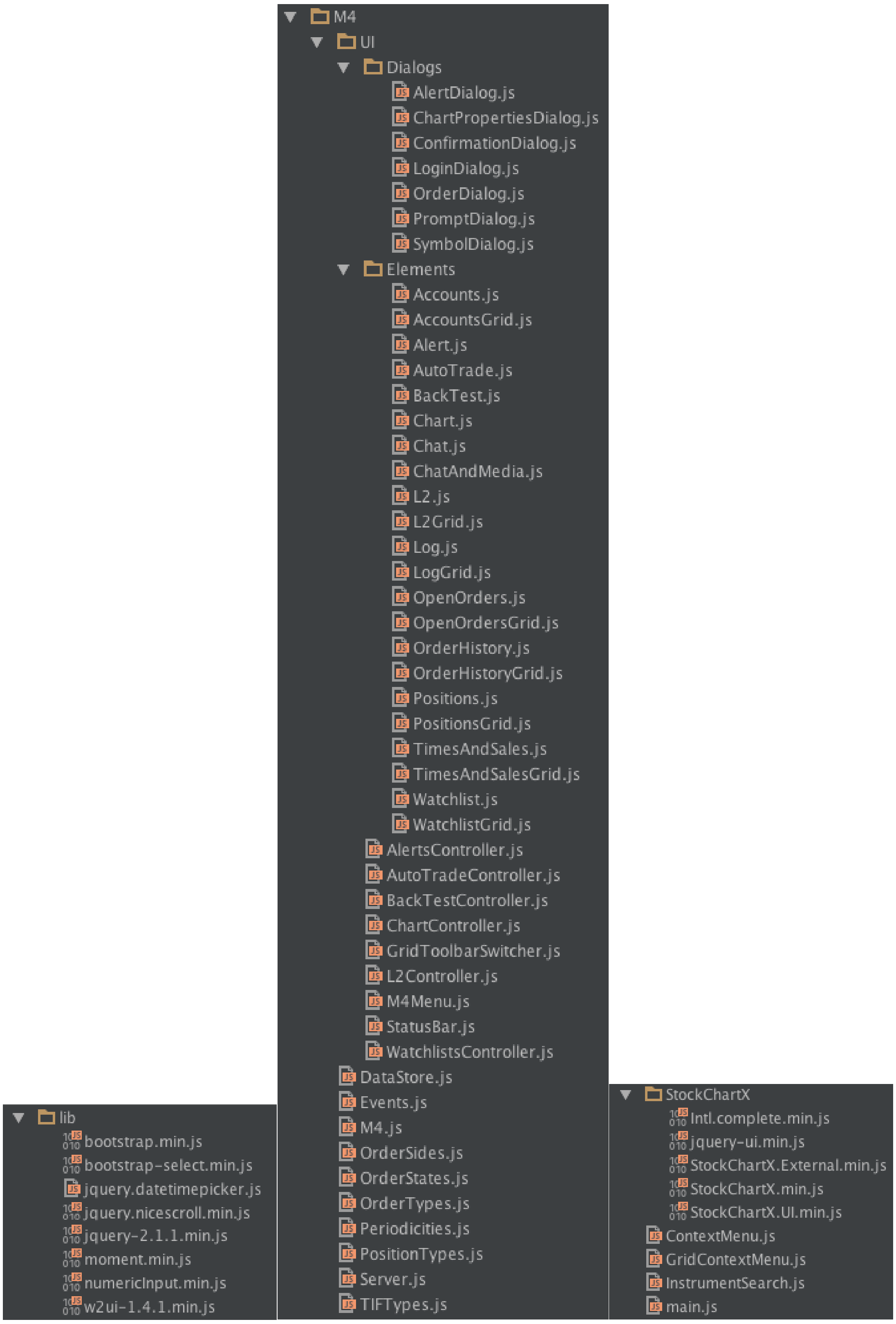
The DataStore class contains static data and also it can generate random values for quotes, bars, times & sales and other types of data.

The System class has only one user and all clients that connect to the server will see the same data in their views.

To implement a data feed on the server, you have to rewrite the method OnMessageReceived in Server.cs according to your needs. By the way you may remove the classes Ticker, DataStore and others that are related to generating random data.

2. Client

The client application logic can be found in all *.js files. A full tree of those files is shown below.



```
▼ M4
    ▼ UI
        ▼ Dialogs
                AlertDialog.js
                ChartPropertiesDialog.js
                ConfirmationDialog.js
                LoginDialog.js
                OrderDialog.js
                PromptDialog.js
                SymbolDialog.js
        ▼ Elements
                Accounts.js
                AccountsGrid.js
                Alert.js
                AutoTrade.js
                BackTest.js
                Chart.js
                Chat.js
                ChatAndMedia.js
                L2.js
                L2Grid.js
                Log.js
                LogGrid.js
                OpenOrders.js
                OpenOrdersGrid.js
                OrderHistory.js
                OrderHistoryGrid.js
                Positions.js
                PositionsGrid.js
                TimesAndSales.js
                TimesAndSalesGrid.js
                Watchlist.js
                WatchlistGrid.js
            AlertsController.js
            AutoTradeController.js
            BackTestController.js
            ChartController.js
            GridToolbarSwitcher.js
            L2Controller.js
            M4Menu.js
            StatusBar.js
            WatchlistsController.js
        DataStore.js
        Events.js
        M4.js
        OrderSides.js
        OrderStates.js
        OrderTypes.js
        Periodicities.js
        PositionTypes.js
        Server.js
        TIFTypes.js
```

```
▼ lib
        bootstrap.min.js
        bootstrap-select.min.js
        jquery.datetimepicker.js
        jquery.nicescroll.min.js
        jquery-2.1.1.min.js
        moment.min.js
        numericInput.min.js
        w2ui-1.4.1.min.js
```

```
▼ StockChartX
        Intl.complete.min.js
        jquery-ui.min.js
        StockChartX.External.min.js
        StockChartX.min.js
        StockChartX.UI.min.js
    ContextMenu.js
    GridContextMenu.js
    InstrumentSearch.js
    main.js
```

The application starts in main.js after a page has been loaded. By the way most of the dialogs initialize before that time and they are located in global variables with appropriate names.

It should be noted that the file StockChartX.External.min.js includes the following required libraries:

- jquery.gwfselect.js
- numericInput.js
- switchery.js
- bootstrap-select.js
- jquery.autocolumnlist.js
- spectrum.js

After the page loads, a login dialog initializes and if the user login is correct, the main application view with all docks and tabs initializes itself.

All grids are based on the popular w2ui grid control. Most of styles are provided by bootstrap.

There are also several “enums” for defining order types, states, etc. They also have methods for converting their type to a string and vice versa:

- OrderSides.js
- OrderStates.js
- OrderTypes.js
- Periodicities.js
- PositionTypes.js
- TIFTypes.js

Tab scripts are located in the folder “Elements” and every item has its own grid customization. The tabs which can be opened in any quantity have an associated *Controller.js.

Also there is Server.js that provides connection to the server, sends and receives data, knows how convert it to build requests, and so on.

DataStore.js delivers data in tables and windows. It has get-type and subscribe-type methods and gets data from Server.js. Also it accumulates some data in itself to reduce the number of requests to the server.

M4.js is called after the user login has succeeded. For the first time, it initializes docks, then all view elements (tabs), menu, and status bar. Server and DataStore are initialized before that moment and are ready to use by their associated tabs.

Thanks for reading. More coming soon!