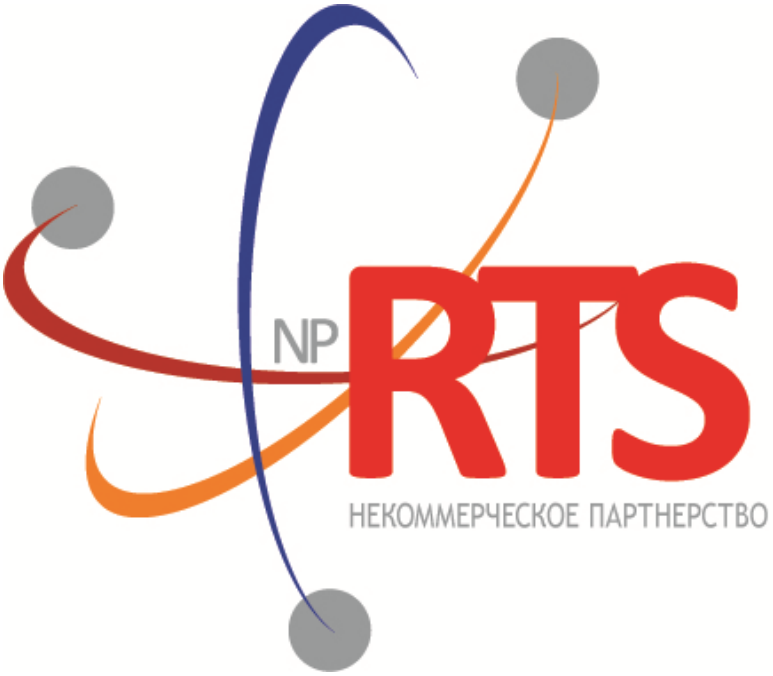

RTS Board ClienAPI

Методические указания
по использованию библиотеки ClientAPI системы Quatro

Версия системы 1.7

Версия документа 1.0.4

05.02.2018

Содержание

Список примеров

1. Описание интерфейса

Интерфейс, к которому относятся данные указания, представлен в виде библиотеки Microsoft .NET версии 4.5. Использование асинхронных операций и Microsoft SignalR client не позволяет обеспечить обратную совместимость с фреймворками ниже указанной версии.

Версия API в виде библиотеки позволяет получить более легкий доступ к данным и операциям системы Quatro, без необходимости реализовывать транспортный уровень взаимодействия и описывать классы-контракты.

Базовым для осуществления взаимодействия является класс OtcConnection, содержащий, в качестве свойств, все доступные для взаимодействия с системой Quatro программные интерфейсы. Создание экземпляра класса OtcConnection производится с передачей экземпляра класса OtcConnectionSettings, содержащего в себе сведения, необходимые для установления соединения.

2. Общие принципы работы с API

API условно можно разделить на два блока – блок работы с данными по запросу (REST) и блок получения данных по подписке (SignalR). Зачастую, для составления правильной иерархии объектов на стороне клиента, необходимо использовать подходы совместно. В качестве примера можно привести операции над сделками. Для того чтобы понять, какие сделки есть у клиента в системе, необходимо подписаться на поток GenericTrades (подписка на потоки будет рассмотрена в разделе 4) и, на основании полученных данных (ID сделки, текущий офис и статус), запросить сведения об интересующем документе через функционал REST. Таким образом, обеспечивается минимальная достаточность данных, поставляемых пользователю по подписке.

3. Установление и поддержка соединения

Для установления и поддержки соединения используются следующие методы класса OtcConnection:

- ConnectAsync – метод, используемый для подключения к системе (осуществляется одновременное подключение как к REST API, так и к SignalR DataFeed). Данный метод асинхронный и может принимать в качестве опционального параметра токен отмены операции;
- ReconnectFeedAsync – метод, позволяющий осуществить переподключение к SignalR DataFeed. Метод асинхронный и может принимать в качестве опционального параметра токен отмены операции;
- RefreshAccessTokenAsync – метод, позволяющий обновить токен авторизации после длительного бездействия пользователя (в течение более чем 5 минут не осуществлялся вызов REST и потоки данных были отключены)

Фрагмент кода подключения к системе расположен в приложении (см. пример 1).

4. Операции с потоками данных

В зависимости от желаемых данных необходимо выбрать конкретный SignalR поток и осуществить к нему подключение, путем вызова подписчика (SubscribeAsync) соответствующего класса (например, для публичных котировок это будет OtcConnection.Quotes.SubscribeToPublicQuotesAsync). Ниже предоставлен перечень возможных потоков с сопоставлением их свойствам класса OtcConnection:

- GenericTrades – поток общих сделок. В системе любая сделка, кроме рыночной, является производной от сущности GenericTradeContract. Сущность GenericTradeContract содержит в себе глобальные параметры сделки, в том числе ее идентификатор в соответствующем офисе, ее ревизию и статус. В приложении предоставлен вариант обработки полученной сущности GenericTrade и запроса через REST соответствующей дочерней сущности (см. пример 3);
- MarketTrades – поток рыночных сделок. Только публичные сделки попадают в данный поток;
- PublicQuotes – поток публичных котировок, включающий все доступные данному логину котировки. Также в этом потоке находятся адресные котировки, выставленные в адрес организации, которой принадлежит данный логин;
- OwnQuotes – поток собственных котировок, включающий все собственные котировки, включая черновики;
- SettlementSchemes.Schemes – расчетные схемы;
- SettlementAccounts.SettlementAccounts – банковские счета;
- DepoAccounts – депозитарные счета;
- Parties – таблица участников;
- Alerts – поток уведомлений;
- Chat – поток диалога с контрагентами. Также можно подписаться на поток таблицы чатов при помощи метода SubscribeToConversationListAsync.

5. Операции с документами

Практически все бизнес-сущности в системе интерпретируются как документы. Это позволяет унифицировать подходы к обработке. Основными параметрами для идентификации любого документа (класс BaseDocumentContract) является идентификатор Id (строковый параметр, не подлежащий сортировке по возрастанию/убыванию) и ревизия Revision (числовой параметр с автоматическим увеличением после внесения каких-либо изменений в документ). Также в любом документе имеется сокращенное или полное описание текущего статуса State (класс, свойствами которого являются идентификатор статуса, возможные действия над документом, описание ошибок в документе).

Однако, т.к. система потенциально способна поддерживать любые финансовые инструменты, в том числе и не имеющие аналога цены, описание продукта, над которым совершается операция, вынесено в отдельный XML-блок, являющийся частью документа. Данный блок хранится в свойстве Content.

6. Описание основных бизнес-процессов системы

6.1. Создание котировки

Необходимые для действия данные:

- Текущий аккаунт (для заполнения поля PartyId)
- Справочник валют (для заполнения валюты цены и валюты расчетов)
- Инструмент (для заполнения контента и идентификатора инструмента)

Операции над объектами:

После создания объекта OTC.ClientAPI.Documents.Quotes.QuoteContract и заполнения его необходимыми данными, вызывается метод OtcConnection.Quotes.CreateAsync, в качестве параметра которого передается контракт котировки. В случае успеха в системе создается котировка, имеющая статус «DRAFT». Возвращаемый методом объект содержит в себе полное описание котировки, включая ее идентификатор и номер ревизии. Для активации необходимо вызвать метод OtcConnection.Quotes.SendAsync, в котором передается идентификатор и номер ревизии котировки.

6.2. Обновление и удаление котировки

Необходимые для действия данные:

- Объект котировки

Операции над объектами:

Для отмены котировки вызывается метод OtcConnection.Quotes.CancelAsync, параметрами которого являются идентификатор и номер ревизии котировки. Для обновления котировки вызывается метод OtcConnection.Quotes.UpdateAsync, в качестве параметра которого передается идентификатор, номер текущей ревизии и обновленный объект котировки.

6.3. Создание договора

Необходимые для действия данные:

- Справочник валют;
- Справочник инструментов;
- Справочник контрагентов;
- Справочник расчетных схем;
- Справочник клиентов (если операция производится с указанием клиента);
- Справочник типов клиентских взаимодействий
- Текущий аккаунт

Операции над объектами:

После создания объекта OTC.ClientAPI.Documents.BackTrades.BackTradeContract и заполнения его необходимыми данными, вызывается метод OtcConnection.BackTrades.CreateAsync, в качестве параметра которого передается контракт договора. В случае успеха в системе создается договор в статусе «NEGOTIATING». Возвращаемый методом объект содержит в себе полное описание договора, включающее его идентификатор, номер ревизии и XML-контент, в который помещены все сведения о договоре. Так же договор становится доступным контрагенту.

6.4. Обновление/заполнение договора

Необходимые для действия данные:

- Объект договора

Операции над объектами:

Для обновления договора вызывается метод OtcConnection.BackTrades.UpdateAsync, в качестве параметров которого передается идентификатор, номер текущей ревизии и обновленный объект договора. Существует несколько важных моментов при совершении обновления договора. Первым является общий принцип работы с сущностями, доступными для параллельного редактирования как агентом, так и контрагентом. Перед началом любого изменения сущности необходимо заблокировать ее, чтобы исключить ситуацию «одновременного» изменения. Данное действие для сделок бэк-офиса производится с использованием метода OtcConnection.BackTrades.LockAsync. В качестве параметров передаются идентификатор и номер текущей ревизии документа. После исполнения метода документ блокируется от редактирования для всех, кроме логина, вызвавшего данный метод. Необходимо понимать, что при разработке приложения необходимо предусмотреть внештатные ситуации и при не ожидаемом выходе из процесса обновления требуется разблокировать договор для возможности дальнейшей работы с ним. Разблокировка производится вызовом метода OtcConnection.BackTrades.UnlockAsync, параметрами которого являются идентификатор и номер текущей ревизии документа. В случае успешного обновления создается новая ревизия документа, и разблокировать требуется уже новую ревизию, т.к. именно она является текущей. Данные о номере ревизии содержатся в контракте ответа на запрос обновления договора. Вторым важным моментом является, что запрашивать договор необходимо с флагом OTC.ClientAPI.Documents.ContentInclusionMode.DocumentInternals. Данный флаг означает, что сервер вернет не полный экземпляр XML договора, а только необходимую для редактирования часть. Внесение изменений в полный XML договора не предусмотрено, т.к. он формируется на сервере.

7. Настройка примера QuatroExpo перед использованием

На нашем ftp-сервере опубликован пример работы с библиотекой ClientAPI системы Quatro. Скачать его вы можете по ссылке. Для начала работы с примером необходимо будет внести следующие изменения в конфигурационные файлы:

- В файле OtcConnectionSettings.xml указать логины и пароли трейдера и бэк-офицера
- В файле QuatroExpo.exe.config указать правильную строку соединения с базой данных.

Приложение A. Примеры

Пример 1. Подключение к системе

```
 public async Task Connect(CancellationToken ct)
   {
        OtcConnectionSettings settings = new OtcConnectionSettings();

        OtcCredential cred = OtcCredential.Plain(<логин>, <пароль>);
        settings.EnableSignalRTracing = true;
        settings.EnableFeedMessageTracing = true;
        settings.Transports = DataFeedTransport.All;
        OtcEndpoint ep = new OtcEndpoint(<адрес полигона, например http://test.rtsboard.ru>);
        settings.Endpoint = ep;
        settings.Credential = cred;
        Connection = new OtcConnection(settings);
        Connection.FeedConnectionStateChanged += Connection_FeedConnectionStateChanged;
        Connection.RestConnectionStateChanged += Connection_RestConnectionStateChanged;
        await Connection.ConnectAsync(ct);
   }
```

Пример 2. Подключение к потокам рыночных данных

```
async Task SubscribeFeeds(string subscribeString)
 {
   if (Subscriptions == null)
       Subscriptions = new Dictionary<string, IDisposable>();
   else
   {
       UnsubscribeFeeds();
       Subscriptions.Clear();
   }

   foreach (string s in _settings.EnabledFeeds.Split(';'))
   {
     //"PublicQuotes;OwnQuotes;FrontTrades;GenericTrades;Alerts";
     switch (s)
       {
        case "Quote":
          if (_settings.ConnectionType == OtcConnectionType.BACKOFFICER)
          {
           Log.OtcInteraction.Warn("Подписка на поток публичных котировок с ролью BACKOFFICER может
            привести к ошибкам в данных. Необходимо исправить конфигурацию");
          }
          SubscriptionListener<OTC.ClientAPI.Documents.Quotes.QuoteContract> pubQuotesListener
           = new SubscriptionListener<OTC.ClientAPI.Documents.Quotes.QuoteContract>(ProcessPublicQuoteRecord,s);

          OTC.ClientAPI.DataFeed.ISubscription<OTC.ClientAPI.Documents.Quotes.QuoteContract> pq
           = await Connection.Quotes.SubscribeToPublicQuotesAsync(pubQuotesListener, s, null);

          Subscriptions.Add(s, pq);
        break;
        case "OwnQuote":
           if (_settings.ConnectionType == OtcConnectionType.BACKOFFICER)
            {
             Log.OtcInteraction.Warn("Подписка на поток собственных котировок с ролью BACKOFFICER может
              привести к ошибкам в данных. Необходимо исправить конфигурацию");
            }
           SubscriptionListener<OTC.ClientAPI.Documents.Quotes.QuoteContract> ownQuotesListener
             = new SubscriptionListener<OTC.ClientAPI.Documents.Quotes.QuoteContract>(ProcessOwnQuoteRecord, s);

           OTC.ClientAPI.DataFeed.ISubscription<OTC.ClientAPI.Documents.Quotes.QuoteContract> oq
              = await Connection.Quotes.SubscribeToOwnQuotesAsync(ownQuotesListener, s, null);

            Subscriptions.Add(s, oq);
        break;
        case "MarketTrade":
          if (_settings.ConnectionType == OtcConnectionType.BACKOFFICER)
             {
              Log.OtcInteraction.Warn("Подписка на поток рыночных сделок с ролью BACKOFFICER может
              привести к ошибкам в данных. Необходимо исправить конфигурацию");
             }
            MarketTrade trade = db.GetMaxValueRecord<MarketTrade>("TradeDate", "MarketTrade");
            OTC.ClientAPI.Documents.MarketTrades.FilterParameters filter = new
OTC.ClientAPI.Documents.MarketTrades.FilterParameters();

          if (trade != null)
```

```
            {
              DatetimeRange dtr = new DatetimeRange();
              dtr.From = trade.TradeDate;
              filter.Date = dtr;
            }
          else
           {
             filter = null;
           }
            SubscriptionListener<OTC.ClientAPI.Documents.MarketTrades.MarketTradeContract> marketTradesListener
             = new
SubscriptionListener<OTC.ClientAPI.Documents.MarketTrades.MarketTradeContract>(ProcessMarketTradeRecord, s);

             OTC.ClientAPI.DataFeed.ISubscription<OTC.ClientAPI.Documents.MarketTrades.MarketTradeContract> mt
             = await Connection.MarketTrades.SubscribeAsync(marketTradesListener, s, filter);

             Subscriptions.Add(s, mt);
        break;
        case "GenericTrade":

         SubscriptionListener<OTC.ClientAPI.Documents.GenericTrades.GenericTradeContract> genericTradesListener
         = new
SubscriptionListener<OTC.ClientAPI.Documents.GenericTrades.GenericTradeContract>(ProcessGenericTradeRecord, s);

           OTC.ClientAPI.DataFeed.ISubscription<OTC.ClientAPI.Documents.GenericTrades.GenericTradeContract> gt =
           await Connection.GenericTrades.SubscribeAsync(genericTradesListener, s, null);
           Subscriptions.Add(s, gt);
        break;
       }
   }
}
```

Пример 3. Обработка потока GenericTrade

```
async Task GenericTradeProcessing(OTC.ClientAPI.Documents.GenericTrades.GenericTradeContract data, string recID)
 {
   switch (data.State.Split(':')[0])
   {
       case "FRONT":
           if (_settings.ConnectionType == OtcConnectionType.TRADER)
           {
               try
               {

OTC.ClientAPI.Documents.Extensions.GetDocumentResponse<OTC.ClientAPI.Documents.FrontTrades.FrontTradeContract> f
                       = await Connection.FrontTrades.GetAsync(data.ActiveId);
                   if (f.Success)
                   {
                       FrontTrade ft = QuatroMapping.GetObject<FrontTrade>(f.Document);
                       db.OnAddOrUpdate<FrontTrade>(ft, ft.Id);
                   }
               }
               catch (ApiErrorException e)
               {

               }
           }
       break;
       case "MIDDLE":
           if (_settings.ConnectionType == OtcConnectionType.BACKOFFICER)
           {
               try
               {

OTC.ClientAPI.Documents.Extensions.GetDocumentResponse<OTC.ClientAPI.Documents.MiddleTrades.MiddleTradeContract> m
                       = await Connection.MiddleTrades.GetAsync(data.ActiveId);
                   if (m.Success)
                   {
                       MiddleTrade mt = QuatroMapping.GetObject<MiddleTrade>(m.Document);
                       db.OnAddOrUpdate<MiddleTrade>(mt, mt.Id);
                   }
               }
               catch (ApiErrorException e)
               {

               }
           }
       break;
       case "BACK":
           if (_settings.ConnectionType == OtcConnectionType.BACKOFFICER)
           {
               try
```

```
                {

OTC.ClientAPI.Documents.Extensions.GetDocumentResponse<OTC.ClientAPI.Documents.BackTrades.BackTradeContract> b
                    = await Connection.BackTrades.GetAsync(data.ActiveId);
                if (b.Success)
                {
                    BackTrade bt = QuatroMapping.GetObject<BackTrade>(b.Document);
                    db.OnAddOrUpdate<BackTrade>(bt, bt.Id);
                }
            }
            catch (ApiErrorException e)
            { }
        }
      break;
      case "FINAL":
          if (_settings.ConnectionType == OtcConnectionType.BACKOFFICER)
          {
              try
              {

OTC.ClientAPI.Documents.Extensions.GetDocumentResponse<OTC.ClientAPI.Documents.FinalTrades.FinalTradeContract> f
                    = await Connection.FinalTrades.GetAsync(data.ActiveId);
                if (f.Success)
                {
                    FinalTrade ft = QuatroMapping.GetObject<FinalTrade>(f.Document);
                    db.OnAddOrUpdate<FinalTrade>(ft, ft.Id);
                }
                else
                {
                }
            }
            catch (ApiErrorException e)
            {

            }
        }
      break;
  }
 }
```

Пример 4. Формирование контента документов

```
public struct TransactionDefinition
  {
    public decimal? Price;
    public int? Qty;
    public string Direction;
    public DateTime? SettlDate;
    public DateTime? DeliveryDate;
    public string PriceCurrency;
    public string SettlCurrency;
    public string DeliveryMethod;
    public bool isFrontOffice;
  }


 public static string MakeSecurityQuoteContent( TransactionDefinition td, Instrument i)
  {
    string tranType = string.Empty;
    string instrumentType = string.Empty;

    switch (i.ClassId)
    {
        case "stocks":
            tranType = "equityTransaction";
            instrumentType = "equity";

            break;
        case "bonds":
            tranType = "bondTransaction";
            instrumentType = "bond";

            break;
        default:
            tranType = "equityTransaction";
            instrumentType = "equity";
            break;
    }

    XElement root = new XElement(rtsotc + tranType,
        new XAttribute(XNamespace.Xmlns + "fpml", fpml),
        new XAttribute(XNamespace.Xmlns + "rtsrep", rtsrep),
        new XAttribute(XNamespace.Xmlns + "fpmlext", fpmlext),
```

```
        new XAttribute(XNamespace.Xmlns + "dsig", ds),
        new XAttribute(XNamespace.Xmlns + "rtsotc", rtsotc)
       );

    XDocument doc = new XDocument(root);

    string buyerRef = "party1";
    string sellerRef = "party2";

    if (td.isFrontOffice)
    {
        buyerRef = (td.Direction.ToUpper() == "BUY") ? "quote-owner" : "counterparty";
        sellerRef = (td.Direction.ToUpper() == "BUY") ? "counterparty" : "quote-owner";
    }
    else
    {
        buyerRef = (td.Direction.ToUpper() == "BUY") ? "party1" : "party2";
        sellerRef = (td.Direction.ToUpper() == "BUY") ? "party2" : "party1";

    }

    root.Add(new XElement(fpml + "buyerPartyReference", new XAttribute("href", buyerRef)));
    root.Add(new XElement(fpml + "sellerPartyReference", new XAttribute("href", sellerRef)));
    root.Add(new XElement(rtsotc + "issuingVolumes", (i.IssuingVolume.HasValue) ?
 i.IssuingVolume.Value.ToString() : string.Empty));
    root.Add(new XElement(rtsotc + "numberOfUnits", (td.Qty.HasValue) ? td.Qty.Value.ToString() : string.Empty));
    root.Add(new XElement(rtsotc + "unitPrice", (td.Price.HasValue) ? GetStringNumber(td.Price.Value) :
 string.Empty));
    root.Add(new XElement(rtsotc + "priceCurrency", td.PriceCurrency));

    //блок идентификации инструмента
    XElement instrumentDefinition = new XElement(rtsotc + instrumentType);
    instrumentDefinition.Add(new XElement(fpml + "instrumentId", new XAttribute("instrumentIdScheme",
        "http://www.fpml.ru/coding-scheme/instrument-id#code"), i.Id));
    instrumentDefinition.Add(new XElement(fpml + "instrumentId", new XAttribute("instrumentIdScheme",
        "http://www.fpml.ru/coding-scheme/instrument-id#regnum"), i.Regnum));
    instrumentDefinition.Add(new XElement(fpml + "instrumentId", new XAttribute("instrumentIdScheme",
        "http://www.fpml.ru/coding-scheme/instrument-id#cfi"), i.CFI));
    instrumentDefinition.Add(new XElement(fpml + "instrumentId", new XAttribute("instrumentIdScheme",
        "http://www.fpml.ru/coding-scheme/instrument-id#isin"), i.ISIN));
    instrumentDefinition.Add(new XElement(fpml + "description", i.NameRu));
    instrumentDefinition.Add(new XElement(fpml + "currency", i.Currency));
    instrumentDefinition.Add(new XElement(fpml + "exchangeId", i.ExchangeId));
    root.Add(instrumentDefinition);

    root.Add(new XElement(rtsotc + "unitNotional", GetStringNumber(i.UnitNominal)));
    root.Add(new XElement(rtsotc + "deliveryMethod", td.DeliveryMethod));
    root.Add(new XElement(rtsotc + "settlementDate", (td.SettlDate.HasValue) ? td.SettlDate.Value.ToString("yyyy-
MM-dd") : string.Empty));
    root.Add(new XElement(rtsotc + "deliveryDate", (td.DeliveryDate.HasValue) ?
 td.DeliveryDate.Value.ToString("yyyy-MM-dd") : string.Empty));
    root.Add(new XElement(rtsotc + "settlementCurrency", td.SettlCurrency));

    return root.ToString(SaveOptions.None);
  }
```

Пример 5. Создание котировки

```
/// <summary>
  /// Создание собственной котировки
  /// </summary>
  /// <param name="InstrId">Идентификатор инструмента</param>
  /// <param name="Price">Цена</param>
  /// <param name="Qty">Количество</param>
  /// <param name="direction">Направление (b-BUY, s-SELL)</param>
  /// <returns>Структурированный результат выполнения действия</returns>
  public async Task<DocumentData> CreateQuote(string InstrId, decimal Price, int Qty, string direction)
  {
    OTC.ClientAPI.Accounts.GetResponse acc=await Connection.Accounts.GetCurrentAsync(_settings.Login);
    if (!acc.Success)
    {
        Log.OtcInteraction.Info("Невозможно получить данные об аккаунте для логина {0}", _settings.Login);
        return new DocumentData() { Error = string.Format("Невозможно получить данные об аккаунте для логина {0}",
 _settings.Login) };
    }


    OTC.ClientAPI.Documents.Quotes.QuoteContract q = new OTC.ClientAPI.Documents.Quotes.QuoteContract();
    q.Comment = "Autogenerated";
    q.DeliveryDate = DateTime.Now.AddDays(4d);
    q.DeliveryMethod = "DeliveryVersusPayment";
    q.Direction = (direction.ToUpper() == "B") ? OTC.ClientAPI.Documents.Quotes.OperationDirectionContract.Buy :
```

```
        OTC.ClientAPI.Documents.Quotes.OperationDirectionContract.Sell;
    q.Instrument=InstrId;
    q.IsDynamic=false;
    q.IsIndicative=false;
    q.IsTargeted=false;
    q.PartyId = acc.Account.PartyId;
    q.Price = Price;
    q.PriceCurrencyId = "USD";
    q.Quantity = Qty;
    q.SettlementCurrencyId = "USD";
    q.SettlementDate = DateTime.Now.AddDays(4d);
    q.TimeToLive = OTC.ClientAPI.Documents.Quotes.QuoteTimeToLiveContract.GoodTillCancelled;
    Instrument i = db.GetRecord<Instrument>(q.Instrument);
    q.Content=ContentBuilder.MakeSecurityQuoteContent(q,i);

    //создание черновика котировки
    OTC.ClientAPI.Documents.Extensions.CreateDocumentResponse<OTC.ClientAPI.Documents.Quotes.QuoteContract> x
        = await Connection.Quotes.CreateAsync(q);
    if (x.Success)
    {

        Log.OtcInteraction.Info("Котировка {0}/{1}/{2} отправлена успешно", InstrId, Price, Qty);
        //и активирование его в системе
        return await SendQuote(x.Document.Id, x.Document.Revision);

    }
    else
    {
        Log.OtcInteraction.Info("Ошибка при отправке котировки {0}/{1}/{2}", InstrId, Price, Qty);
        return new DocumentData() { Id = string.Empty, Rev = 0, Error = x.Errors.DocumentError.Message.Ru };
    }
  }
```

Пример 6. Создание договора бэк-офиса

```
/// <summary>
  /// Создание нового догооврa бэк-офиса
  /// </summary>
  /// <param name="price">Цена</param>
  /// <param name="qty">Количество</param>
  /// <param name="direction">Направление (b-BUY, s-SELL)</param>
  /// <param name="instrId">Тикер инструмента</param>
  /// <param name="party2Id">Идентификатор контрагента</param>
  /// <param name="clientId">Идентификатор клиента</param>
  /// <param name="settlSchemeId">Идентификатор расчетной схемы</param>
  /// <returns>Структурированный результат выполнения действия</returns>
  public async Task<DocumentData> CreateBackTrade(decimal price, int qty, string direction, string instrId,
    string party2Id, int? clientId, int settlSchemeId)
  {

    OTC.ClientAPI.Accounts.GetResponse acc = await Connection.Accounts.GetCurrentAsync(_settings.Login);
    if (!acc.Success)
    {
        Log.OtcInteraction.Info("Невозможно получить данные об аккаунте для логина {0}", _settings.Login);
        return new DocumentData() { Error = string.Format("Невозможно получить данные об аккаунте для логина {0}",
 _settings.Login) };
    }

    OTC.ClientAPI.Documents.BackTrades.BackTradeContract contract = new
 OTC.ClientAPI.Documents.BackTrades.BackTradeContract();
    contract.AdditionalTerms = "NONE";
    contract.Comment = "Autogenerated";
    TransactionDefinition def = new TransactionDefinition()
    {
        Price = price,
        Qty = qty,
        Direction = (direction.ToUpper()=="B")? "Buy":"Sell",
        isFrontOffice = false,
        PriceCurrency = "USD",
        SettlCurrency = "USD",
        SettlDate = DateTime.Now.AddDays(4),
        DeliveryDate = DateTime.Now.AddDays(4),
        DeliveryMethod = "DeliveryVersusPayment"
    };
    Instrument i = db.GetRecord<Instrument>(instrId);
    contract.Content = ContentBuilder.MakeSecurityQuoteContent(def, i);
    contract.CheckLimits = false;
    contract.DeliveryDate = def.DeliveryDate;
    contract.DeliveryMethod = def.DeliveryMethod;
    contract.ExchangeRate = null;
    contract.ExternalTradeIdentifier1 = Guid.NewGuid().ToString();
    contract.InstrumentId = i.Id;
```

```
   contract.IsPublic = false;

   if (clientId.HasValue)
   {
       Client c = db.GetRecord<Client>(clientId);
       contract.Party1ClientCode = c.Name;
       contract.Party1ClientRelation = "AgentAgreement";
       contract.Party1ClientRelationBasis = "Некоторое основание";
   }
   contract.Party1Id = acc.Account.PartyId;
   contract.Party2Id = party2Id;
   contract.Price = def.Price;
   contract.PriceCurrencyId = def.PriceCurrency;
   contract.Quantity = def.Qty;
   contract.SecuritiesTransferParty = "SettlementDepository";
   contract.SettlementCurrencyId = def.SettlCurrency;
   contract.SettlementDate = def.SettlDate;

   OTC.ClientAPI.SettlementSchemes.Schemes.GetResponse settlScheme = await
Connection.SettlementSchemes.GetAsync(settlSchemeId, 1);
   if (settlScheme.Success)
   {
       contract.Party1SettlementSchemeXml = settlScheme.SettlementScheme.Body;
   }


OTC.ClientAPI.Documents.Extensions.CreateDocumentResponse<OTC.ClientAPI.Documents.BackTrades.BackTradeContract>
result
       = await Connection.BackTrades.CreateAsync(contract);
   if (result.Success)
   {
       return new DocumentData()
       {
           Id = result.Document.Id,
           Rev = result.Document.Revision,
           State = result.Document.State.Id
       };
   }
   return new DocumentData();
 }
```