



METATRADER 4 MANAGER API

Contents

Introduction	4
Installation	4
Uses.....	5
GUI	8
Main Tab	9
Connect.....	13
Login.....	14
Disconnect.....	16
Groups.....	18
Mail Send	21
Send News.....	23
Pumping	25
Symbols Tab	27
User Tab	30
Online Tab	33
Orders Tab.....	36
Request	36
User History	39
Journal Tab.....	42
Market Watch	45
Summary	49
Exposure.....	52
Margin.....	55
News	58
Mail Box	61
Plugins.....	64
Daily Reports.....	67
Others	71

Server Feed	71
Server Time	74
Ping	75
Server Restart	76
Dependencies Graph.....	77
Other methods.....	78
Connection and authorization	78
Manager interface	78
administrator functions	78
database backups	79
Symbols	79
Direct access to the server databases	79
References	81

Introduction

MetaTrader 4 Manager API is a library in form of a DLL file containing the complete set of administrator and manager commands to access to MetaTrader Server. It's a .NET Wrapper around native mtmanapi.dll/ mtmanapi64.dll, for MetaTrader 4 manager API.

Installation

- Install Visual C++

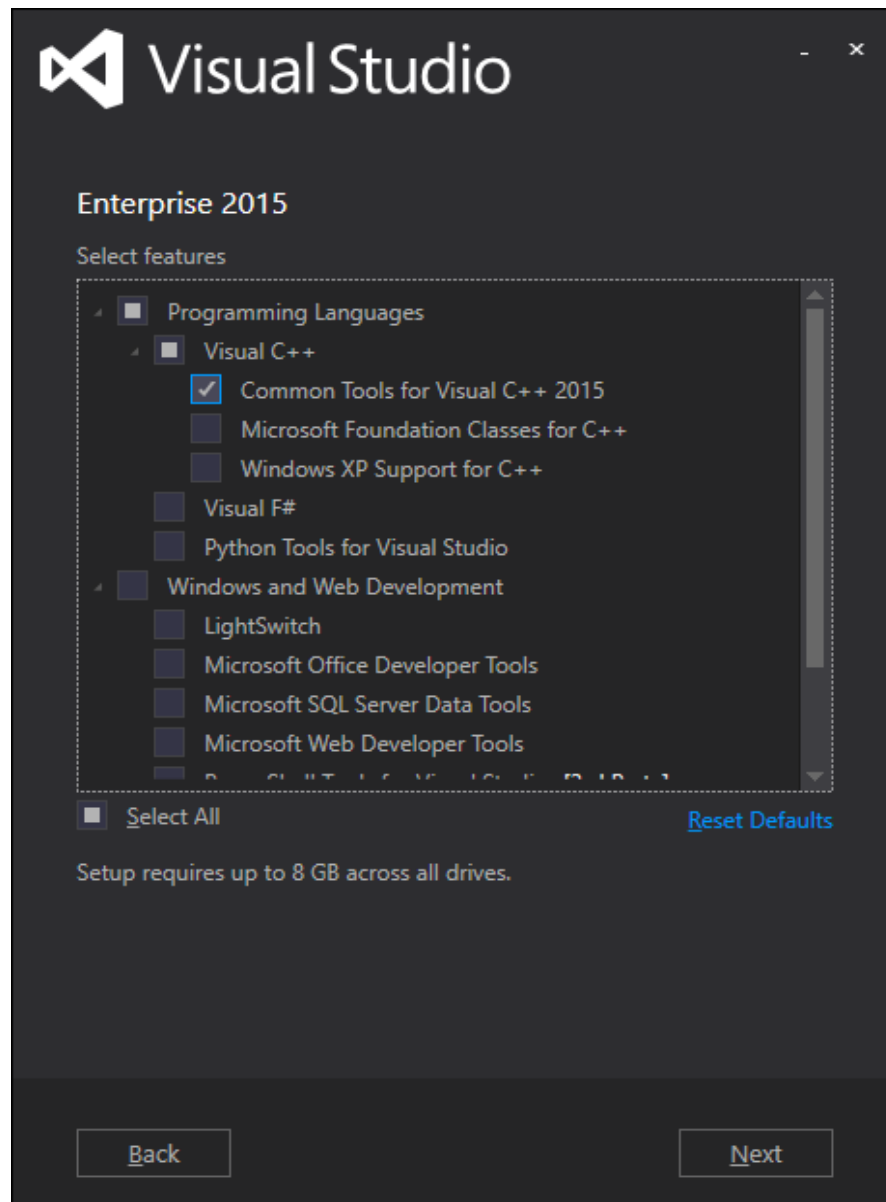


Fig: Installation of Visual C++

- To use these providers we will need to install the MetaTrader4.Manager.Wrapper NuGet package:
`Install-Package MetaTrader4.Manager.Wrapper`

Uses

Include the library

```
using P23.MetaTrader4.Manager;
using P23.MetaTrader4.Manager.Contracts;
```

Common Language Runtime(CLR) wrapper

```
/// <summary>
/// Create an object of ClrWrapper
/// Wrapper around mtmanapi.dll to
/// provide managed access to MT4 manager API
/// </summary>
public ClrWrapper clrWrapper = new ClrWrapper();

clrWrapper.Connect("175.41.246.194:443");
int n = clrWrapper.IsConnected();
Console.WriteLine("Connection : "+clrWrapper.ErrorDescription(n));

int l = clrWrapper.Login(0000, "atomap");
Console.WriteLine("Login : " + clrWrapper.ErrorDescription(l));
```

`Connect()` function that accepts the server address formatted as 'server:port'. If there is no port specified, port 443 will be used by default.

`IsConnected()` function allows to check the connection status. The `ErrorDescription()` function returns a text description of the return value of `IsConnected()` method.

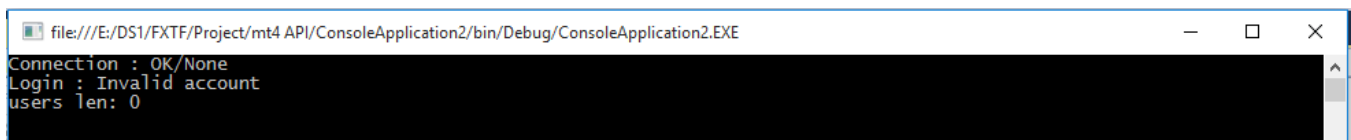


Fig: the connection is established but can't login to the server due to the login/password invalid.

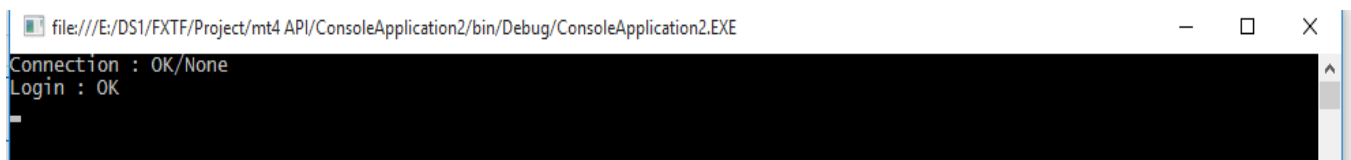


Fig: After correct login, password and server credentials.

The list of all accounts can be requested by the `UsersRequest()` function.

```
IList<UserRecord> users= clrWrapper.UsersRequest();
Console.WriteLine("users len: "+ users.Count);
foreach (var p in users)
{
    Console.WriteLine(p);
}
```

Using wrapper in Pumping mode

Pumping is a saving and quick mode of uploading data from the server. When transferring to pumping mode, manager interface requests the server for symbol settings, groups, account databases, orders, and trading requests, after that the server sends only updated data to the connected manager.

After having connected to the server, the manager interface is transferred to the pumping mode with the `PumpingSwitch()` function. The pointer to the callback function to be used by the manager interface to notify about the data updating, or window handle and identifier of the user message to which the notification about the data updating will be sent, must be passed as a parameter of this function

Pumping mode allows to get notifications about changes regarding users, trades and etc. Data can be accessed using `*Get` methods.

```
var are = new AutoResetEvent(false); // using System.Threading;
clrWrapper.PumpingSwitch(i =>
{
    if (i == 0) // 0 - means pumping started
        are.Set();
});

are.WaitOne();

IList<UserRecord> usersRecords = clrWrapper.UsersGet();
```

Here, `PUMP_UPDATE_USERS` — the account list has been updated, the updated account list can be obtained with functions named `UsersGet()` or `UserRecordGet()`;

Extended pumping

Besides standard pumping mode, in which only notifications about data change come, there is a special mode with replaying of coming transactions. In this mode there is a possibility to receive information about changes in client and order records, as well as changes in group and symbol settings.

For switching to the extended pumping mode, the PumpingSwitchEx method is used:

```
var are = new AutoResetEvent(false); // using System.Threading;

clrWrapper.PumpingSwitchEx(PumpingMode.Default);
clrWrapper.TradeAdded += (sender, record) =>
{
    Console.WriteLine(record);
};

clrWrapper.PumpingStarted += (sender, eventArgs) =>
{
    are.Set();
};
are.WaitOne();
```

GUI

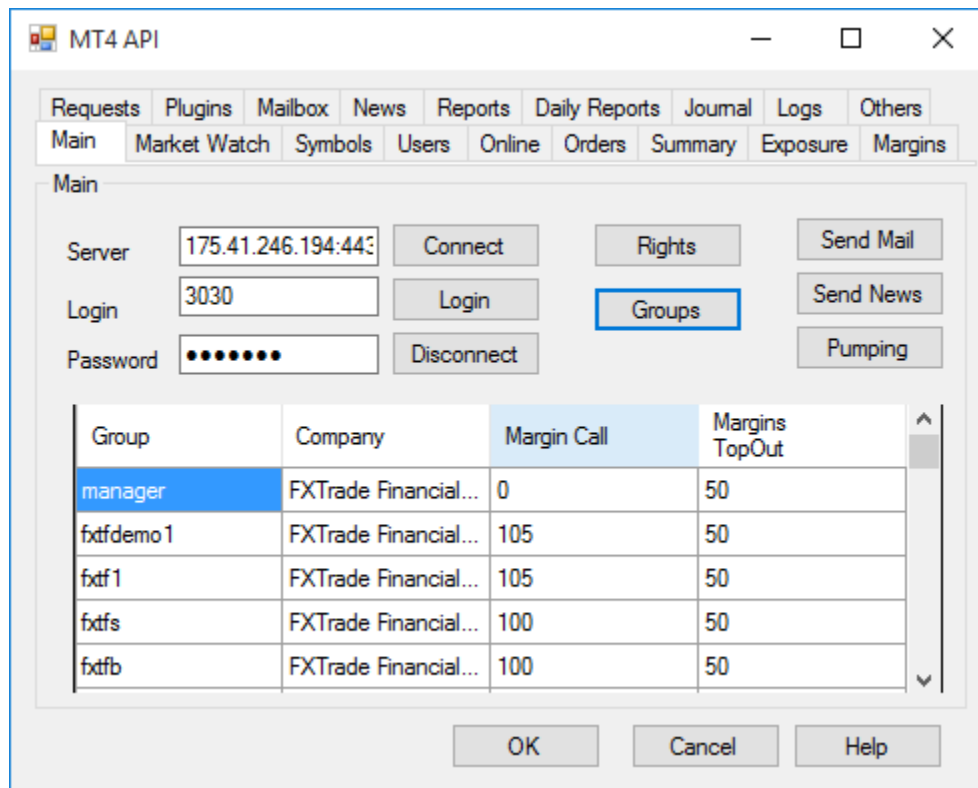


Fig: GUI Screen of the MetaTrader 4 API in C#

- Create a Win Form project
- To use these providers we will need to install the MetaTrader4.Manager.Wrapper NuGet package:

`Install-Package MetaTrader4.Manager.Wrapper`

- Build the gui as the above figure.

Main Tab

Initialization

```
/// <summary>
/// Initially define the number
/// of tab pages size
/// </summary>
public TabPage[] newPage = new TabPage[100];
/// <summary>
/// Define the tab pages name
/// and in the runtime create the tab control
/// tab named here
/// </summary>
public string[] tabPageStrings={ "Main", "Market Watch", "Symbols", "Users",
"Online", "Orders", "Summary", "Exposure", "Margins", "Requests", "Plugins",
/*"Dealer",*/ "Mailbox", "News", "Reports", "Daily Reports", "Journal", "Logs", "Others"
};

/// <summary>
/// Define the Group Box name
/// and in the runtime show the appropriate
/// Group box and hide the others
/// </summary>
public string[] groupBoxStrings = { "Main", "MarketWatch", "Symbols", "Users",
"Online", "Orders", "Summary", "Exposure", "Margins", "Requests", "Plugins", "Dealer",
"Mailbox", "News", "Reports", "DailyReports", "Journal", "Logs", "Others" };

/// <summary>
/// Create an object of ClrWrapper
/// Wrapper around mtmanapi.dll to
/// provide managed access to MT4 manager API
/// </summary>
public ClrWrapper clrWrapper = new ClrWrapper();
/// <summary>
/// Switch flag to check
/// pumping switch
/// </summary>
public bool switchFlag = false;

/// <summary>
/// log in flag
/// </summary>
public bool isLoggedIn = false;
/// <summary>
/// Display message in case of no connection or logged out.
/// </summary>
public const string CONNECT_FIRST = "Please, connect first.";
```

In form constructor,

```
/// <summary>
/// Constructor
/// initialize the component
/// </summary>
public Form1()
{
    InitializeComponent();
    int i = 0;

    foreach(var pages in tabPageStrings)
    {

        newPage[i] = new TabPage(pages);
        tabControl1.TabPages.Add(newPage[i]);
        newPage[i].AutoScroll = true;
        i++;
    }
    ShowCurrentPanel("Main");
    tabControl1.SelectedIndexChanged += tabControl1_SelectedIndexChanged;
}
```

In the form load,

```
/// <summary>
/// Initialize in form load
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Form1_Load(object sender, EventArgs e)
{

    FromDateTimePicker_DailyReports.Value=DateTime.Today.AddDays(-365);
    dateTimePickerFrom.Value= DateTime.Today.AddDays(-30);
    dateTimePicker_From.Value = DateTime.Today.AddDays(-30);

    switchFlag = false;
    textBox1_server.Text = "xxx.xxx.xxx.xxx:443";
    textBox2_login.Text = "*****";
    textBox3_password.Text = "*****";

    List<JournalFilter> journalFilterList = new List<JournalFilter>();
    journalFilterList.Add(new JournalFilter { FilterName = "Standard" });
    journalFilterList.Add(new JournalFilter { FilterName = "Logins" });
    journalFilterList.Add(new JournalFilter { FilterName = "Trades" });
    journalFilterList.Add(new JournalFilter { FilterName = "Errors" });
    journalFilterList.Add(new JournalFilter { FilterName = "Full" });

    comboBox1_Journal.DataSource = journalFilterList;
    comboBox1_Journal.DisplayMember = "FilterName";
    comboBox1_Journal.ValueMember = "FilterName";
}
```

And the `JournalFilter` class is:

```
/// <summary>
/// stores Journal Filter name
/// </summary>
class JournalFilter
{
    /// <summary>
    /// Store for the name property
    /// </summary>
    public string FilterName { get; set; }

}
```

Event Handler for tab control's selected tab index changed,

```
/// <summary>
/// Event Handler for tab control's selected tab index changed
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void tabControl1_SelectedIndexChanged(Object sender, EventArgs e)
{
    MarketWatchTimeEnable = false;
    string tabNow = tabControl1.SelectedTab.Text;
    ShowCurrentPanel(tabNow);
}
```

Exit/Cancel the application,

```
/// <summary>
/// Exit the Application
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void OkButton_MainClick(object sender, EventArgs e)
{
    Application.Exit();
}

/// <summary>
/// Exit the Application
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CancelButton_MainClick(object sender, EventArgs e)
{
    Application.Exit();
}
```

Display the selected tab and hides others tab.

```
/// <summary>
/// Show the selected tab GroupBox
/// Hide other Group Box
/// <param name="_tabNow">Text</param>
/// </summary>
private void ShowCurrentPanel(string _tabNow)
{
    _tabNow = _tabNow.Replace(" ", String.Empty);
    foreach (var pages in groupBoxStrings)
    {
        try
        {
            (Controls[pages] as GroupBox).Visible = false;
        }
        catch (Exception ee1)
        {
        }
    }

    try
    {
        (Controls[_tabNow] as GroupBox).Visible = true;
        (Controls[_tabNow] as GroupBox).Location = new System.Drawing.Point(12,
50);
    }
    catch (Exception ee1)
    {
    }
}
```

Connect

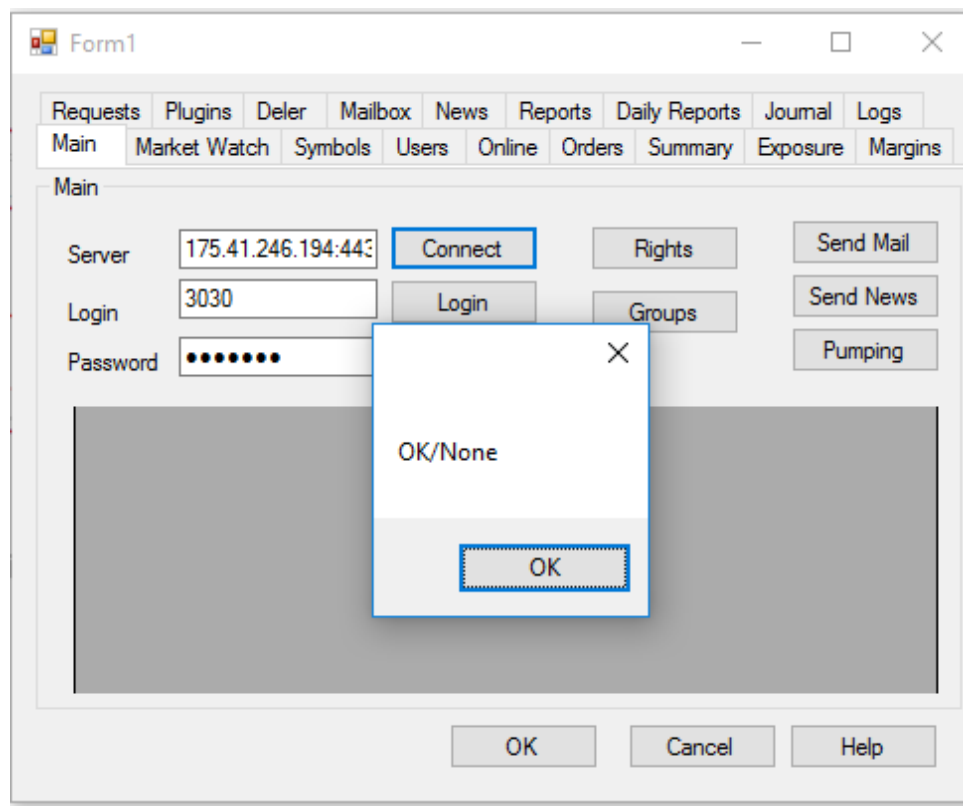


Fig: Successful Connect

The "Connect" button is use for connecting to the MT4 server.

```
/// <summary>
/// Connect To the server
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ConnectButton_MainClick(object sender, EventArgs e)
{
    ConnectToServer();
    int n = clrWrapper.IsConnected();
    MessageBox.Show(clrWrapper.ErrorDescription(n));
}
/// <summary>
/// Connect To the Server
/// </summary>
public void ConnectToServer()
{
    string connectText = textBox1_server.Text;
    clrWrapper.Connect(connectText);
}
```

After Connect, there will be confirmation about successful or unsuccessful Connect.

Login

The "login" button is use for connecting to the MT4 server.

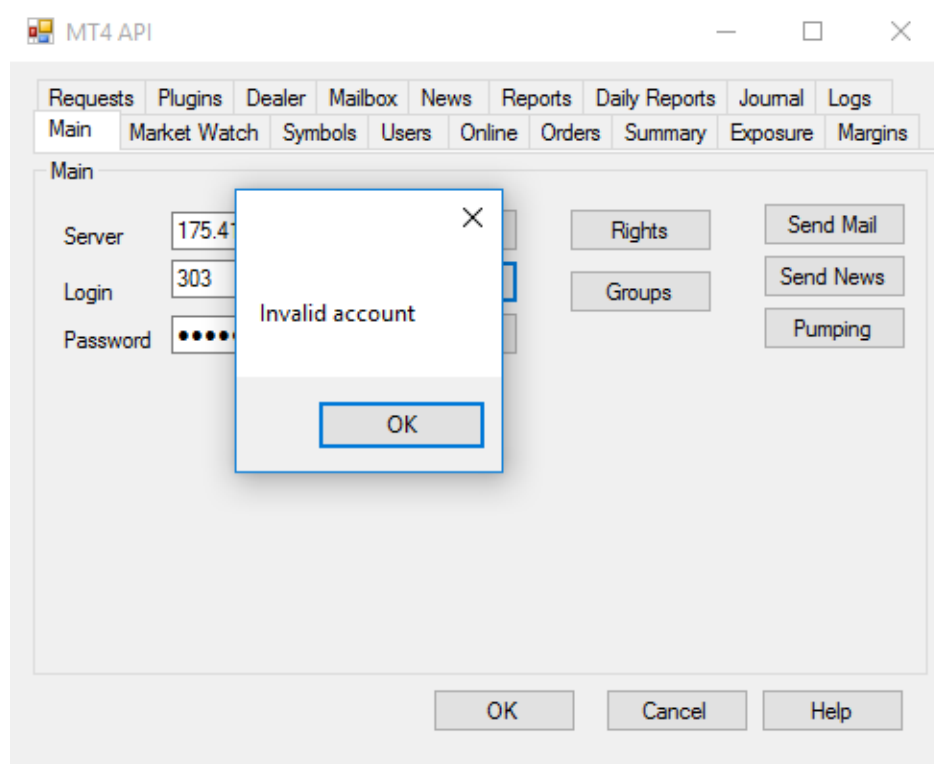


Fig: Unsuccessful Login

```
/// <summary>
/// Login to the server method
/// </summary>
/// <returns>return -1 if can't login to the server </returns>
private int LogIn()
{
    switchFlag = false;
    string login = textBox2_login.Text;
    string pass = textBox3_password.Text;

    try
    {
        int log = Int32.Parse(login);
        ConnectToServer();

        int l = clrWrapper.Login(log, pass);
        return l;
    }
    catch (Exception e3)
    {
        return -1;
    }
}
/// <summary>
```

```

/// Login to the server
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void LoginButton_MainClick(object sender, EventArgs e)
{
    int isSuccess=LogIn();
    if (isSuccess!=-1)
    {
        MessageBox.Show(clrWrapper.ErrorDescription(isSuccess));
    }
    else
    {
        string msg = "Can't parse the login/username";
        MessageBox.Show(msg);
    }
}

```

After Connect, there will be confirmation about successful or unsuccessful login.

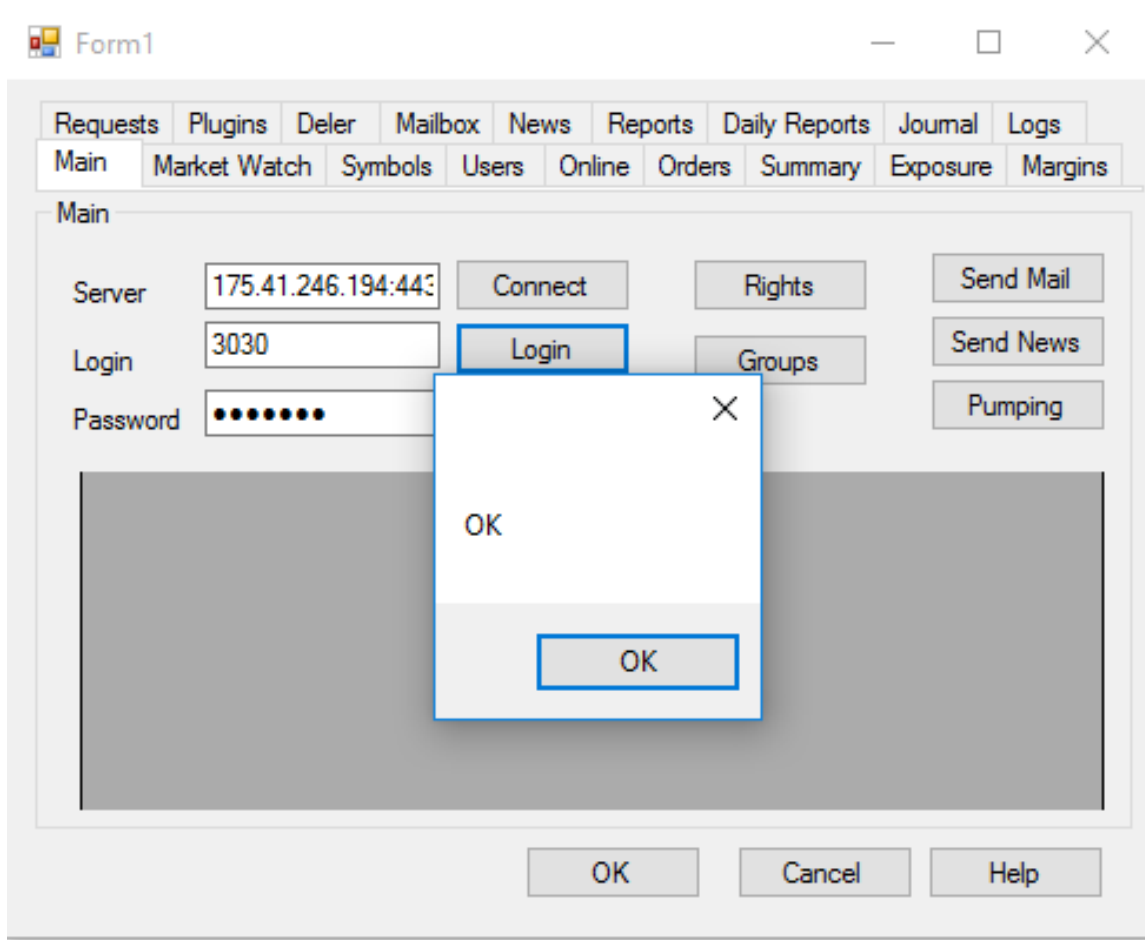


Fig: Successful login

Disconnect

The “disconnect” button is use for connecting to the MT4 server.

```
/// <summary>
/// Disconnect from the server
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void DisconnectButton_MainClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        clrWrapper.Disconnect();
        string msg = "Disconnect from the server.";
        MessageBox.Show(msg);
    }
    else
    {
        string msg = "Not connected to the server.";
        MessageBox.Show(msg);
    }
}
```

After disconnect, there will be confirmation about successful or unsuccessful disconnect.

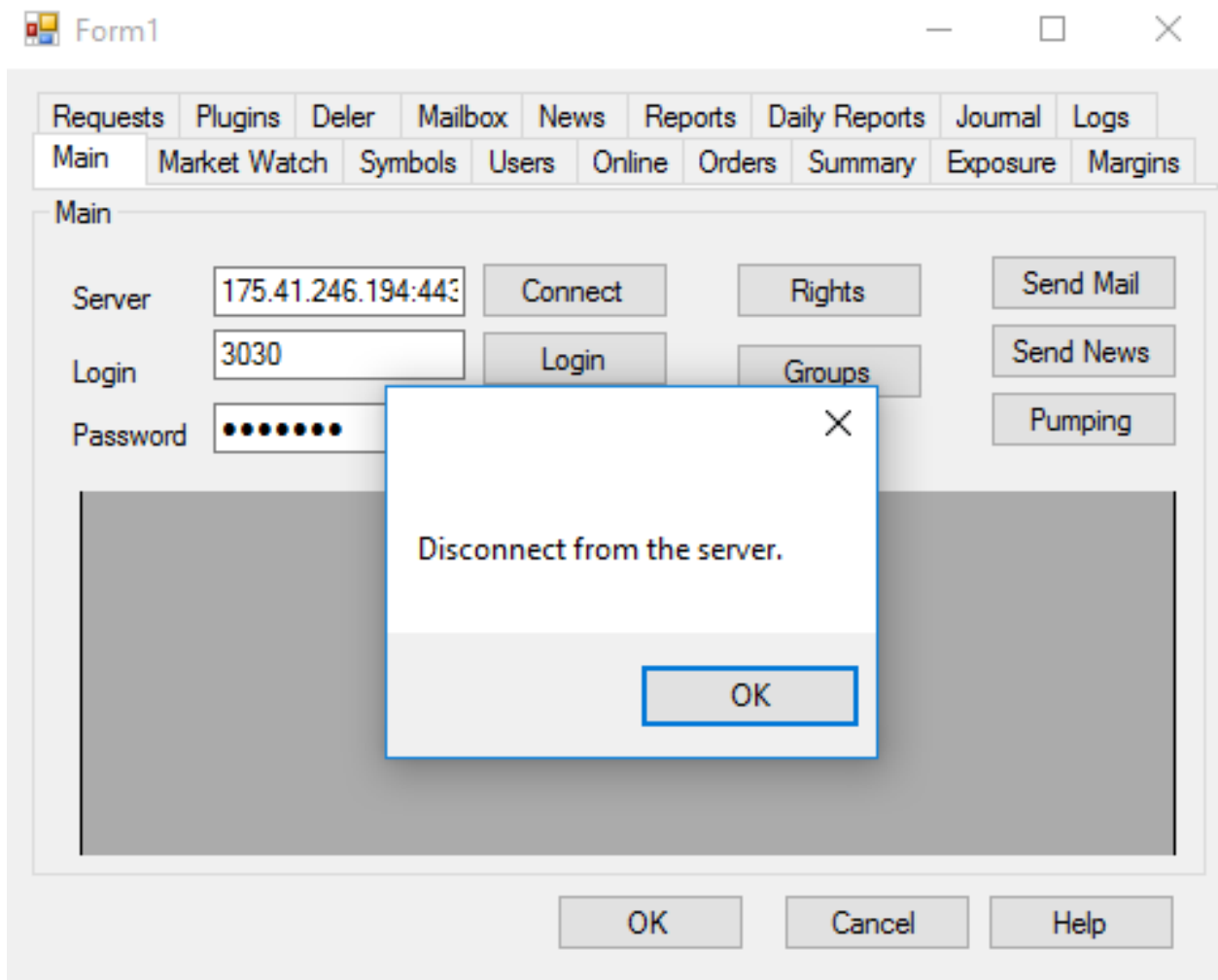


Fig: successfully disconnect from the server.

Groups

The "Groups" button is use for Request a list of available groups of accounts. Here Group means Object that represents group configuration.

```
/// <summary>
/// Request a list of available groups of accounts
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>

private void GroupsButton_MainClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        LogIn();
        IList<Group> users = clrWrapper.GroupsRequest();
        IList<GroupListFxtf> list = new List<GroupListFxtf>();

        foreach (var p in users)
        {
            string group = p.Name;
            string company = p.Company;
            int margincall = p.MarginCall;
            int marginstopOut = p.MarginStopout;
            list.Add(new GroupListFxtf(group, company, margincall,
marginstopOut));
        }

        dataGridView_groupsMain.Visible = true;
        dataGridView_groupsMain.DataSource = list;
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridView_groupsMain.DataSource = null;
    }
}
```

And the `GroupListFxtf` class is:

```
/// <summary>
/// Available groups of accounts
/// Object that represents group configuration
/// </summary>
public class GroupListFxtf
{
    /// <summary>
    /// Group name
    /// </summary>
    [DisplayName("Group")]
    public string Group { get; set; }

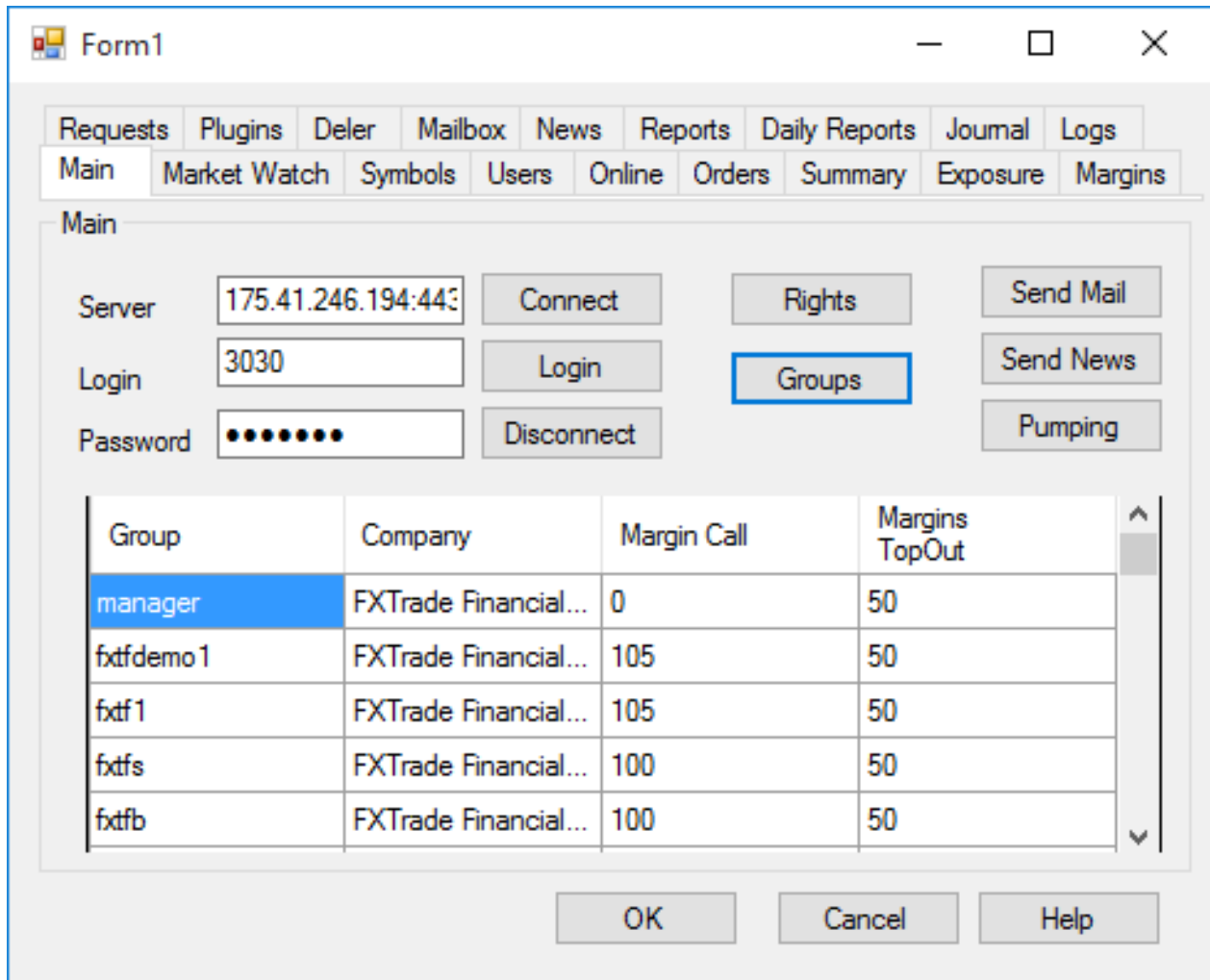
    /// <summary>
    /// Company name
    /// </summary>
    [DisplayName("Company")]
    public string Company { get; set; }

    /// <summary>
    /// Margin call level (percent's)
    /// </summary>
    [DisplayName("Margin Call")]
    public int Margincall { get; set; }

    /// <summary>
    /// Stop out level
    /// </summary>
    [DisplayName("Margins TopOut")]
    public int MarginstopOut { get; set; }

    /// <summary>
    /// Groups of accounts
    /// </summary>
    /// <param name="_group">Text</param>
    /// <param name="_company">Text</param>
    /// <param name="_margincall">Number</param>
    /// <param name="_marginstopOut">Number</param>
    public GroupListFxtf(string _group, string _company, int _margincall, int
_marginstopOut)
    {
        Group = _group;
        Company = _company;
        Margincall = _margincall;
        MarginstopOut = _marginstopOut;
    }
}
```

After Groups button click, there will be list of available groups of accounts. For demonstration we pick only four columns.



The screenshot shows a software window titled "Form1" with a menu bar and a main panel. The menu bar includes: Requests, Plugins, Deler, Mailbox, News, Reports, Daily Reports, Journal, Logs, Main, Market Watch, Symbols, Users, Online, Orders, Summary, Exposure, and Margins. The "Main" menu is selected. The main panel has a "Main" section with input fields for Server (175.41.246.194:443), Login (3030), and Password (masked with dots). There are buttons for Connect, Login, Disconnect, Rights, Groups (highlighted with a blue border), Send Mail, Send News, and Pumping. Below the input fields is a table with four columns: Group, Company, Margin Call, and Margins TopOut. The table contains five rows of data. The first row is highlighted in blue.

Group	Company	Margin Call	Margins TopOut
manager	FXTrade Financial...	0	50
fxtdemo1	FXTrade Financial...	105	50
fxtf1	FXTrade Financial...	105	50
fxtf5	FXTrade Financial...	100	50
fxtfb	FXTrade Financial...	100	50

At the bottom of the window are buttons for OK, Cancel, and Help.

Fig: List of available groups of accounts

Mail Send

To send mail, click in the send mail buttons.

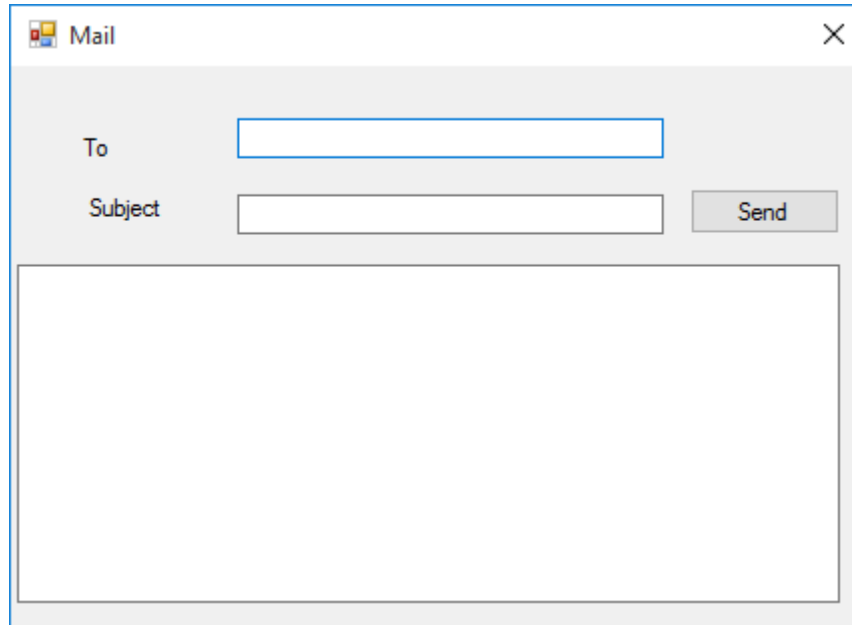


Fig: Mail Send

```
/// <summary>
/// Mail Send
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SendMailButton_MainClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        MailSend mail= new MailSend(clrWrapper);
        mail.Show();
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
    }
}

/// <summary>
/// Mail Send
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button1_SendMail_Click(object sender, EventArgs e)
{
    MailBox mailBox = new MailBox();
}
```

```
mailBox.Body = BodyTextBox_SendMail.ToString();
mailBox.Subject = textBox2_Subject.ToString();

mailBox.To =int.Parse(textBox1_T0.ToString());

IList<int> logins = new List<int>();

int status = clrWrapper.MailSend(mailBox, logins);
MessageBox.Show(clrWrapper.ErrorDescription(status));

}
```

`mailBox` is an Object that represents `MailBox` and send the email by using `MailSend()` method.

Send News

To send mail, click in the send mail buttons.

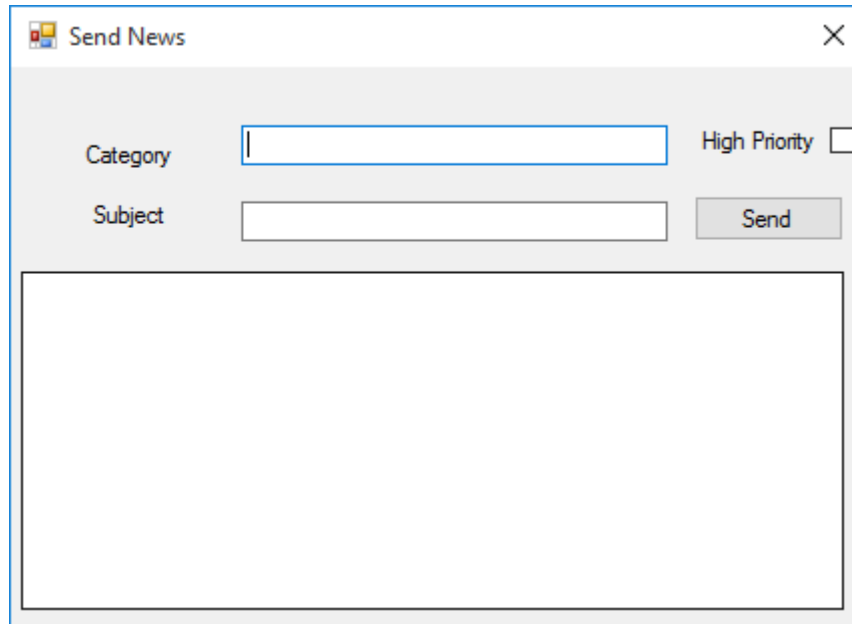


Fig: Send News

```
/// <summary>
/// Send News
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SendNewsButton_MainClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        NewsSend newsForm = new NewsSend(clrWrapper);
        newsForm.Show();
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
    }
}
```

```
/// <summary>
/// Publish news
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SendButton_SendMailClick(object sender, EventArgs e)
{
```

```
string category = CategoryTextBox_NewsSend.Text;
string subject = SubjectTextBox_NewsSend.Text;
bool priority = HighPriorityCheckBox.Checked;
string body = BodyTextBox_SendNews.Text;

NewsTopic news = new NewsTopic();
news.Category = category;
news.Topic = subject;
news.Body = body;

int status=clrWrapper.NewsSend(news);
MessageBox.Show(clrWrapper.ErrorDescription(status));

}
```

`NewsTopic` is an Object that represents news topic.

Pumping

To Enable Pumping Switch.

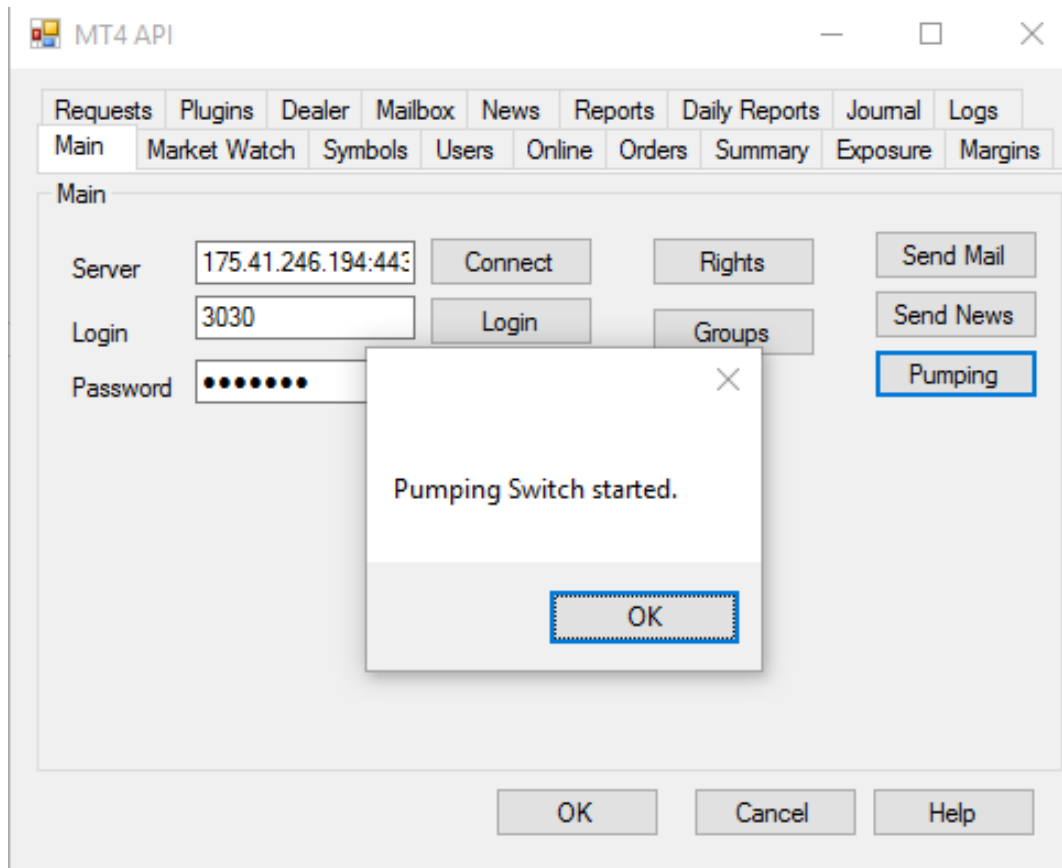


Fig: pumping enabled.

```

    /// <summary>
    /// Enable pumping switch mode
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void PumpingButton_MainClick(object sender, EventArgs e)
    {
        if (clrWrapper.IsConnected() != 0)
        {
            PumpingSwitch();
            string msg = "Pumping Switch started.";
            MessageBox.Show(msg);
        }
        else
        {
            MessageBox.Show(CONNECT_FIRST);
        }
    }

    /// <summary>
    /// Switch into pumping mode
    /// delegate will be invoked on any pumping event
    /// </summary>
    private void PumpingSwitch()
    {
        if (switchFlag == false)
        {
            var are = new AutoResetEvent(false);
            clrWrapper.PumpingSwitch(i =>
            {
                if (i == 0) // 0 - means pumping started
                    are.Set();
            });

            are.WaitOne();
            switchFlag = true;
        }
    }
}

```

Symbols Tab

This tab is used for Download symbols configuration from trade server.

Symbol	Description	Stops Level	Spread
CLIENTPL		0	0
CLIENTVOL		0	0
SUM.P		0	0
SUM.N		0	0
FIX.P		0	0
TOTAL.P		0	0
TOTAL.N		0	0
OPTIONUPPP	USD Options posi...	0	1
OPTIONDOWN	USD Options nega	0	1

Fig: Symbol List

```
/// <summary>
/// SymbolsRefresh() Download symbols configuration from trade server
/// SymbolsRefresh() should be called before this method
/// SymbolsGetAll() Get symbols configuration.
///
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RefreshButton_SymbolsClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        clrWrapper.SymbolsRefresh();
        IList<Symbol> symbols = clrWrapper.SymbolsGetAll();

        IList<SymbolListFxtf> list = new List<SymbolListFxtf>();

        foreach (var p in symbols)
        {
            string symbol = p.Name;
            string description = p.Description;
            int stopsLevel = p.StopsLevel;
        }
    }
}
```

```

        int spread = p.Spread;

        double bidTickValue = p.BidTickValue;
        double askTickValue = p.AskTickValue;
        list.Add(new SymbolListFxtf(symbol, description, stopsLevel, spread,
bidTickValue, askTickValue));

    }
    dataGridView_Symbol.Visible = true;
    dataGridView_Symbol.DataSource = list;
    LogIn();
}
else
{
    MessageBox.Show(CONNECT_FIRST);
    dataGridView_groupsMain.DataSource = null;
}
}
}

```

And the `SymbolListFxtf` is:

```

    /// <summary>
    /// Object that represents symbol configuration
    ///
    /// </summary>
    public class SymbolListFxtf
    {
        /// <summary>
        /// Name
        ///
        /// </summary>
        [DisplayName("Symbol")]
        public string Symbol { get; set; }

        /// <summary>
        /// Description
        ///
        /// </summary>
        [DisplayName("Description")]
        public string Description { get; set; }

        /// <summary>
        /// Stops Level
        ///
        /// </summary>
        [DisplayName("Stops Level")]
        public int StopsLevel { get; set; }

        /// <summary>
        /// Spread
        ///
        /// </summary>
        [DisplayName("Spread")]
    }

```

```

        public int Spread { get; set; }

        /// <summary>
        /// Tick value for bid
        ///
        /// </summary>
        public double BidTickValue { get; set; }
        /// <summary>
        /// Tick value for ask
        ///
        /// </summary>
        public double AskTickValue { get; set; }

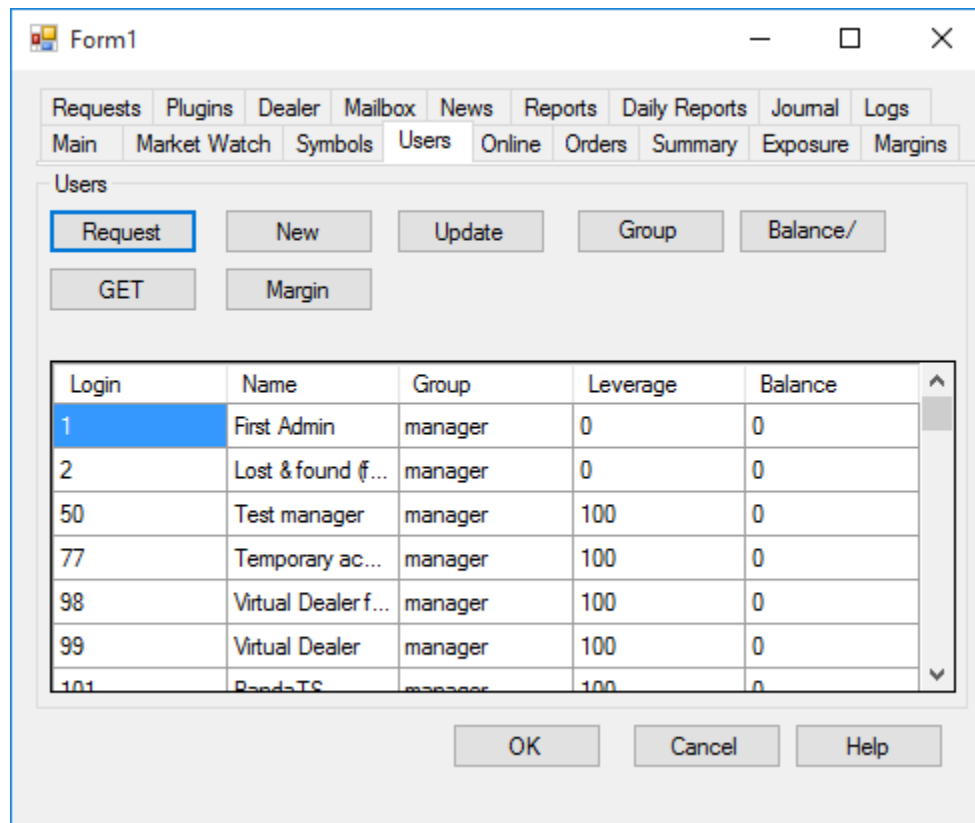
        /// <summary>
        /// Symbol configuration
        /// </summary>
        /// <param name="_symbol">Text</param>
        /// <param name="_description">Text</param>
        /// <param name="_stopsLevel">Number</param>
        /// <param name="_spread">Number</param>
        /// <param name="_bidTickValue">Number</param>
        /// <param name="_askTickValue">Number</param>
        public SymbolListFxtf(string _symbol, string _description, int _stopsLevel,
int _spread, double _bidTickValue, double _askTickValue)
        {
            Symbol = _symbol;
            Description = _description;
            StopsLevel = _stopsLevel;
            Spread = _spread;
            BidTickValue = _bidTickValue;
            AskTickValue = _askTickValue;
        }
    }
}

```

`SymbolsGetAll()` Get symbols configuration. `SymbolsRefresh()` should be called before this method.

User Tab

The Users Tab gets all the users.



Login	Name	Group	Leverage	Balance
1	First Admin	manager	0	0
2	Lost & found (f...	manager	0	0
50	Test manager	manager	100	0
77	Temporary ac...	manager	100	0
98	Virtual Dealer f...	manager	100	0
99	Virtual Dealer	manager	100	0
101	Reada TS	manager	100	0

Fig: Users Tab

`PumpingSwitch()` method used for switching into pumping mode.

```
/// <summary>
/// Switch into pumping mode
/// delegate will be invoked on any pumping event
/// </summary>
private void PumpingSwitch()
{
    if (switchFlag == false)
    {
        var are = new AutoResetEvent(false);
        clrWrapper.PumpingSwitch(i =>
        {
            if (i == 0) // 0 - means pumping started
                are.Set();
        });

        are.WaitOne();
        switchFlag = true;
    }
}
```

UsersGet() method Get all users in pumping mode.

```
/// <summary>
/// Get all users in pumping mode
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>

private void RequestButton_UsersClick(object sender, EventArgs e)
{
    //user

    if (clrWrapper.IsConnected() != 0)
    {
        LogIn();
        //User Request
        PumpingSwitch();
        IList<UserRecord> tradeRecords = clrWrapper.UsersGet();
        IList<UserListFxtf> list = new List<UserListFxtf>();

        foreach (var p in tradeRecords)
        {
            int login = p.Login;
            string name = p.Name;
            string group = p.Group;
            int leverage = p.Leverage;
            double balance = p.Balance;
            list.Add(new UserListFxtf(login, name, group, leverage, balance));
        }
        dataGridView2_Users.Visible = true;
        dataGridView2_Users.DataSource = list;
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridView2_Users.DataSource = null;
    }
}
```

And the `UserListFxtf` class is:

```
/// <summary>
/// Object that represents user record
///
/// </summary>
public class UserListFxtf
{
    /// <summary>
    /// Account number
    ///
    /// </summary>
    [DisplayName("Login")]
    public int Login { get; set; }
    /// <summary>
    /// Name
    ///
    /// </summary>
    [DisplayName("Name")]
    public string Name { get; set; }
    /// <summary>
    /// Group user belongs to
    ///
    /// </summary>
    [DisplayName("Group")]
    public string Group { get; set; }
    /// <summary>
    /// Leverage
    ///
    /// </summary>
    [DisplayName("Leverage")]
    public int Leverage { get; set; }
    /// <summary>
    /// Balance
    ///
    /// </summary>
    [DisplayName("Balance")]
    public double Balance { get; set; }
    /// <summary>
    /// Symbol configuration
    /// </summary>
    /// <param name="_login">Text</param>

    /// <param name="_name">Text</param>
    /// <param name="_group">Text</param>
    /// <param name="_leverage">number</param>
    /// <param name="_balance">number</param>

    public UserListFxtf(int _login, string _name, string _group, int _leverage,
double _balance)
    {
        Login = _login;
        Name = _name;
        Group = _group;
        Leverage = _leverage;
        Balance = _balance;
    }
}
```


Online Tab

The Online Tab gets all the online users.

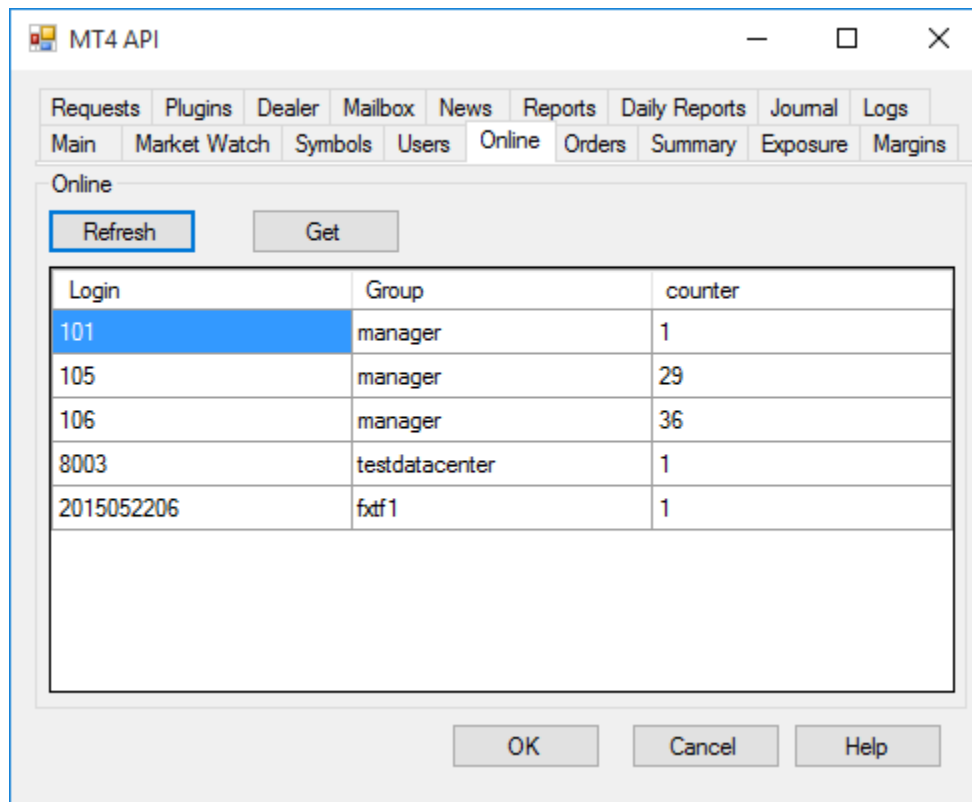


Fig: online users

```
/// <summary>
/// Switch into pumping mode
/// delegate will be invoked on any pumping event
/// </summary>
private void PumpingSwitch()
{
    if (switchFlag == false)
    {
        var are = new AutoResetEvent(false);
        clrWrapper.PumpingSwitch(i =>
        {
            if (i == 0) // 0 - means pumping started
                are.Set();
        });

        are.WaitOne();
        switchFlag = true;
    }
}
```

OnlineGet() method get all the online users in pumping mode.

```
private void RefreshButton_OnlineClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        LogIn();
        PumpingSwitch();

        IList<OnlineRecord> onlineRecords = clrWrapper.OnlineGet();
        IList<OnlineRecordsListFxtf> list = new List<OnlineRecordsListFxtf>();

        foreach (var p in onlineRecords)
        {
            int login = p.Login;
            string group = p.Group;
            int counter = p.Counter;

            list.Add(new OnlineRecordsListFxtf(login, group, counter));
        }

        dataGridViewOnlineUser.Visible = true;
        dataGridViewOnlineUser.DataSource = list;
    }
    else
    {
        dataGridViewOnlineUser.DataSource = null;
        MessageBox.Show(CONNECT_FIRST);
    }
}
```

And the `OnlineRecordsListFxtf` class is:

```
/// <summary>
/// Object that represents online record
///
/// </summary>
public class OnlineRecordsListFxtf
{
    /// <summary>
    /// User login
    ///
    /// </summary>
    [DisplayName("Login")]
    public int Login { get; set; }
    /// <summary>
    /// User group
    ///
    /// </summary>
    [DisplayName("Group")]
    public string Group { get; set; }
    /// <summary>
    /// Connections counter
    ///
    /// </summary>
    [DisplayName("counter")]
    public int Counter { get; set; }

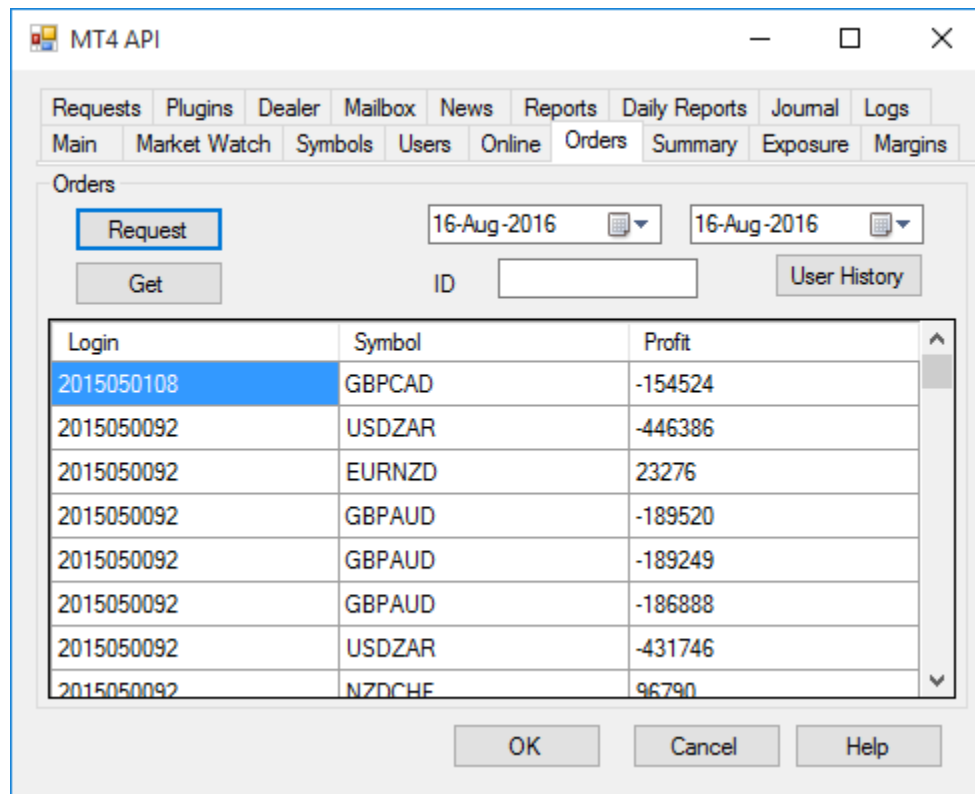
    /// <summary>
    /// Symbol configuration
    /// </summary>
    /// <param name="_login">Number</param>
    /// <param name="_group">Text</param>
    /// <param name="_counter">Number</param>
    public OnlineRecordsListFxtf(int _login, string _group, int _counter)
    {
        Login = _login;
        Group = _group;
        Counter = _counter;
    }
}
```

Orders Tab

This tab is used for requesting list of all orders.

Request

Here, list of all orders are displayed.



The screenshot shows the MT4 API window with the 'Orders' tab selected. The 'Request' button is highlighted. The date range is set to '16-Aug-2016' to '16-Aug-2016'. The 'Get' button is visible. Below the buttons is a table of orders.

Login	Symbol	Profit
2015050108	GBPCAD	-154524
2015050092	USDZAR	-446386
2015050092	EURNZD	23276
2015050092	GBPAUD	-189520
2015050092	GBPAUD	-189249
2015050092	GBPAUD	-186888
2015050092	USDZAR	-431746
2015050092	NZDCHF	96790

At the bottom of the window are 'OK', 'Cancel', and 'Help' buttons.

Fig: Request of Orders Tab.

TradesRequest() method is used for requesting list of all orders.

```
/// <summary>
/// Request list of all orders
///
/// </summary>
private void RequestButton_OrdersTabClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        Login();
        IList<TradeRecord> tradeRecords = clrWrapper.TradesRequest();
        IList<TradeRecordsListFxtf> list = new List<TradeRecordsListFxtf>();

        foreach (var p in tradeRecords)
        {
            int login = p.Login;
            string symbol = p.Symbol;
            double profit = p.Profit;

            list.Add(new TradeRecordsListFxtf(login, symbol, profit));
        }
        dataGridView1_Orders.Visible = true;
        dataGridView1_Orders.DataSource = list;
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridView1_Orders.DataSource = null;
    }
}
```

And the `TradeRecordsListFxtf` class is:

```
/// <summary>
/// Object that represents trade record
///
/// </summary>
public class TradeRecordsListFxtf
{
    /// <summary>
    /// Owner's login
    ///
    /// </summary>
    [DisplayName("Login")]
    public int Login { get; set; }
    /// <summary>
    /// Security
    ///
    /// </summary>
    [DisplayName("Symbol")]
    public string Symbol { get; set; }

    /// <summary>
    /// Profit
    ///
    /// </summary>
    [DisplayName("Profit")]
    public double Profit { get; set; }

    /// <summary>
    /// Trade record
    /// </summary>
    /// <param name="_login">Number</param>

    /// <param name="_symbol">Text</param>
    /// <param name="_profit">Number</param>

    public TradeRecordsListFxtf(int _login, string _symbol, double _profit)
    {
        Login = _login;
        Symbol = _symbol;
        Profit = _profit;
    }
}
```

User History

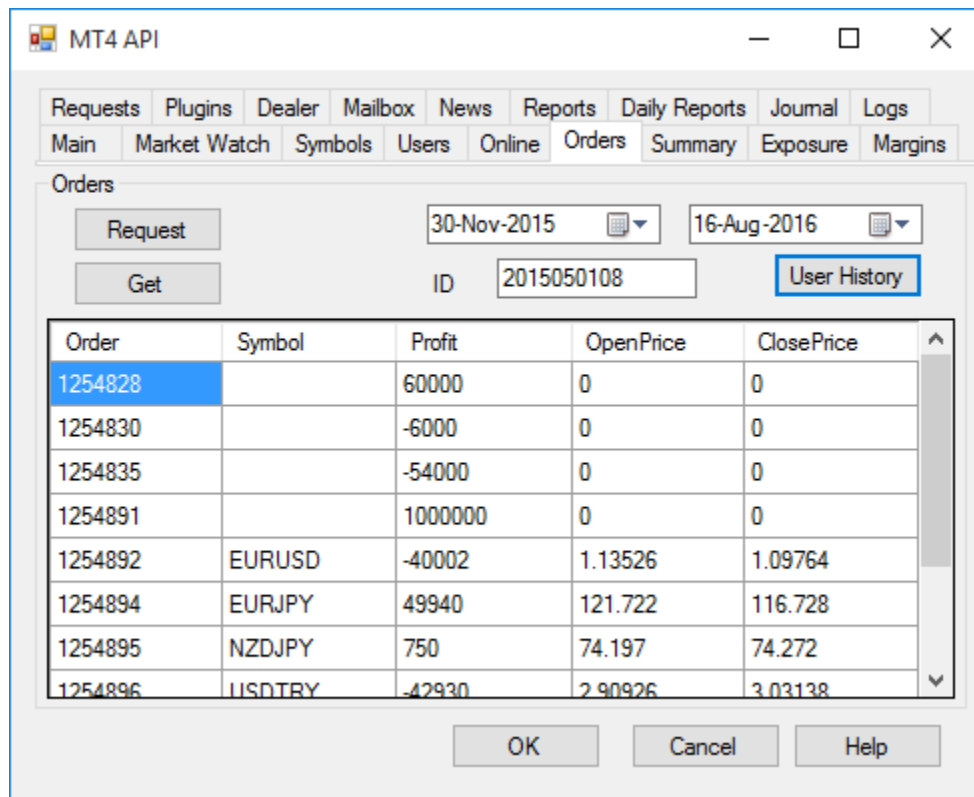


Fig: User History of Orders Tab.

```

/// <summary>
/// Request trades for specific user in period
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void UserHistoryButton_OrdersClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        Login();
        DateTime dt1 = dateTimePickerFrom.Value.Date;
        uint from = ToUnixTime(dt1);

        DateTime dt2 = dateTimePickerTo.Value.Date;
        uint to = ToUnixTime(dt2);

        if ((int)to - (int)from >= 0)
        {
            try
            {
                int login = int.Parse(LoginTextBox_Orders.Text);
                IList<TradeRecord> tradeRecords =
clrWrapper.TradesUserHistory(login, from, to);
                IList<TradeRecordFxtf> list = new List<TradeRecordFxtf>();
            }
        }
    }
}

```

```

        foreach (var p in tradeRecords)
        {
            int order = p.Order;
            string symbol = p.Symbol;
            double openPrice = p.OpenPrice;
            double closePrice = p.ClosePrice;
            double profit = p.Profit;

            list.Add(new TradeRecordFxtf(order, symbol, openPrice,
closePrice, profit));
        }
        dataGridView1_Orders.Visible = true;
        dataGridView1_Orders.DataSource = list;
    }
    catch (Exception exception)
    {
        const string msg = "Can't parse the ID field.";
        MessageBox.Show(msg);
    }

    }
    else
    {
        const string msg = "'To date' must be greater or equal to 'from
date'.";
        MessageBox.Show(msg);
    }

    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridView1_Orders.DataSource = null;
    }
}

```


And the `TradeRecordFxtf` class is:

```
/// <summary>
/// Object that represents trade record
/// </summary>
public class TradeRecordFxtf
{
    /// <summary>
    /// Order ticket
    ///
    /// </summary>
    public int Order { get; set; }
    /// <summary>
    /// Security
    ///
    /// </summary>
    public string Symbol { get; set; }
    /// <summary>
    /// Open price
    ///
    /// </summary>
    public double OpenPrice { get; set; }
    /// <summary>
    /// Close price
    ///
    /// </summary>
    public double ClosePrice { get; set; }
    /// <summary>
    /// Profit
    ///
    /// </summary>
    public double Profit { get; set; }
    /// <summary>
    ///
    /// </summary>
    /// <param name="_order">Number</param>
    /// <param name="_symbol">Text</param>
    /// <param name="_openPrice">Number</param>
    /// <param name="_closePrice">Number</param>
    /// <param name="_profit">Number</param>
    public TradeRecordFxtf(int _order, string _symbol, double _openPrice, double
_closePrice, double _profit)
    {
        Order = _order;
        Symbol = _symbol;
        OpenPrice = _openPrice;
        ClosePrice = _closePrice;
        Profit = _profit;
    }
}
```

Journal Tab

This tab shows the requesting log of the server.

Time	IP	Message
2016.08.10	(4096 kb)	-----
2016.08.10 00:00:01.056	Monitor	connections: 88 free mem...
2016.08.10 00:00:01.069	Monitor	process cpu: 0% net in: 16...
2016.08.10 00:00:15.767		Service: preparation for ba...
2016.08.10 00:00:16.210		Service: the database of th...
2016.08.10 00:00:16.476		Service: the database of th...
2016.08.10 00:00:16.755		Service: the daily reports d...
2016.08.10 00:00:16.755		Service: the archive datab...

Fig: Journal Tab

```
/// <summary>
/// Request for the server log for a certain period of time
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RequestButton_JournalClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        LogIn();
        string filter = comboBox1_Journal.SelectedText;

        DateTime dt1 = dateTimePicker_From.Value.Date;
        uint from = ToUnixTime(dt1);

        DateTime dt2 = dateTimePicker_To.Value.Date;
        uint to = ToUnixTime(dt2);

        if ((int) to - (int) from >= 0)
```

```

        {
            IList<ServerLog> serverLogs = clrWrapper.JournalRequest(0, from, to,
filter);
            IList<JournalRequestFxtf> list = new List<JournalRequestFxtf>();

            foreach (var p in serverLogs)
            {
                string time = p.Time;
                string ip = p.Ip;
                string message = p.Message;

                list.Add(new JournalRequestFxtf(time, ip, message));
            }
            dataGridView_JournalRequest.Visible = true;
            dataGridView_JournalRequest.DataSource = list;
        }
        else
        {
            string msg = "'To date' must be greater or equal to 'from date'.";
            MessageBox.Show(msg);
        }
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridView_JournalRequest.DataSource = null;
    }
}

```

`JournalRequest(int mode, uint from, uint to, string filter)` is used for Requesting for the server log for a certain period of time.

```

    /// <summary>
    /// Request for the server log for a certain period of time
    ///
    /// </summary>
    /// <param name="mode">mode</param>
    /// <param name="from">from</param>
    /// <param name="to">to</param>
    /// <param name="filter">filter</param>
    public IList<ServerLog> JournalRequest(int mode, uint from, uint to, string filter);

```

And the `JournalRequestFxtf` class is:

```
/// <summary>
/// Object that represents server log
///
/// </summary>
public class JournalRequestFxtf
{
    /// <summary>
    /// Time
    ///
    /// </summary>
    [DisplayName("Time")]
    public string Time { get; set; }
    /// <summary>
    /// IP
    ///
    /// </summary>
    [DisplayName("IP")]
    public string IP { get; set; }
    /// <summary>
    /// Message
    ///
    /// </summary>
    [DisplayName("Message")]
    public string Message { get; set; }

    /// <summary>
    /// Server log
    /// </summary>
    /// <param name="_time">Text</param>

    /// <param name="_ip">Text</param>
    /// <param name="_message">Text</param>

    public JournalRequestFxtf(string _time, string _ip, string _message)
    {
        Time = _time;
        IP = _ip;
        Message = _message;
    }
}
```

All dates in MT4 represented as unixtimestamp with slight modification: it defined as the number of seconds that have elapsed since 00:00:00 Trade Server Time Zone, Thursday, 1 January 1970.

Market Watch

This tab is for getting updated prices in pumping mode.

Symbol	Bid	Ask
GBPUSD-M	1.30551	1.30617
EURGBP-M	0.85535	0.8559
EURJPY-M	113.133	113.182
USDCAD-M	1.30346	1.30396
GBPAUD	1.69063	1.69143
GBPNZD	1.80491	1.8058100000000001
GBPAUD-M	1.69051	1.69156
GBPCHF-M	1.27596	1.2769599999999999

Fig: Market Watch Tab.

`PumpingSwitch()` method used for switching into pumping mode.

```
/// <summary>
/// Switch into pumping mode
/// delegate will be invoked on any pumping event
/// </summary>
private void PumpingSwitch()
{
    if (switchFlag == false)
    {
        var are = new AutoResetEvent(false);
        clrWrapper.PumpingSwitch(i =>
        {
            if (i == 0) // 0 - means pumping started
                are.Set();
        });

        are.WaitOne();
        switchFlag = true;
    }
}
```

```
}
```

`SymbolInfoUpdated()` return updated prices in pumping mode.

```

IList<SymbolInfoFxtf> listMarketWatch = new List<SymbolInfoFxtf>();
/// <summary>
/// Market Watch
/// </summary>
public void MarketWatchFunc()
{
    if (MarketWatchTimeEnable && clrWrapper.IsConnected() != 0)
    {
        MarketWatchTimeEnable = false;

        IList<SymbolInfo> symbolInfos = clrWrapper.SymbolInfoUpdated();

        foreach (var p in symbolInfos)
        {
            string symbol = p.Symbol;
            double bid = p.Bid;
            double ask = p.Ask;

            //update the price of matched symbol
            var list = listMarketWatch.Where(d => d.Symbol ==
symbol).FirstOrDefault();
            if (list != null)
            {
                list.Bid = bid;
                list.Ask = ask;
            }
            else
            {
                listMarketWatch.Add(new SymbolInfoFxtf(symbol, bid, ask));
            }
        }

        dataGridView_MarketWatch.DataSource = listMarketWatch.ToList();
        dataGridView_MarketWatch.Visible = true;
        MarketWatchTimeEnable = true;
    }
}

```

```

/// <summary>
/// Get updated prices in pumping mode
///
/// </summary>
private void RefreshButton_MarketWatchClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        PumpingSwitch();
        MarketWatchTimeEnable = true;
        MarketWatchFunc();
        timer1.Enabled = true;
    }
    else
    {
        MarketWatchTimeEnable = false;
        MessageBox.Show(CONNECT_FIRST);
        dataGridView_MarketWatch.DataSource = null;
    }
}

```

```

/// <summary>
/// Timer that refresh ask and bid value in every 3000 milliseconds
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void timer1_Tick(object sender, EventArgs e)
{
    timer1.Interval = 3000; //milliseconds
    MarketWatchFunc();
}

```

And the `SymbolInfoFxtf` class is:

```

/// <summary>
/// Object that represents symbol info
///
/// </summary>
public class SymbolInfoFxtf
{
    /// <summary>
    /// Symbol name
    ///
    /// </summary>
    [DisplayName("Symbol")]
    public string Symbol { get; set; }
    /// <summary>
    /// Bid
    ///
    /// </summary>
    [DisplayName("Bid")]
    public double Bid { get; set; }
    /// <summary>
    /// Ask
    ///
    /// </summary>
    [DisplayName("Ask")]
    public double Ask { get; set; }

    /// <summary>
    /// Symbol info
    /// </summary>
    /// <param name="_symbol">Text</param>

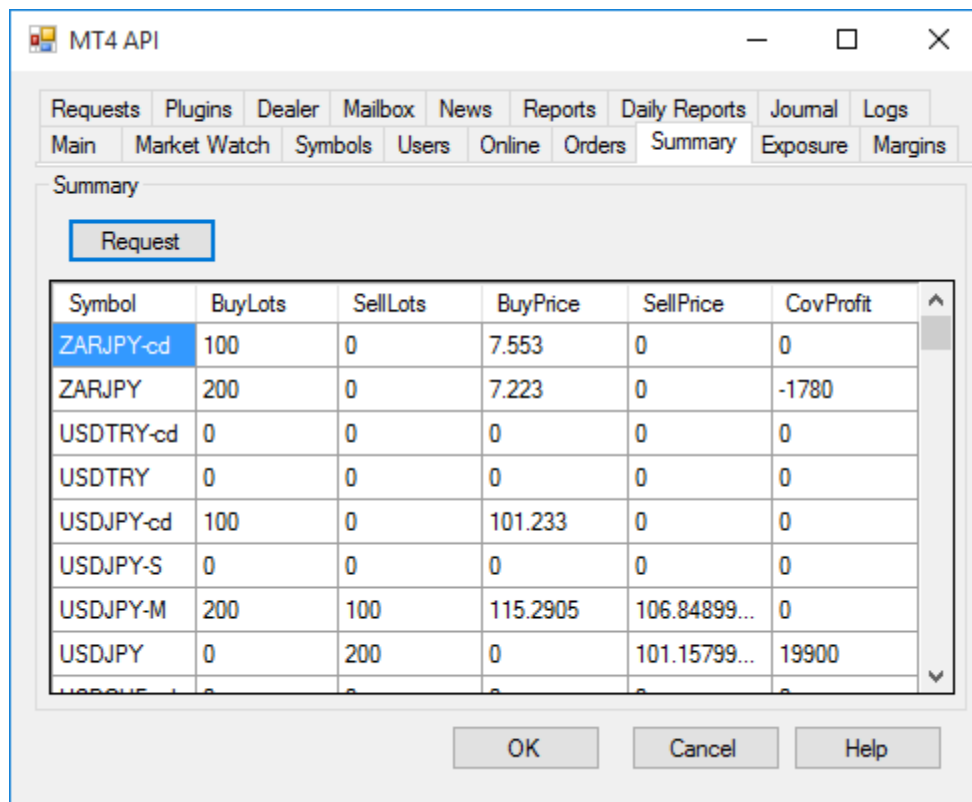
    /// <param name="_bid">Number</param>
    /// <param name="_ask">Number</param>

    public SymbolInfoFxtf(string _symbol, double _bid, double _ask)
    {
        Symbol = _symbol;
        Bid = _bid;
        Ask = _ask;
    }
}

```


Summary

This tab is for getting trade summary for all symbols in pumping mode.



Symbol	BuyLots	SellLots	BuyPrice	SellPrice	CovProfit
ZARJPY-cd	100	0	7.553	0	0
ZARJPY	200	0	7.223	0	-1780
USDTRY-cd	0	0	0	0	0
USDTRY	0	0	0	0	0
USDJPY-cd	100	0	101.233	0	0
USDJPY-S	0	0	0	0	0
USDJPY-M	200	100	115.2905	106.84899...	0
USDJPY	0	200	0	101.15799...	19900

Fig: Summary Tab.

`PumpingSwitch()` method used for switching into pumping mode.

```
/// <summary>
/// Switch into pumping mode
/// delegate will be invoked on any pumping event
/// </summary>
private void PumpingSwitch()
{
    if (switchFlag == false)
    {
        var are = new AutoResetEvent(false);
        clrWrapper.PumpingSwitch(i =>
        {
            if (i == 0) // 0 - means pumping started
                are.Set();
        });

        are.WaitOne();
        switchFlag = true;
    }
}
```

```

/// <summary>
/// Get trade summary for all symbols in pumping mode
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RequestButton_SummaryClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        PumpingSwitch();
        IList<SymbolSummary> symbolSummaries = clrWrapper.SummaryGetAll();
        IList<SummaryFxtf> list = new List<SummaryFxtf>();

        foreach (var p in symbolSummaries)
        {
            string symbol = p.Symbol;
            long buyLots = p.BuyLots;
            long sellLots = p.SellLots;

            double buyPrice = p.BuyPrice;
            double sellPrice = p.SellPrice;
            double covProfit = p.CovProfit;
            list.Add(new SummaryFxtf(symbol, buyLots, sellLots, buyPrice,
sellPrice, covProfit));
        }

        dataGridViewRequest_Summary.DataSource = list;
        dataGridViewRequest_Summary.Visible = true;
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridViewRequest_Summary.DataSource = null;
    }
}

```

And the `SummaryFxtf` class is:

```

/// <summary>
/// Object that represents symbol summary
/// </summary>
public class SummaryFxtf
{
    /// <summary>
    /// Symbol
    /// </summary>
    public string Symbol { get; set; }

    /// <summary>
    /// Buy volume

```

```

    ///
    /// </summary>
    public long BuyLots { get; set; }
    /// <summary>
    /// Sell volume
    ///
    /// </summary>
    public long SellLots { get; set; }

    /// <summary>
    /// Average buy price
    ///
    /// </summary>
    public double BuyPrice { get; set; }
    /// <summary>
    /// Average sell price
    ///
    /// </summary>
    public double SellPrice { get; set; }

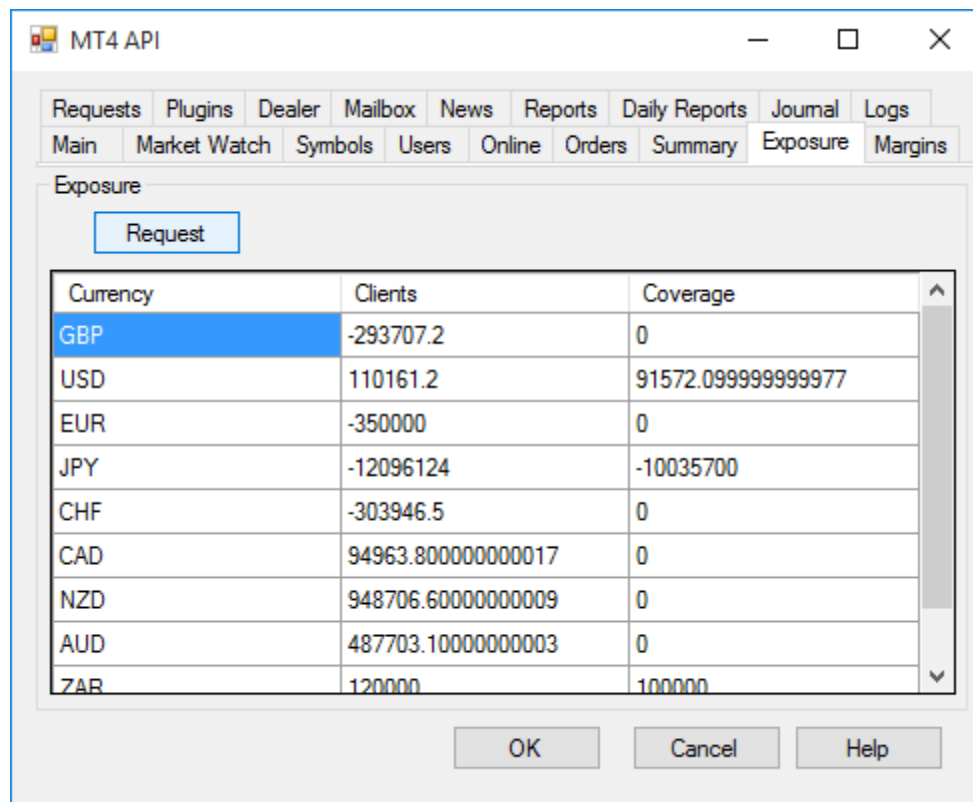
    /// <summary>
    /// Coverage profit
    ///
    /// </summary>
    public double CovProfit { get; set; }

    /// <summary>
    ///
    /// </summary>
    /// <param name="_symbol">Text</param>
    /// <param name="_buyLots">Number</param>
    /// <param name="_sellLots">Number</param>
    /// <param name="_buyPrice">Number</param>
    /// <param name="_sellPrice">Number</param>
    /// <param name="_covProfit">Number</param>
    public SummaryFxtf(string _symbol, long _buyLots, long _sellLots, double
_buyPrice, double _sellPrice, double _covProfit )
    {
        Symbol = _symbol;
        BuyLots = _buyLots;
        SellLots = _sellLots;
        BuyPrice = _buyPrice;
        SellPrice = _sellPrice;
        CovProfit = _covProfit;
    }
}

```

Exposure

This tab is for getting company's exposure for currencies in pumping mode.



Currency	Clients	Coverage
GBP	-293707.2	0
USD	110161.2	91572.099999999977
EUR	-350000	0
JPY	-12096124	-10035700
CHF	-303946.5	0
CAD	94963.800000000017	0
NZD	948706.600000000009	0
AUD	487703.100000000003	0
ZAR	120000	100000

Fig: Exposure Tab.

`PumpingSwitch()` method used for switching into pumping mode.

```
/// <summary>
/// Switch into pumping mode
/// delegate will be invoked on any pumping event
/// </summary>
private void PumpingSwitch()
{
    if (switchFlag == false)
    {
        var are = new AutoResetEvent(false);
        clrWrapper.PumpingSwitch(i =>
        {
            if (i == 0) // 0 - means pumping started
                are.Set();
        });

        are.WaitOne();
        switchFlag = true;
    }
}
```

```

/// <summary>
/// Get company's exposure for currencies in pumping mode
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RequestButton_ExposureClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        PumpingSwitch();
        IList<ExposureValue> exposureValues= clrWrapper.ExposureGet();
        IList<ExposureValueFxtf> list = new List<ExposureValueFxtf>();

        foreach (var p in exposureValues)
        {
            string currency = p.Currency;
            double clients = p.Clients;
            double coverage = p.Coverage;
            list.Add(new ExposureValueFxtf(currency,clients,coverage));
        }

        dataGridViewExposure.DataSource = list;
        dataGridViewExposure.Visible = true;
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridViewExposure.DataSource = null;
    }
}

```

And the `ExposureValueFxtf` class is:

```

public class ExposureValueFxtf
{
    /// <summary>
    /// Currency
    /// </summary>
    public string Currency { get; set; }
    /// <summary>
    /// Clients volume
    /// </summary>
    public double Clients { get; set; }
    /// <summary>
    /// Coverage volume
    /// </summary>
    public double Coverage { get; set; }

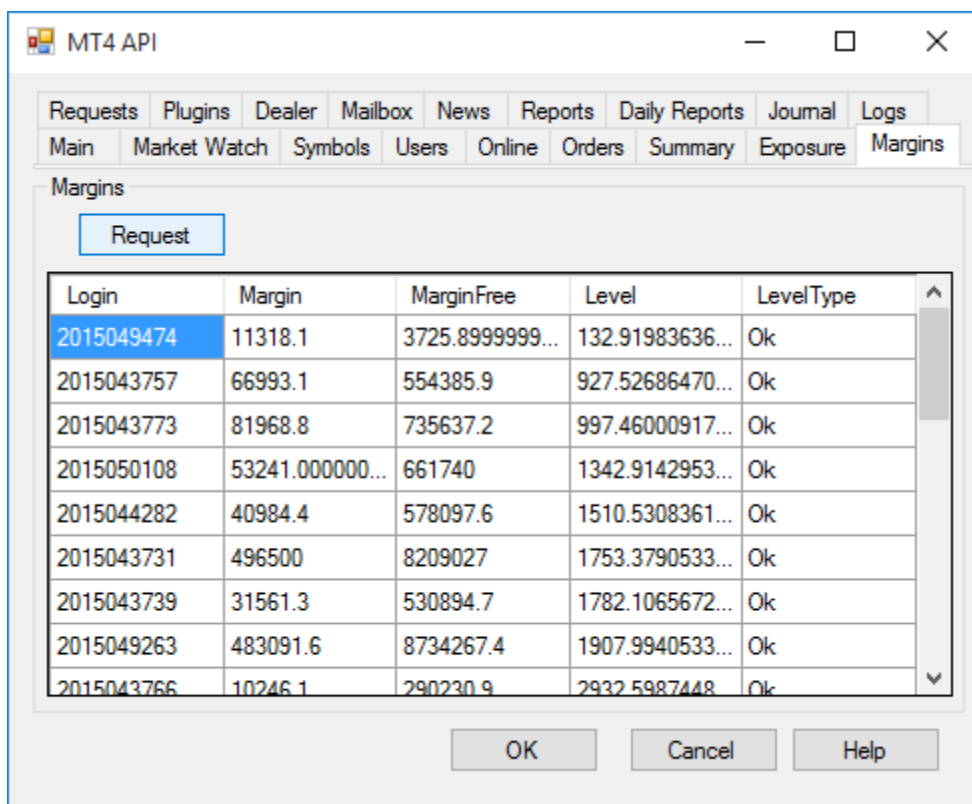
    /// <summary>
    ///

```

```
/// </summary>
/// <param name="_currency">Text</param>
/// <param name="_clients">Number</param>
/// <param name="_coverage">Number</param>
public ExposureValueFxtf(string _currency, double _clients, double _coverage)
{
    Currency = _currency;
    Clients = _clients;
    Coverage = _coverage;
}
}
```

Margin

This tab is for getting list of margin requirements of accounts in pumping mode.



The screenshot shows the 'MT4 API' window with the 'Margins' tab selected. A 'Request' button is visible above a table. The table has five columns: Login, Margin, MarginFree, Level, and LevelType. It contains ten rows of data, with the first row highlighted in blue. At the bottom of the window are 'OK', 'Cancel', and 'Help' buttons.

Login	Margin	MarginFree	Level	LevelType
2015049474	11318.1	3725.8999999...	132.91983636...	Ok
2015043757	66993.1	554385.9	927.52686470...	Ok
2015043773	81968.8	735637.2	997.46000917...	Ok
2015050108	53241.000000...	661740	1342.9142953...	Ok
2015044282	40984.4	578097.6	1510.5308361...	Ok
2015043731	496500	8209027	1753.3790533...	Ok
2015043739	31561.3	530894.7	1782.1065672...	Ok
2015049263	483091.6	8734267.4	1907.9940533...	Ok
2015043766	10246.1	290230.9	2932.5987448	Ok

Fig: Margin Tab.

`PumpingSwitch()` method used for switching into pumping mode.

```
/// <summary>
/// Switch into pumping mode
/// delegate will be invoked on any pumping event
/// </summary>
private void PumpingSwitch()
{
    if (switchFlag == false)
    {
        var are = new AutoResetEvent(false);
        clrWrapper.PumpingSwitch(i =>
        {
            if (i == 0) // 0 - means pumping started
                are.Set();
        });

        are.WaitOne();
        switchFlag = true;
    }
}
```

```

/// <summary>
/// Get list of margin requirements of accounts in pumping mode
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RequestButton_MarginsClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        PumpingSwitch();
        IList<MarginLevel> marginLevels = clrWrapper.MarginsGet();
        IList<MarginLevelFxtf> list = new List<MarginLevelFxtf>();

        foreach (var p in marginLevels)
        {
            int login = p.Login;
            double margin = p.Margin;
            double marginFree = p.MarginFree;
            double level = p.Level;
            string levelType = p.LevelType.ToString();
            list.Add(new MarginLevelFxtf(login, margin, marginFree, level,
levelType));
        }

        dataGridViewMargins.DataSource = list;
        dataGridViewMargins.Visible = true;
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridViewMargins.DataSource = null;
    }
}

```

And the `MarginLevelFxtf` class is:

```

/// <summary>
/// Object that represents margin level
/// </summary>
public class MarginLevelFxtf
{
    /// <summary>
    /// User login
    /// </summary>
    public int Login { get; set; }
    /// <summary>
    /// Margin requirements
    /// </summary>
    public double Margin { get; set; }
    /// <summary>
    /// Free margin
    /// </summary>
}

```



```

    public double MarginFree { get; set; }
    /// <summary>
    /// Margin level
    ///
    /// </summary>
    public double Level { get; set; }
    /// <summary>
    /// Level type(ok/margincall/stopout)
    ///
    /// </summary>
    public string LevelType { get; set; }

    /// <summary>
    ///
    /// </summary>
    /// <param name="_login">Number</param>
    /// <param name="_margin">Number</param>
    /// <param name="_marginFree">Number</param>
    /// <param name="_level">Number</param>
    /// <param name="_levelType">Text</param>
    public
MarginLevelFxtf(int _login,double _margin, double _marginFree, double _level,string
_levelType)
    {
        Login = _login;
        Margin = _margin;
        MarginFree = _marginFree;
        Level = _level;
        LevelType = _levelType;
    }
}

```

News

This tab is for getting trade summary for all symbols in pumping mode.

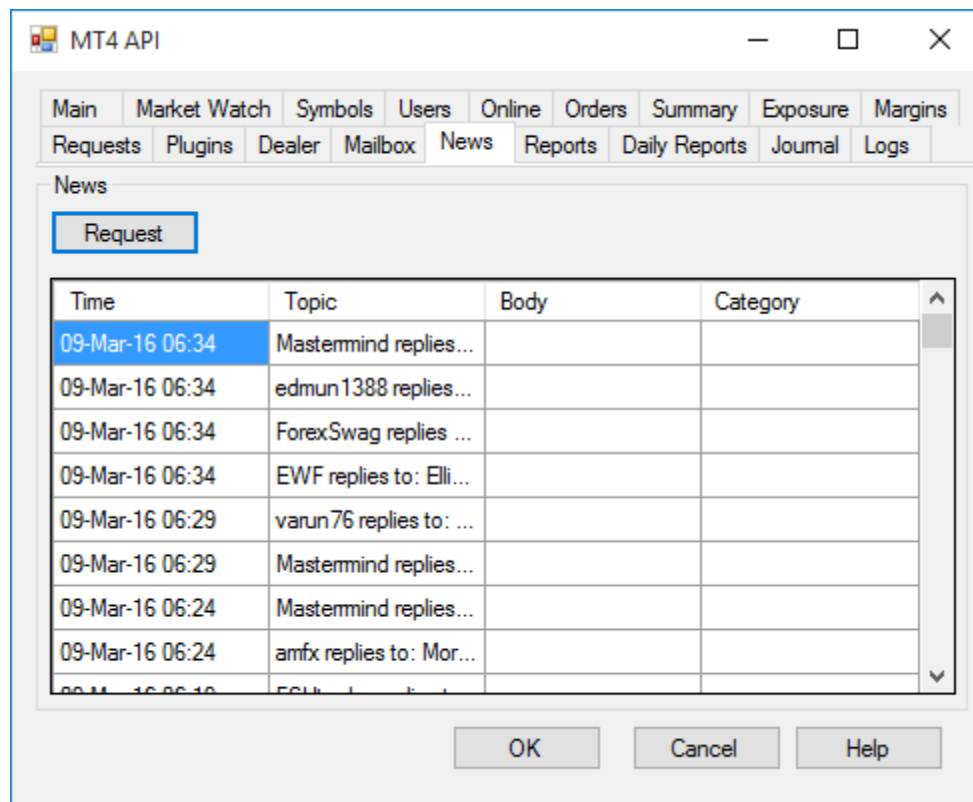


Fig: News Tab.

`PumpingSwitch()` method used for switching into pumping mode.

```
/// <summary>
/// Switch into pumping mode
/// delegate will be invoked on any pumping event
/// </summary>
private void PumpingSwitch()
{
    if (switchFlag == false)
    {
        var are = new AutoResetEvent(false);
        clrWrapper.PumpingSwitch(i =>
        {
            if (i == 0) // 0 - means pumping started
                are.Set();
        });

        are.WaitOne();
        switchFlag = true;
    }
}
```

```

/// <summary>
/// Get heading of income news in pumping mode
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RequestButton_NewsClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        PumpingSwitch();
        IList<NewsTopic> newsTopics = clrWrapper.NewsGet();
        IList<NewsTopicFxtf> list = new List<NewsTopicFxtf>();

        foreach (var p in newsTopics)
        {
            uint time = p.Time;
            DateTime date = ToDateTime(time);
            string topic = p.Topic;
            string body = p.Body;
            string category = p.Category;
            list.Add(new NewsTopicFxtf(date, body, category, topic));
        }

        dataGridViewNews.DataSource = list;
        dataGridViewNews.Visible = true;
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridViewNews.DataSource = null;
    }
}

```

And the `NewsTopicFxtf` class is:

```

/// <summary>
/// Object that represents news topic
/// </summary>
public class NewsTopicFxtf
{
    /// <summary>
    /// News time
    ///
    /// </summary>
    public DateTime Time { get; set; }
    /// <summary>
    /// News topic
    ///
    /// </summary>
    public string Topic { get; set; }
    /// <summary>

```

```

    /// Body (if present)
    ///
    /// </summary>
    public string Body { get; set; }
    /// <summary>
    /// News Category
    ///
    /// </summary>
    public string Category { get; set; }
    /// <summary>
    ///
    /// </summary>
    /// <param name="_date">DateTime</param>
    /// <param name="_body">Text</param>
    /// <param name="_category">Text</param>
    /// <param name="_topic">Text</param>
    public NewsTopicFxtf(DateTime _date, string _body, string _category, string
_topic)
    {
        Time = _date;
        Body = _body;
        Topic = _topic;
        Category = _category;
    }
}

```

Mail Box

This tab is for last mails of internal mail system.

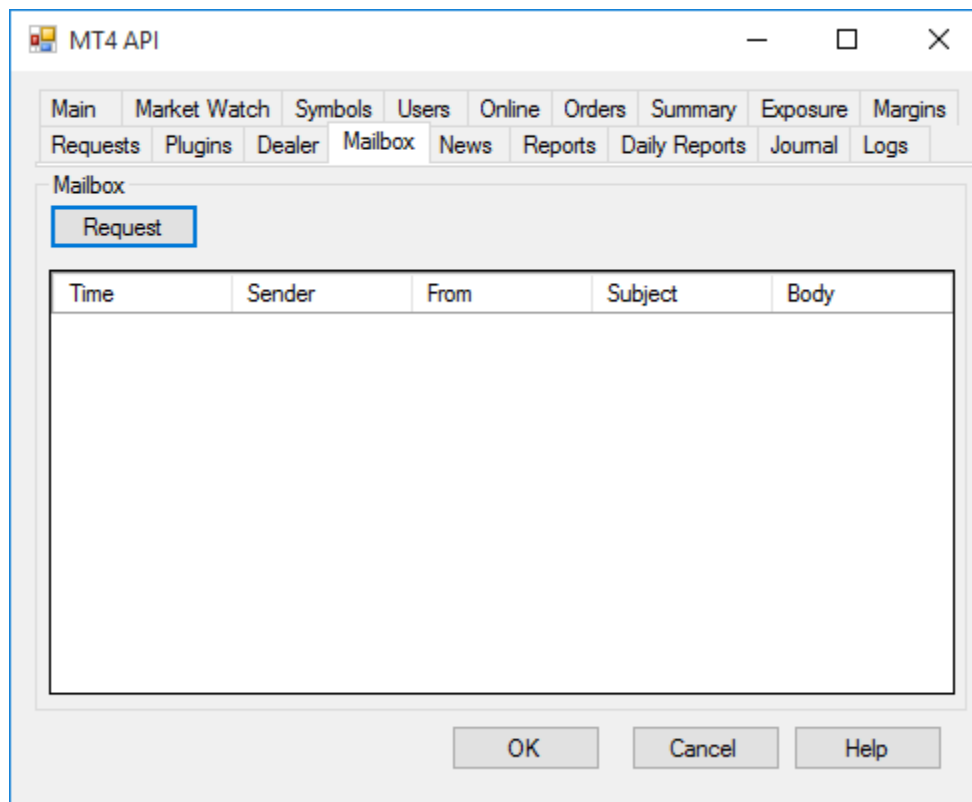


Fig: Mail Box Tab.

```
/// <summary>
/// Get last mails of internal mail system
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RequestButton_MailBoxClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        IList<MailBox> mailBoxs = clrWrapper.MailsRequest();
        IList<MailBoxFxtf> list = new List<MailBoxFxtf>();

        foreach (var p in mailBoxs)
        {
            uint time = p.Time;
            DateTime date = ToDateTime(time);
            string from = p.From;
            string subject = p.Subject;
            string body = p.Body;
        }
    }
}
```

```
        int send= p.Sender;
        list.Add(new MailBoxFxtf(date, from, subject, body, send));
    }

    dataGridViewMailBox.DataSource = list;
    dataGridViewMailBox.Visible = true;
}
else
{
    MessageBox.Show(CONNECT_FIRST);
    dataGridViewMailBox.DataSource = null;
}
LogIn();
}
```

And the MailBoxFxtf class is:

```
/// <summary>
/// Object that represents mailbox
/// </summary>
class MailBoxFxtf
{
    /// <summary>
    /// Receive time
    ///
    /// </summary>
    public DateTime Time { get; set; }

    /// <summary>
    /// Mail sender (login)
    ///
    /// </summary>
    public int Sender { get; set; }
    /// <summary>
    /// Mail sender (name)
    ///
    /// </summary>
    public string From { get; set; }
    /// <summary>
    /// Mail subject
    ///
    /// </summary>
    public string Subject { get; set; }
    /// <summary>
    /// Pointer to mail Body
    ///
    /// </summary>
    public string Body { get; set; }
    /// <summary>
    ///
    /// </summary>
    /// <param name="_date">DateTime</param>
    /// <param name="_from">Text</param>
    /// <param name="_subject">Text</param>
    /// <param name="_body">Text</param>
    /// <param name="_sender">Text</param>
    public MailBoxFxtf(DateTime _date, string _from, string _subject, string
_body, int _sender)
    {
        Time = _date;
        From = _from;
        Subject = _subject;
        Body = _body;
        Sender = _sender;
    }
}
```

Plugins

This tab is for getting list of installed plugins in pumping mode.

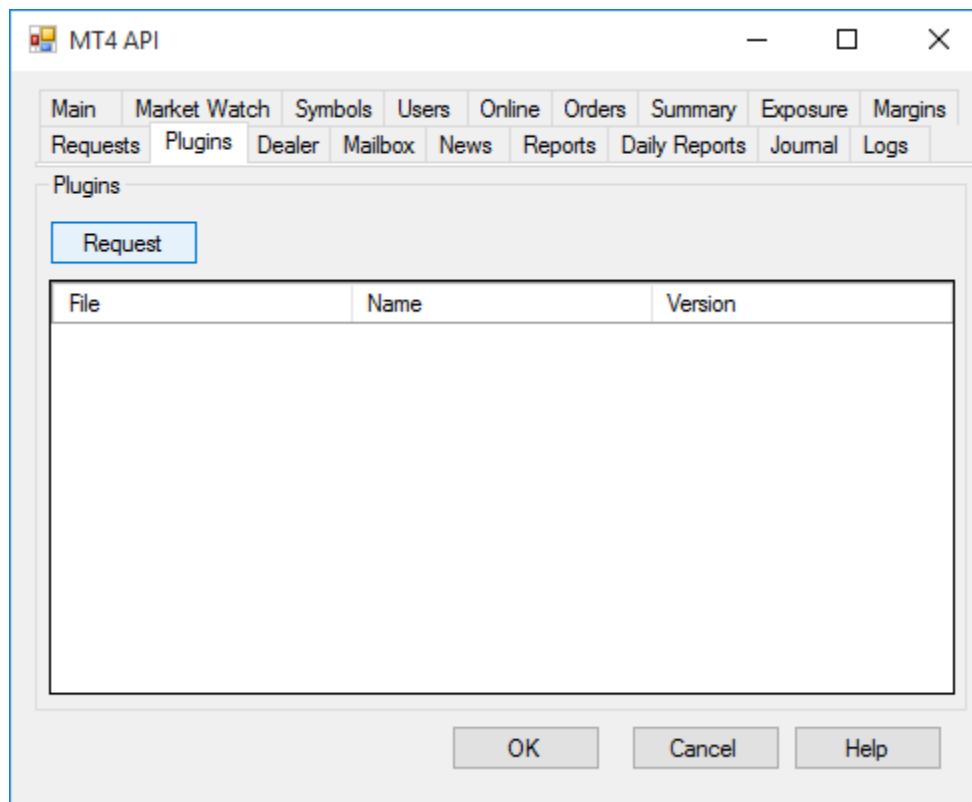


Fig: Plugins Tab.

`PumpingSwitch()` method used for switching into pumping mode.

```
/// <summary>
/// Switch into pumping mode
/// delegate will be invoked on any pumping event
/// </summary>
private void PumpingSwitch()
{
    if (switchFlag == false)
    {
        var are = new AutoResetEvent(false);
        clrWrapper.PumpingSwitch(i =>
        {
            if (i == 0) // 0 - means pumping started
                are.Set();
        });

        are.WaitOne();
        switchFlag = true;
    }
}
```



```

/// <summary>
/// Get list of installed plugins in pumping mode
///
/// </summary>
private void RequestButton_PluginsClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        PumpingSwitch();
        IList<Plugin> plugins = clrWrapper.PluginsGet();
        IList<PluginFxtf> list = new List<PluginFxtf>();

        foreach (var p in plugins)
        {
            string name = p.Info.Name;
            uint version = p.Info.Version;
            string file = p.File;
            list.Add(new PluginFxtf(name, version, file));
        }

        dataGridViewPlugins.DataSource = list;
        dataGridViewPlugins.Visible = true;
    }
    else
    {
        MessageBox.Show(CONNECT_FIRST);
        dataGridViewPlugins.DataSource = null;
    }
}

```

And the `PluginFxtf` class is:

```
/// <summary>
/// Object that represents plugin information configuration
/// </summary>
class PluginFxtf
{
    /// <summary>
    /// Plugin file name
    ///
    /// </summary>
    public string File { get; set; }
    /// <summary>
    /// Plugin name
    ///
    /// </summary>
    public string Name { get; set; }
    /// <summary>
    /// Plugin version
    ///
    /// </summary>
    public uint Version { get; set; }

    /// <summary>
    ///
    /// </summary>
    /// <param name="_name">Text</param>
    /// <param name="_version">Number</param>
    /// <param name="_file">Text</param>
    public PluginFxtf(string _name,uint _version,string _file)
    {
        Name = _name;
        Version = _version;
        File = _file;
    }
}
```

Daily Reports

This tab is for getting list of installed plugins in pumping mode.

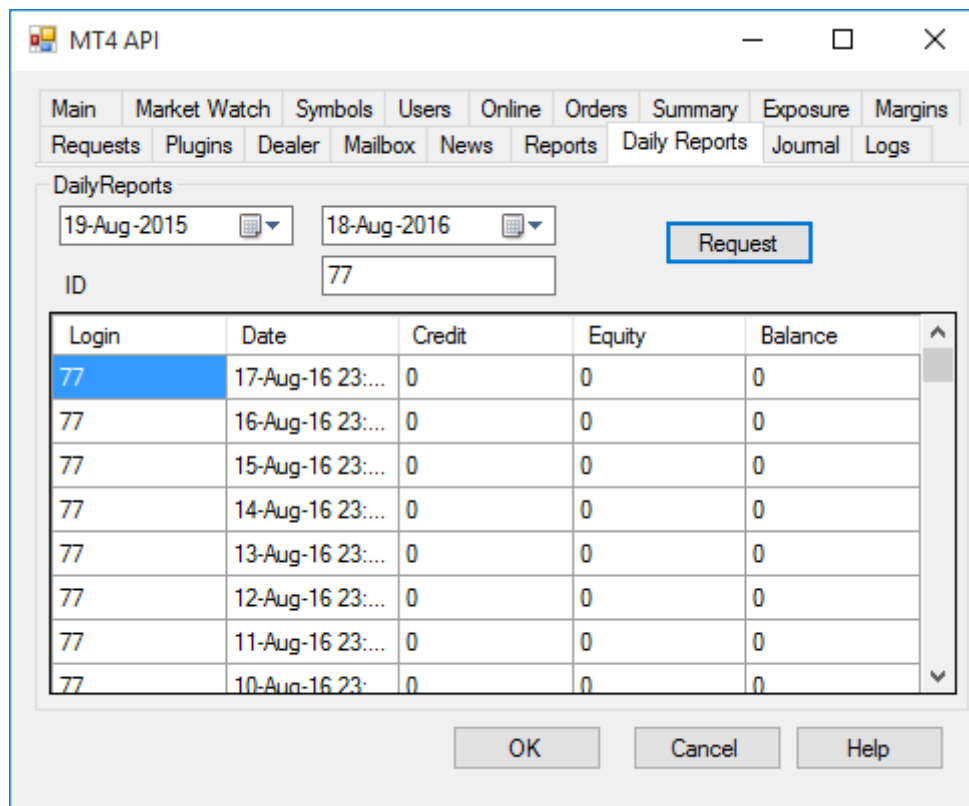


Fig: Daily Reports Tab.

`PumpingSwitch()` method used for switching into pumping mode.

```
/// <summary>
/// Switch into pumping mode
/// delegate will be invoked on any pumping event
/// </summary>
private void PumpingSwitch()
{
    if (switchFlag == false)
    {
        var are = new AutoResetEvent(false);
        clrWrapper.PumpingSwitch(i =>
        {
            if (i == 0) // 0 - means pumping started
                are.Set();
        });

        are.WaitOne();
        switchFlag = true;
    }
}
```

```
}
```

DailyGroupRequest() method is used for get daily reports.

```
/// <summary>
/// Get daily reports
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RequestButton_DailyReportsClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        Login();

        DateTime dt1 = FromDateTimePicker_DailyReports.Value.Date;
        uint from = ToUnixTime(dt1);

        DateTime dt2 = ToDateTimePicker_DailyReports.Value.Date;
        uint to = ToUnixTime(dt2);

        if ((int)to - (int)from >= 86400) //checking one day interval in seconds
        {
            try
            {
                int login = int.Parse(LogintextBox_DailyReports.Text);

                IList<int> listLogin = new List<int>();
                listLogin.Add(login);

                DailyGroupRequest request = new DailyGroupRequest();
                request.To = to;
                request.From = from;
                request.Name = "Daily"; //NEED TO EDIT

                //UserList();

                IList<DailyReport> dailyReportses =
clrWrapper.DailyReportsRequest(request, listLogin);
                IList<DailyReportFxtf> list = new List<DailyReportFxtf>();
                int i = 0;
                foreach (var p in dailyReportses)
                {
                    int log = p.Login;
                    uint time = p.Ctm;
                    DateTime dateTime = ToDateTime(time);
                    double credit = p.Credit;
                    double equity = p.Equity;
                    double balance = p.Balance;

                    list.Add(new DailyReportFxtf(log, dateTime, credit, equity,
balance));
                }
                dataGridView_DailyReports.DataSource = list;
                dataGridView_DailyReports.Visible = true;
            }
            catch { }
        }
    }
}
```

```

    }
    catch (Exception kException)
    {
        const string msg = "Can't parse the ID field.";
        MessageBox.Show(msg);
    }

}
else
{
    const string msg = "From date and To date must be between 1 day.";
    MessageBox.Show(msg);
}
}
else
{
    MessageBox.Show(CONNECT_FIRST);
    dataGridView_DailyReports.DataSource = null;
}
LogIn();
}

```

The 'from date' and the 'to date' must be minimum 1-day interval.

And the `DailyReportFxtf` class is:

```
/// <summary>
/// Object that represents daily report
/// </summary>
class DailyReportFxtf
{
    /// <summary>
    /// Login
    ///
    /// </summary>
    public int Login { get; set; }

    /// <summary>
    /// Time
    ///
    /// </summary>
    public DateTime Date { get; set; }

    /// <summary>
    /// Credit
    ///
    /// </summary>
    public double Credit { get; set; }

    /// <summary>
    /// Equity
    ///
    /// </summary>
    public double Equity { get; set; }

    /// <summary>
    /// Balance
    ///
    /// </summary>
    public double Balance { get; set; }

    /// <summary>
    ///
    /// </summary>
    /// <param name="_Login"></param>
    /// <param name="_dateTime"></param>
    /// <param name="_credit"></param>
    /// <param name="_equity"></param>
    /// <param name="_balance"></param>
    public DailyReportFxtf(int _Login, DateTime _dateTime, double _credit, double
    _equity, double _balance)
    {
        Login = _Login;
        Date = _dateTime;
        Credit = _credit;
        Equity = _equity;
        Balance = _balance;
    }
}
```

Others

This tab is for providing other useful functionalities.

Server Feed

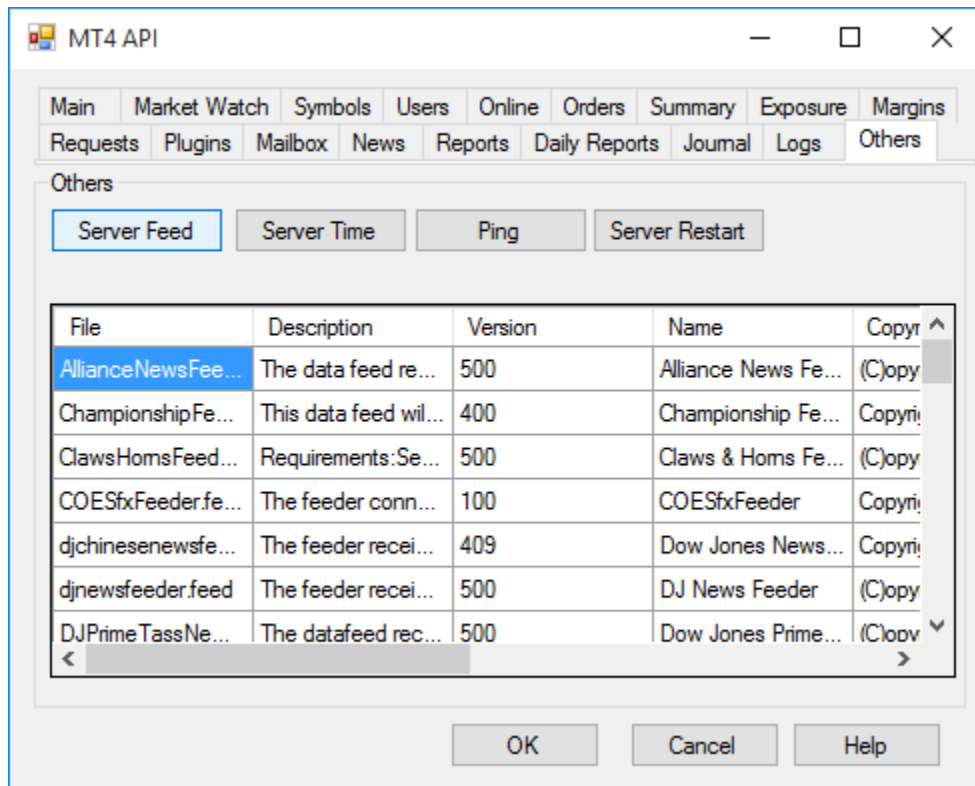


Fig: Server Feed of Others Tab.

```
/// <summary>
/// Request all available on MT server data feeds
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ServerFeedButton_OthersClick(object sender, EventArgs e)
{
    if (clrWrapper.IsConnected() != 0)
    {
        IList<ServerFeed> serverFeeds = clrWrapper.SrvFeeders();
        IList<ServerFeedFxtf> list = new List<ServerFeedFxtf>();

        foreach (var p in serverFeeds)
        {
            string file = p.File;
            string description = p.Feed.Description;
            int version = p.Feed.Version;
```

```

        string name = p.Feed.Name;
        string copyright = p.Feed.Copyright;
        string web = p.Feed.Web;
        string email = p.Feed.Email;
        string server = p.Feed.Server;
        string userName = p.Feed.UserName;

        list.Add(new
ServerFeedFxtf(file,description,version,name,copyright,web,email,server,userName));
    }

    dataGridViewOthers.DataSource = list;
    dataGridViewOthers.Visible = true;

}
else
{
    MessageBox.Show(CONNECT_FIRST);
    dataGridViewOthers.DataSource = null;
}
}
LogIn();
}

```

And the `ServerFeedFxtf` class is:

```

/// <summary>
/// Object that represents server feed
/// </summary>
class ServerFeedFxtf
{
    /// <summary>
    /// Feeder file name
    ///
    /// </summary>
    public string File { get; set; }
    /// <summary>
    /// Feeder description
    ///
    /// </summary>
    public string Description { get; set; }

    /// <summary>
    /// Data source version
    ///
    /// </summary>
    public int Version { get; set; }
    /// <summary>
    /// Data source name
    ///
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Copyright string

```



```

    ///
    /// </summary>
    public string Copyright { get; set; }

    /// <summary>
    /// Data source web
    ///
    /// </summary>
    public string Web { get; set; }
    /// <summary>
    /// Data source email
    ///
    /// </summary>
    public string Email { get; set; }
    /// <summary>
    /// Feeder server
    ///
    /// </summary>
    public string Server { get; set; }
    /// <summary>
    /// Default feeder name
    ///
    /// </summary>
    public string UserName { get; set; }

    /// <summary>
    ///
    /// </summary>
    /// <param name="_file">Text</param>
    /// <param name="_description">Text</param>
    /// <param name="_version">Number</param>
    /// <param name="_name">Text</param>
    /// <param name="_copyright">Text</param>
    /// <param name="_web">Text</param>
    /// <param name="_email">Text</param>
    /// <param name="_server">Text</param>
    /// <param name="_userName">Text</param>
    public ServerFeedFxtf(string _file, string _description, int _version, string
_name, string _copyright, string _web, string _email, string _server, string _userName)
    {
        File = _file;
        Description = _description;
        Version = _version;
        Name = _name;
        Copyright = _copyright;
        Web = _web;
        Email = _email;
        Server = _server;
        UserName = _userName;
    }
}

```

Server Time

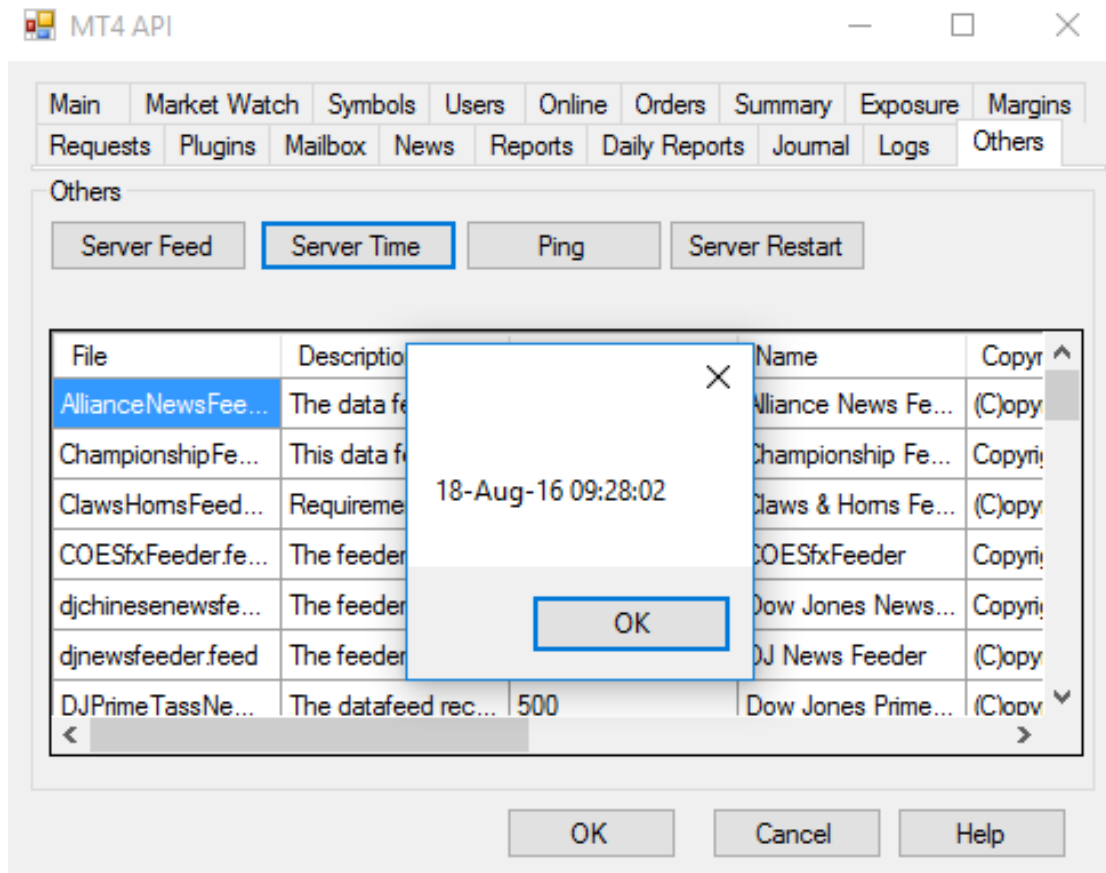


Fig: Server time of Others Tab.

```
private void TimeButton_OthersClick(object sender, EventArgs e)
{
    if (isLoggedIn)
    {
        uint time = (uint)clrWrapper.ServerTime();
        DateTime dateTime = ToDateTime(time);
        MessageBox.Show(dateTime.ToString());
    }
    else
    {
        const string msg = "You aren't logged in.";
        MessageBox.Show(msg);
    }
}
```

Ping

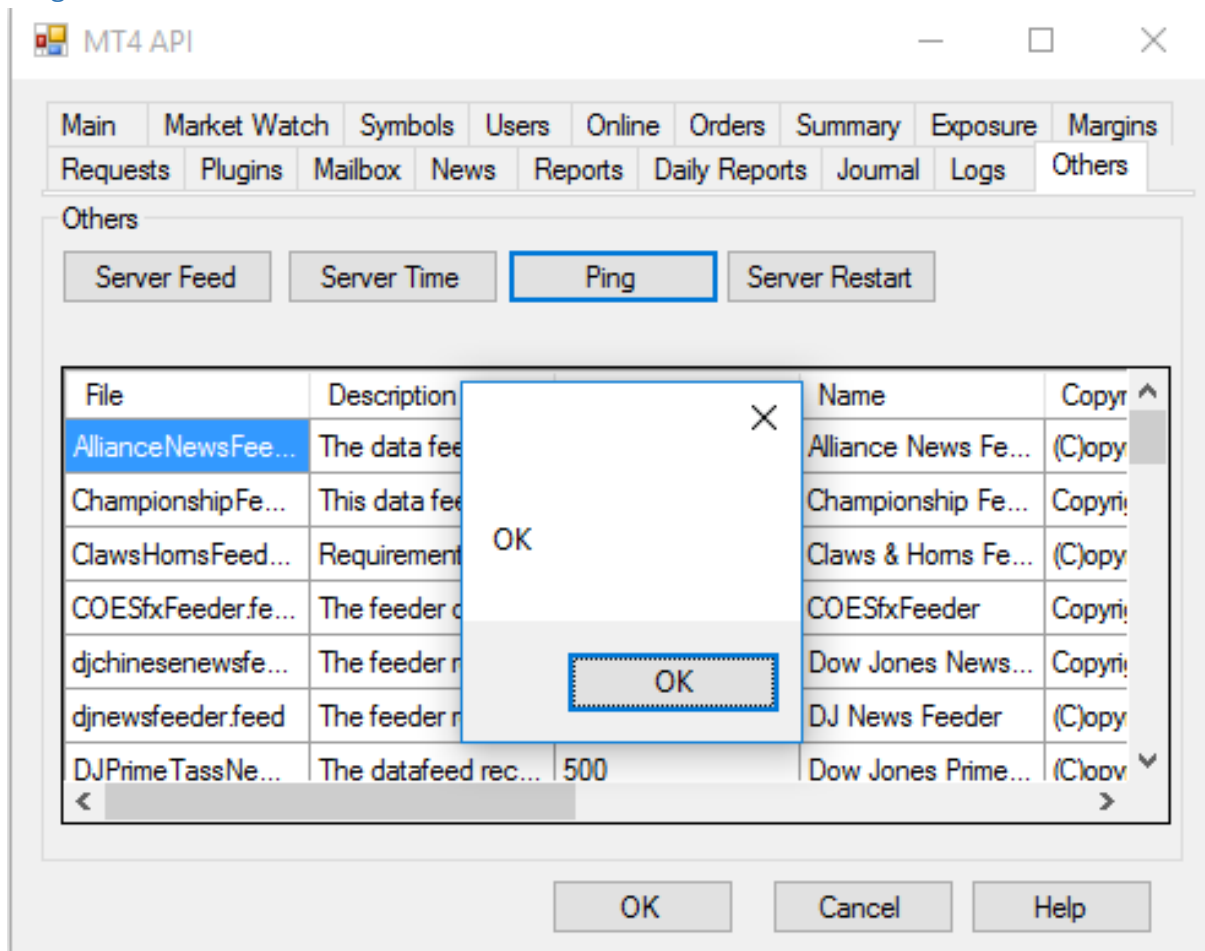


Fig: Ping of Others Tab.

```
private void PingButton_OthersClick(object sender, EventArgs e)
{
    MessageBox.Show(clrWrapper.ErrorDescription(clrWrapper.Ping()));
}
```

Server Restart

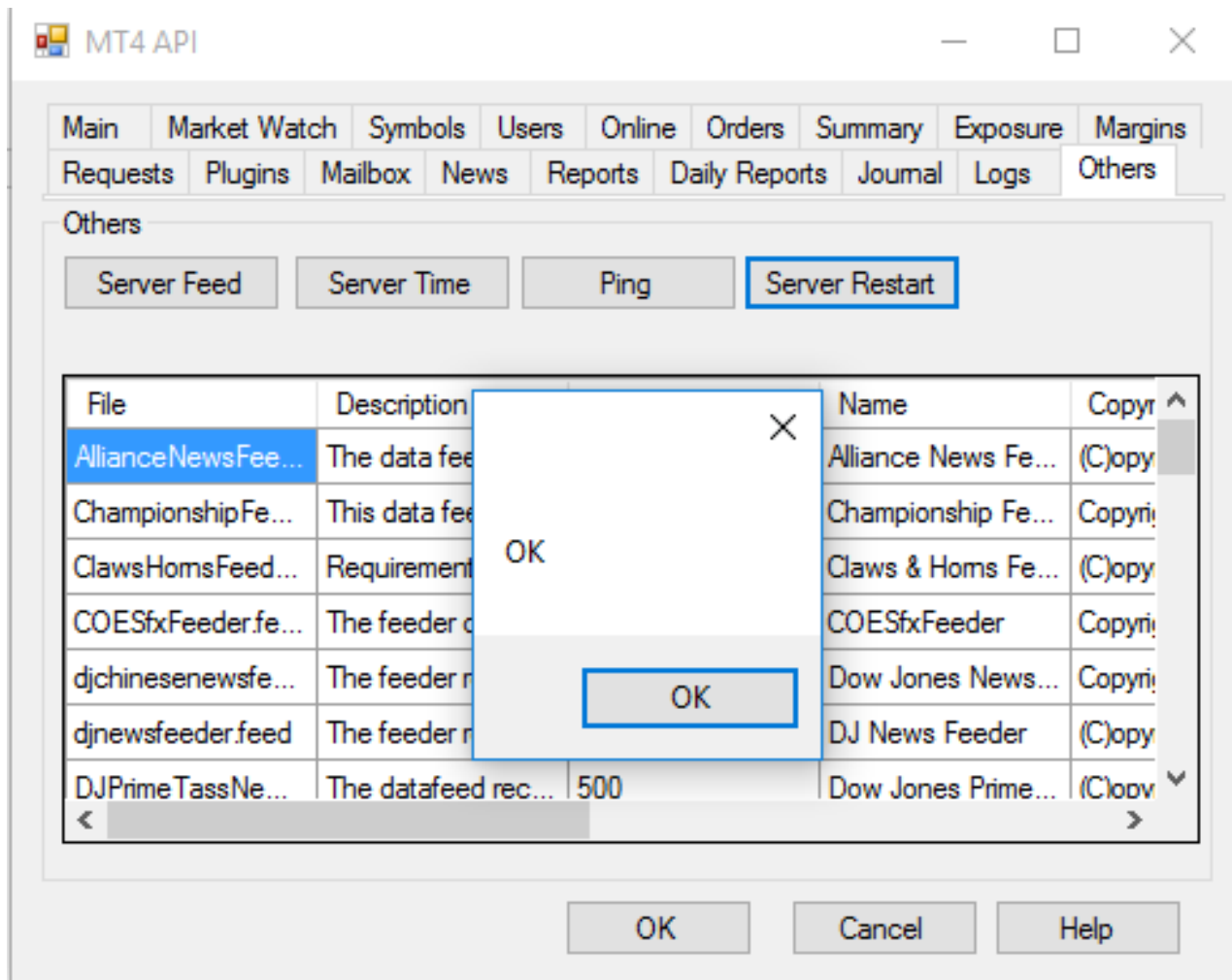
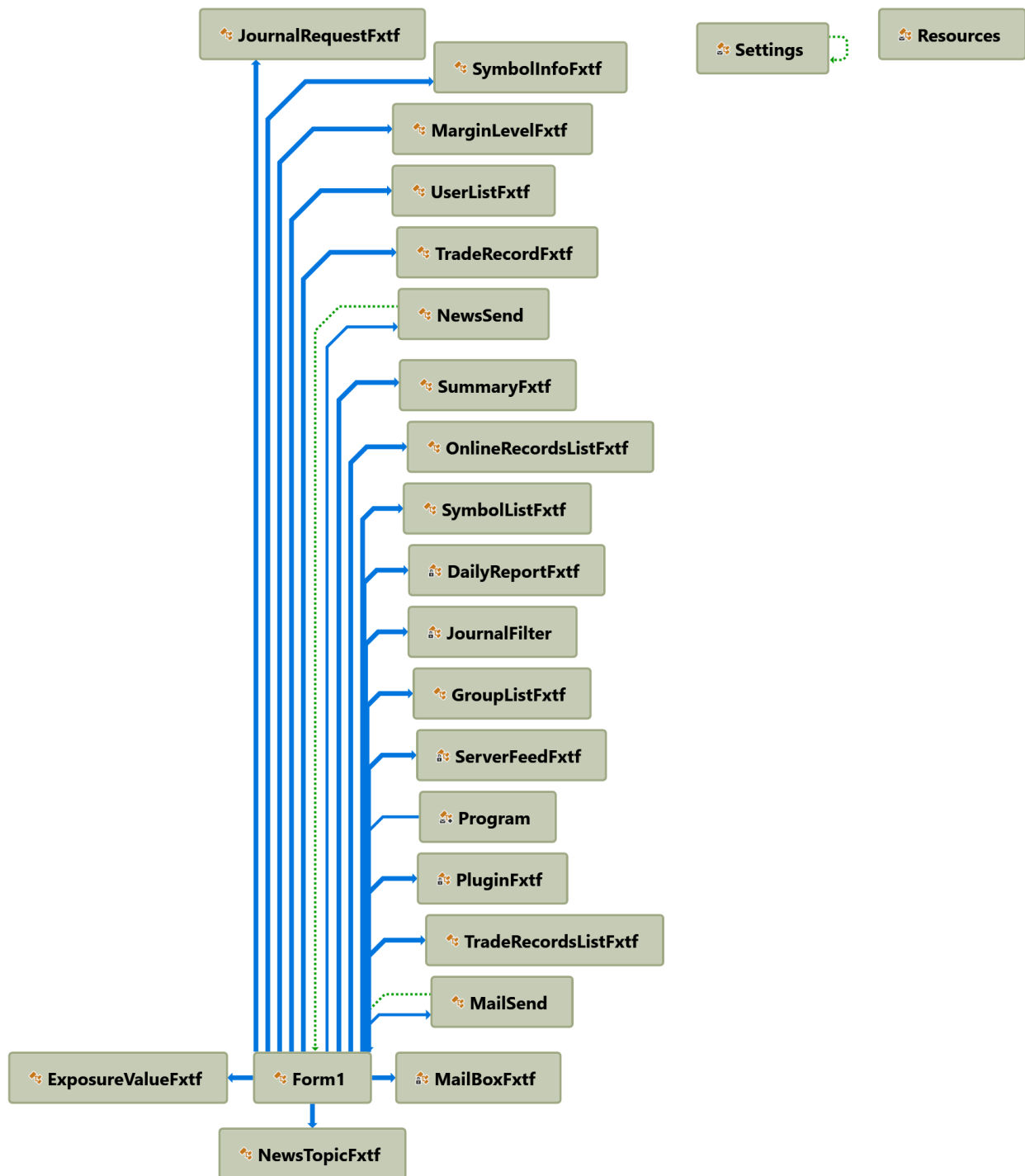


Fig: Server Restart of Others Tab.

```
private void ServerRestartButton_OthersClick(object sender, EventArgs e)
{
    MessageBox.Show(clrWrapper.ErrorDescription(clrWrapper.SrvRestart()));
    isLoggedIn = false;
}
```

Dependencies Graph

The dependencies graph is



Other methods

Name	Purpose
Connection and authorization	
Disconnect()	Allows to disconnect from the server
KeysSend()	At the first connection of the account, the server requires a public key to be sent to it through KeysSend() function. RSA keys can be created with MetaTrader Administrator or MetaTrader Manager. After the key has been sent it is necessary to disconnect and then reconnect and be authorized on the server.
PasswordChange()	One can change one's own manager account password
ServerTime()	Request server time
ManagerRights()	Get one's own rights settings
Ping()	Check connection to the server
Manager interface	
SrvRestart()	Restart MetaTrader Server
SrvChartsSync()	Synchronize history data
SrvLiveUpdateStart()	Start Live Update
SrvFeedsRestart()	Restart data feeds.
SrvFeeders()	All available on the server data feeds can be requested
SrvFeederLog()	Allows to request for the logs of the data feed configured by name
ChartRequest()	Request for history data by symbol and timeframe from the certain moment of time
ChartAdd()	Add bars to the history database
ChartUpdate()	Update bars in the history database
ChartDelete()	Delete bars from the history database
administrator functions	
AdmUsersRequest()	Request for accounts from the current database; the list of groups or accounts separated by commas can be specified as the request string;
AdmTradesRequest()	Request for orders of the current database; list of groups, accounts, or orders, separated by commas, can be specified as the request string;
AdmTradesDelete()	Deleting of orders from the current database;
AdmTradeRecordModify()	Modifying of an order in the current database;
AdmBalanceCheck()	Checking of balance of the account list;

AdmBalanceFix()	Correcting of balance of the account list according to the trade history.
database backups	
BackupInfoUsers()	Request for the file list of backups of the account database;
Symbols	
SymbolsRefresh()	The list of available symbols can be requested from the server
SymbolsGetAll() / SymbolGet()	Settings of the requested symbols or of a specific symbol can be obtained with the following functions
Direct access to the server databases	
GroupsRequest()	Request a list of available groups of accounts
UserRecordsRequest()	The list of certain accounts can be requested
OnlineRequest()	The list of the connected clients can be requested
UserRecordNew()	Allows to select a new account
UserRecordUpate()	Allows to select a new account
UserRecordUpate()	Allows to change these accounts
UserGroupOp()	Allows to conduct a group operation over the list of accounts
UserPasswordCheck()	Allows to check the account password
UserPasswordSet()	Intended for changing of the main or investor's password of the account and, if Advanced Security mode is enabled, allows to reset the public key on the server, at the next connection, the server will request a new public key from the client.
TradesRequest()	The list of all orders can be requested
TradeRecordsRequest()	The list of certain orders can be requested
TradesUserHistory()	Help to request the trading history for an account
TradeTransaction()	Manager interface allows to open an order, to close an order, or to modify an open order
TradeCheckStops()	Function allows to check the Stop Loss and Take Profit levels of an order, as well as the pending order open price transferred to TradeTransInfo

ReportsRequest()	Manager interface allows to request for the list of closed positions at which reports can be generated for a certain set of accounts
DailyReportsRequest()	Daily reports generated on the server for the list of accounts can be requested. When reports are requested, the report period (from-to fields of ReportGroupRequest and DailyGroupRequest) must be conformed the beginning and end of the server trading day (ConCommon::endhour,ConCommon::endminute).
MailsRequest()	Request last mails of internal mail system
MailSend()	Send a message by the internal mailing system
NewsSend()	Throw in a news into the news flow
PluginUpdate()	Configuration of the server plugin can be changed
JournalRequest()	Allows to request for the server log for a certain period of time

References

- I. <https://github.com/Uriil/MetaTrader4.Manager.Wrapper/>