Contents

[OECTrader 2](#_Toc319359332)

[Пример работы с OECTrader 2](#_Toc319359333)

[Стоп-заявки 5](#_Toc319359334)

[Remoting 5](#_Toc319359335)

[Настройка терминала OEC Trader для работы в режиме Primary 6](#_Toc319359336)

## OECTrader

Для написания торговых роботов для OpenECry необходимо использовать адаптер OECTrader (не путать с названием терминала “OEC Trader”). Механизм взаимодействия показан на данном рисунке:



Как видно из рисунка, OECTrader взаимодействует с сервером OEC посредством OEC API (<http://www.openecry.com/myaccountmgm/download.cfm>). Для использования OEC API не требуется наличие работающего терминала OEC Trader.

## Пример работы с OECTrader

В качестве примера работы с OECTrader рассмотрим приложение SampleOEC, показанное на рисунке ниже. Исходные коды примера лежат в дистрибутиве в папке под названием SampleOEC.



1. В самом начале необходимо создать экземпляр класса шлюза OECTrader:

Trader = **new** OECTrader {

 Login = Login.Text,

 Password = Password.Password,

 OECConnectionAddress = address,

 OECConnectionPort = port,

 };

1. Далее следует установить требуемые обработчики событий ITrader:

// событие восстановления соединения

Trader.ReConnectionSettings.ConnectionRestored += () => **this**.GuiAsync(() =>

{

 ChangeConnectStatus(**true**);

 MessageBox.Show(**this**, "Соединение восстановлено.");

});

// событие успешного соединения

Trader.Connected += () =>

{

 \_isConnected = **true**;

 **this**.GuiAsync(() => ChangeConnectStatus(**true**));

};

// событие разрыва соединения

Trader.ConnectionError += error => **this**.GuiAsync(() =>

{

 ChangeConnectStatus(**false**);

 MessageBox.Show(**this**, error.ToString(), "Ошибка соединения");

});

// событие успешного отключения

Trader.Disconnected += () => **this**.GuiAsync(() => ChangeConnectStatus(**false**));

// событие ошибки обработки данных

Trader.ProcessDataError += error => **this**.GuiAsync(() => MessageBox.Show(**this**, error.ToString(), "Ошибка обработки данных"));

// событие получения новых инструментов

Trader.NewSecurities += securities => **this**.GuiAsync(() => \_securitiesWindow.AddSecurities(securities));

// событие получения новых собственных сделок

Trader.NewMyTrades += trades => **this**.GuiAsync(() => \_myTradesWindow.Trades.AddRange(trades));

// событие получения новых сделок по подписанным инструментам

Trader.NewTrades += trades => **this**.GuiAsync(() => \_tradesWindow.Trades.AddRange(trades));

// событие получения новых заявок

Trader.NewOrders += orders => **this**.GuiAsync(() => \_ordersWindow.Orders.AddRange(orders));

// событие получения новых стоп-заявок

Trader.NewStopOrders += orders => **this**.GuiAsync(() => \_stopOrdersWindow.Orders.AddRange(orders));

// событие получения новых портфелей

Trader.NewPortfolios += portfolios =>

{

 portfolios.ForEach(Trader.RegisterPortfolio);

 \_portfoliosWindow.Portfolios.AddRange(portfolios);

};

// событие получения новых позиций

Trader.NewPositions += positions => **this**.GuiAsync(() => \_positionsWindow.Positions.AddRange(positions));

// событие неудачной регистрации заявок

Trader.OrdersRegisterFailed += OrdersFailed;

// событие неудачного снятия заявок

Trader.OrdersCancelFailed += OrdersFailed;

// событие неудачной регистрации стоп-заявок

Trader.StopOrdersRegisterFailed += OrdersFailed;

// событие неудачного снятия стоп-заявок

Trader.StopOrdersCancelFailed += OrdersFailed;

1. Далее необходимо выполнить подключение к серверу OEC:

Trader.Connect();

1. Следующий этап – запустить экспорт данных:

**private** **void** Export\_Click(**object** sender, RoutedEventArgs e)

{

 Trader.StartExport();

}

После запуска экспорта можно ожидать срабатывания событий:

* Trader.SecuritiesChanged – изменение инструмента (последней цены, цены спроса/предложения и т.д.) запрошенного через RegisterSecurity.
* Trader.QuotesChanged – изменение стакана, запрошенного с помощью RegisterQuotes.
* Trader.NewTrades – новые тики по подписанным с помощью RegisterTrades инструментам.
* Trader. PortfoliosChanged – изменение портфелей, запрошенное с помощью RegisterPortfolio.
1. Важным отличием адаптера OECTrader от других реализаций ITrader является необходимость явного запроса инструментов с помощью метода OECTrader.SecurityLookup(). Следующий этап – запрос необходимых инструментов:



**private** **void** SearchText\_PreviewKeyUp(**object** sender, KeyEventArgs e)

{

 **if**(e.Key != Key.Enter) **return**;

 var text = SearchText.Text.Trim();

 **if**(**string**.IsNullOrEmpty(text)) **return**;

 **MainWindow.Instance.Trader.SecurityLookup(text);**

}

Метод SecurityLookup() позволяет запрашивать инструменты по началу имени символа или описания (например «ES» или «e-mini» или «gold») или по точному названию (например «esh3»). Кроме того, метод позволяет задавать следующие (необязательные) дополнительные параметры:

* Фильтр типа инструмента. По умолчанию возвращаются все поддерживаемые типы.
* Название биржи. Примеры: CBT, CME.
* Желаемое количество результатов запроса.

После получения запрошенных инструментов будет вызван обработчик события Trader.OnNewSecurities. В случае, если ни одного инструмента по запросу не найдено, будет вызван обработчик события Trader.ProcessDataError.

1. Для получения обновлений по выбранному инструменту и тиков необходимо воспользоваться методами RegisterSecurity и RegisterTrades соответственно:

**private** **void** BidAsk\_Click(**object** sender, RoutedEventArgs e)

{

 var security = SelectedSecurity;

 var trader = MainWindow.Instance.Trader;

 **if** (\_bidAskSecurities.Contains(security))

 {

 // останавливаем обновления по инструменту

 trader.UnRegisterSecurity(security);

 trader.UnRegisterTrades(security);

 \_bidAskSecurities.Remove(security);

 }

 **else**

 {

 // начинаем получать обновления по инструменту

 trader.RegisterSecurity(security);

 trader.RegisterTrades(security);

 \_bidAskSecurities.Add(security);

 }

}

1. Для запроса обновлений по стаканам необходимо воспользоваться методом RegisterQuotes:

**private** **void** Quotes\_Click(**object** sender, RoutedEventArgs e)

{

 var window = \_quotesWindows.SafeAdd(SelectedSecurity, security =>

 {

 // начинаем получать котировки стакана

 MainWindow.Instance.Trader.RegisterQuotes(security);

 // создаем окно со стаканом

 var wnd = **new** QuotesWindow { Title = security.Code + " котировки" };

 wnd.MakeHideable();

 **return** wnd;

 });

 **if** (window.Visibility == Visibility.Visible)

 window.Hide();

 **else**

 window.Show();

}

1. Для получения обновлений по портфелям используется метод RegisterPortfolio:

Trader.NewPortfolios += portfolios =>

{

 // регистрирует портфели на обновление данных

 portfolios.ForEach(Trader.RegisterPortfolio);

 \_portfoliosWindow.Portfolios.AddRange(portfolios);

};

1. Для регистрации заявок предусмотрена кнопка «Новая заявка». Метод RegisterOrder отправляет заявку на сервер:

**private** **void** Send\_Click(**object** sender, RoutedEventArgs e)

{

 var order = **new** Order

 {

 Type = IsMarket.IsChecked == **true** ? OrderTypes.Market : OrderTypes.Limit,

 Portfolio = Portfolio.SelectedPortfolio,

 Volume = Volume.Text.To<**decimal**>(),

 Price = Price.Text.To<**decimal**>(),

 Security = Security,

 Direction = IsBuy.IsChecked == **true** ? OrderDirections.Buy : OrderDirections.Sell,

 };

 **MainWindow.Instance.Trader.RegisterOrder(order);**

 DialogResult = **true**;

}

### Стоп-заявки

Для регистрации стоп-заявок используется тот же метод RegisterOrder, но с указанием стоп-условия OECStopCondition и типа заявки OrderTypes.Conditional:

**private** **void** Send\_Click(**object** sender, RoutedEventArgs e)

{

 var stopOrder = **new** Order

 {

 Portfolio = Portfolio.SelectedPortfolio,

 Type = OrderTypes.Conditional,

 Volume = Volume.Text.To<**decimal**>(),

 Price = Price.Text.To<**decimal**>(),

 Security = Security,

 Direction = IsBuy.IsChecked == **true** ? OrderDirections.Buy : OrderDirections.Sell,

 ExpiryDate = IsOneDay.IsChecked == **true** ? DateTime.Today : DateTime.MaxValue,

 StopCondition = **new** OECStopCondition(StopMarket ? OECStopType.StopMarket : OECStopType.StopLimit, StopPrice.Text.To<**decimal**>())

 };

 MainWindow.Instance.Trader.RegisterOrder(stopOrder);

 DialogResult = **true**;

}

Для работы с условиями стоп-заявок адаптер OECTrader предоставляет специальный класс OECStopCondition. Данным классом поддерживаются следующие типы стоп-заявок:

* OECStopType.StopMarket – после достижения стоп-цены автоматически выставляется рыночная заявка.
* OECStopType.StopLimit – после достижения стоп-цены автоматически выставляется лимитная заявка.
* OECStopType.TrailingStopMarket – cтоп-цена автоматически следует за рынком, но только в выгодном для позиции направлении, оставаясь на заранее заявленном интервале от рыночной цены. В случае, если рынок достигает стоп-цены, автоматически выставляется рыночная заявка.
* OECStopType.TrailingStopLimit – Как TrailingStopMarket, но при достижении стоп-цены выставляется лимитная заявка.

## Remoting

В зависимости от конкретного пользователя/приложения сервер OEC может не поддерживать одновременное соединение нескольких приложений. В этом случае, если существуют другие соединения, они могут быть разорваны. Для обхода этого ограничения данная реализация OECTrader поддерживает одновременную работу нескольких приложений через одно соединение с OEC сервером – Remoting. Подробное описание функциональности Remoting может быть найдено по адресу <http://www.openecry.com/api/OECAPIRemoting.pdf>.

Поддерживаются следующие режимы Remoting:

* None - Remoting отключен. Приложение создает собственное соединение с сервером OEC. Приложение не может выступать как Primary для других приложений.
* Auto – В момент инициализации выполняется поиск локальных приложений, работающих в режиме Primary. Если такие приложения найдены, то приложение переходит в режим Secondary и использует соединение Primary приложения. В противном случае приложение переходит в режим Primary.
* Primary – приложение создает собственное соединение с сервером OEC.
* Secondary - В момент инициализации выполняется поиск локальных приложений, работающих в режиме Primary. Если такие приложения найдены, используется их соединение с сервером OEC. В противном случае приложение переходит в режим None.

Для явного задания режима Remoting необходимо сразу после создания объекта OECTrader указать требуемый режим. Например, для указания режима Secondary:

Trader.RemotingRequested = OECRemoting.Secondary;

По умолчанию адаптер OECTrader работает в режиме OECRemoting.Auto.

### Настройка терминала OEC Trader для работы в режиме Primary

При необходимости можно настроить терминал OEC Trader для работы в режиме Primary, после чего запустить робота S# в режиме Auto/Secondary. В этом случае терминал и робот будут использовать одно и то же соединение с сервером OEC. Для настройки следует отметить пункт меню File -> Allow Remoting в терминале OEC Trader:

