"BossaAPI .NET Class Library"

Biblioteka **`pjank.BossaAPI.dll`** - napisana w C# (.NET 3.5) - powstała, by ułatwić korzystanie z możliwości API udostępnionego niedawno przez DM BOSSA. Udostępnia ona **czytelne, proste w użyciu klasy** reprezentujące m.in.: poszczególne rachunki użytkownika, papiery tam zgromadzone, ich notowania oraz aktywne zlecenia. Wszystko w takiej formie, by nawet laik, który dopiero zaczyna programować, był w stanie z nich efektywnie korzystać. I to korzystać zawsze tak samo - **niezależnie od wybranego sposobu komunikacji** z systemem. Na dzień dzisiejszy komunikujemy się z aplikacją NOL3 uruchomioną na tym samym komputerze, w przyszłości pewnie będzie możliwy dostęp bezpośrednio do serwerów Domu Maklerskiego... A niezależnie od tego – już niedługo planuję wprowadzić funkcjonalność umożliwiającą każdemu uruchomienie prywatnego „serwera", który pozwoli w bezpieczny sposób współdzielić aktywne połączenie z rachunkiem dla kilku aplikacji jednocześnie, dla innego komputera, a nawet dla komórki, czy innego urządzenia mobilnego (przez Internet).

Do wykorzystania biblioteki można użyć dowolnego języka programowania dostępnego na platformę .NET. Większość funkcji wywołamy za pośrednictwem statycznych metod i właściwości głównej klasy `'Bossa'`. W planie jest również stworzenie komponentu COM, który udostępni tą samą funkcjonalność (i w równie czytelnej formie) np. w Excelu (VBA).

Bibliotekę udostępniam na licencji *Apache v2.0* - wraz z pełnym, **otwartym kodem** źródłowym. W ten sposób każdy może ją wykorzystać w swoich projektach - nieważne, czy to projekt komercyjny, czy darmowy, otwarty czy też nie... Obowiązuje jedynie zasada, by nie zabrakło tam nigdy wzmianki o pierwotnym autorze (czyli o mnie :)) i wszystkich kolejnych (jeśli tacy się pojawią i będą udostępniać swoje modyfikacje, do czego oczywiście zachęcam).

Przemysław Jankowski

http://www.pjank.net/bossa.api/ e-mail: **bossa.api@pjank.net**

Zawartość katalogów

Biblioteka rozwijana jest w darmowym środowisku ***Visual Studio 2010 Express*** Microsoftu. Wystarczy załadować plik *„BossaAPI.sln”* (ew. *„BossaAPI.vs2008.sln” dla Visual Studio 2008*). W ramach tego *„solution”* zawarto trzy projekty, w następujących podkatalogach:

1. **`'BossaAPI'` - kod samej biblioteki**, podzielony na kilka warstw:
   a) warstwa "zewnętrzna", udostępniająca w prostej formie większość funkcji API.

      Obejmuje najważniejszą klasę **`'Bossa'`** i dostępne z niej potem kolejne: `'BosAccount'`, `'BosPaper'`, `'BosOrder'`, `'BosInstrument'`, `'BosTrade'` itd. (wszystko w podkatalogach *"src\AccountData\"* i *"src\MarketData\"*).

   b) niskopoziomowa obsługa protokołu FIXML

      Komplet klas znajdujący się w katalogu *"src\Networking\Fixml\"*. Dla każdego rodzaju komunikatu (tych wysyłanych, jak i odbieranych) mamy indywidualną klasę C# oferującą wszystkie możliwe parametry w formie łatwo dostępnych pól obiektu (reprezentowanych przez kolejne klasy albo typy wyliczeniowe odpowiednie do rodzaju danego parametru).

      Jeśli z jakiegoś powodu chcesz samodzielnie oprogramować komunikację np. bezpośrednio z aplikacją NOL3, to będzie najlepsze rozwiązanie - pozwala skupić się na treści komunikatów zamiast obsłudze samego XML'a i zmniejsza ryzyko popełniania błędów. Możesz też użyć dodatkowej klasy **`'NolClient'`** (katalog *"src\Networking\"*), która wspomaga całą komunikację z NOLem - zajmuje się ustalaniem numeru portu do nawiązania połączenia, zalogowaniem użytkownika, obsługą kanału asynchronicznego w oddzielnym wątku itp. itd.

   c) warstwa pośrednia - łącząca dwie powyższe i zaprojektowana w taki sposób, by umożliwić bardzo szybką podmianę protokołu FIXML czymś zupełnie innym.

      Obejmuje interfejs **`'IBosClient'`** (którego jedyną na razie implementację stanowi wspomniana klasa `'NolClient'`) oraz klasy transportowe (DTO) służące do przekazywania danych między kolejnymi warstwami biblioteki.

2. **`'TestApp1'` - przykładowa aplikacja konsolowa**

   Demonstruje sposób bezpośredniego wykorzystania m.in. klasy `'NolClient'`, a więc tej bardziej "niskopoziomowej" warstwy niniejszej biblioteki.

3. **`'TestApp2'` - przykładowa aplikacja GUI**

   Tutaj wykorzystano już "zewnętrzną" warstwę biblioteki - klasę `'Bossa'`. Aplikacja demonstruje większość dostępnych funkcji, włącznie z możliwością oglądania tabeli aktualnych notowań oraz składania zleceń. Jednocześnie w okienku logu umożliwia podgląd wszystkiego, co się dzieje "pod maską".

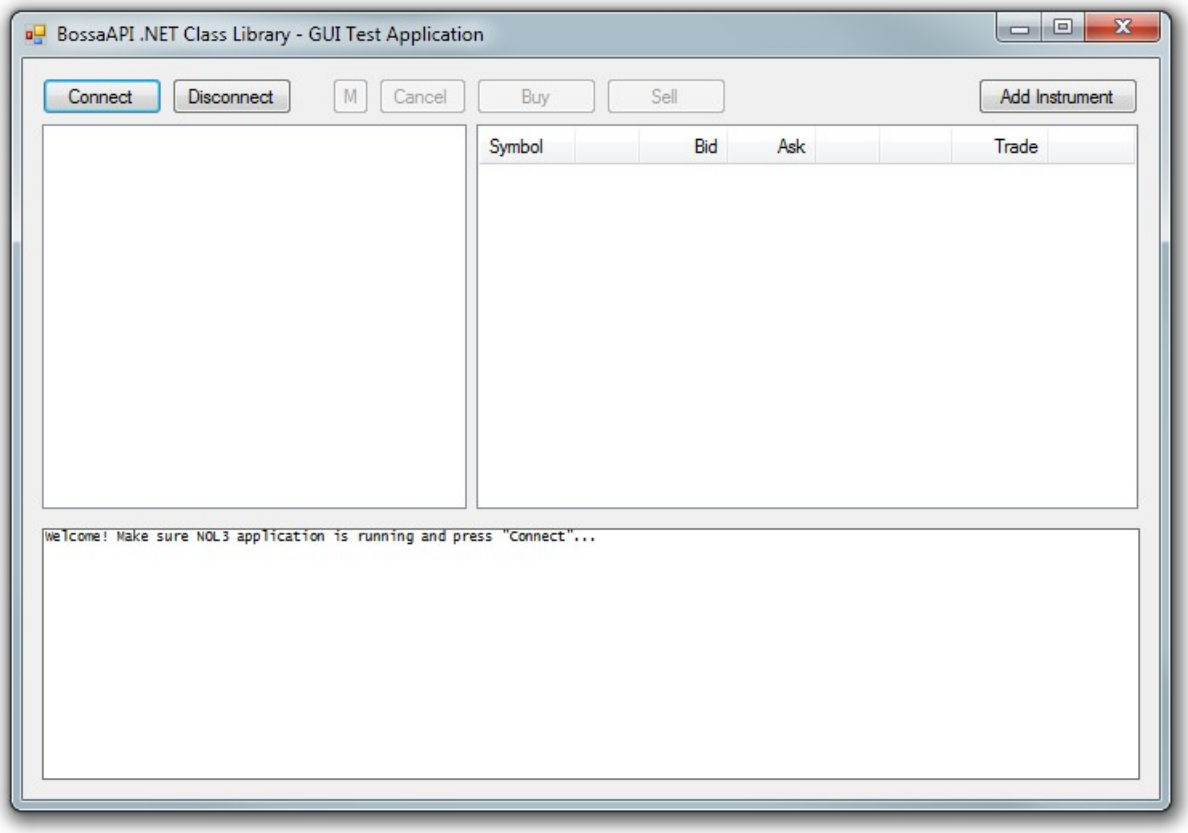
Jak korzystać...

Przykładowa aplikacja

Na początek proponuję uruchomić przykładowy program **`'TestApp2.exe'`**, dzięki któremu w praktyce poznamy podstawowe możliwości biblioteki.

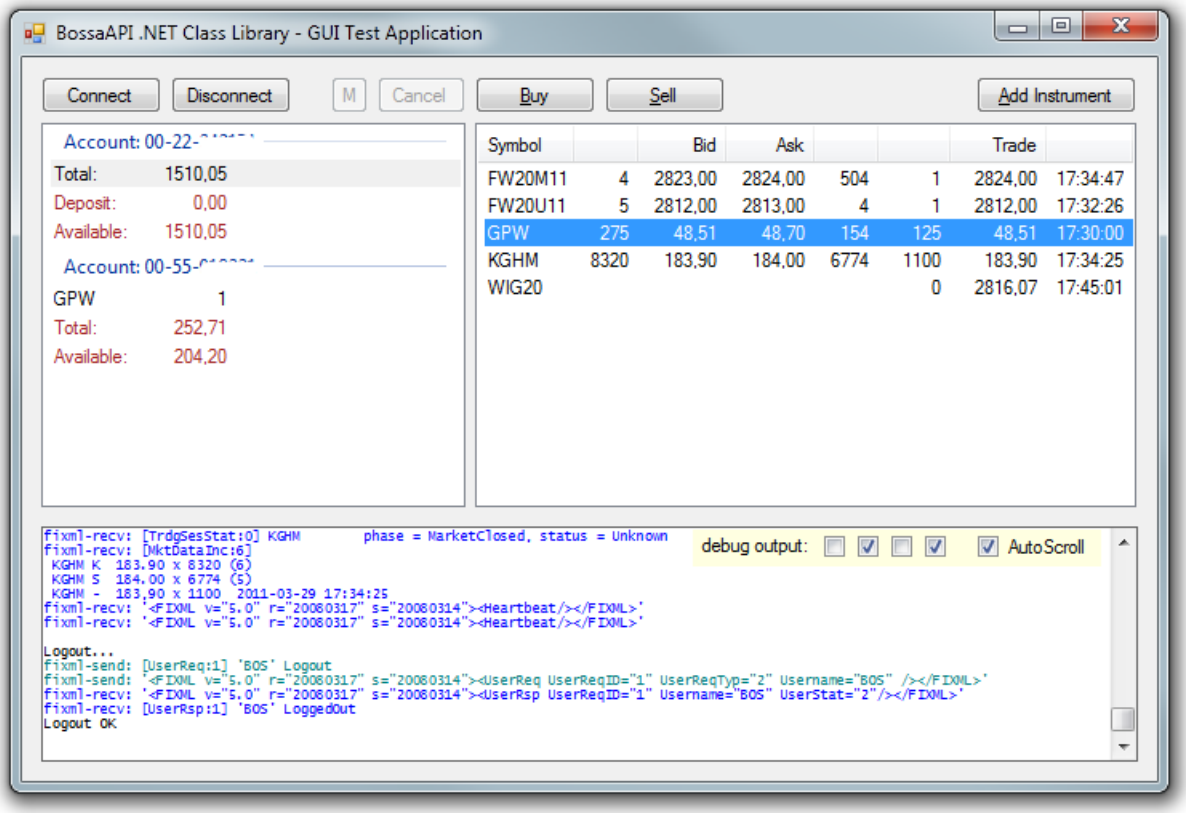
Uwaga: program uruchamiasz na swoim rachunku na własną odpowiedzialność! Nie odpowiadam za ewentualne nieprawidłowości w składaniu zleceń itp. - tym bardziej, że to wciąż wersja rozwojowa biblioteki... nie mówiąc już o samym interfejsie bossaAPI w aplikacji NOL3, który też trudno nazwać wersją „finalną".

Zakładając, że w systemie mamy zainstalowany .NET (w wersji 3.5) i w bieżącym katalogu będzie dostępna biblioteka `'pjank.BossaAPI.dll'` - na ekranie ukaże się następujące okienko:



Widzimy tu następujące obszary:

- na górze - przyciski do wywoływania poszczególnych funkcji,
- po lewej - okienko, gdzie będą wyświetlane informacje o naszych rachunkach: lista obecnych na rachunku papierów wartościowych (otwartych pozycji), lista aktywnych zleceń oraz krótkie zestawienie najważniejszych kwot dotyczących danego rachunku.
- po prawej - tabelka z aktualnymi notowaniami wybranych instrumentów: symbol, najlepsza oferta kupna (liczba walorów, cena), najlepsza oferta sprzedaży (cena, liczba) oraz ostatnio zrealizowana transakcja (liczba walorów, cena, godzina transakcji).
- na dole - okienko logu, gdzie podczas pracy programu będą wyświetlane mniej lub bardziej szczegółowe informacje nt. wewnętrznych działań biblioteki (np. treść przesyłanych komunikatów – oryginalna albo przetworzona do bardziej czytelnej postaci) – poziom tej szczegółowości ustalamy przełącznikami, które się pojawią po najechaniu tam myszką.



Kolejne przyciski, jakie mamy do dyspozycji:

- *Connect* – nawiązanie połączenia z aplikacją NOL3 uruchomioną na tym samym komputerze (zalogowanie użytkownika, otwarcie kanału asynchronicznego, pobranie informacji o rachunkach i automatyczne włączenie subskrypcji notowań dla każdego instrumentu, jaki się znajdzie na którymś z tych rachunków).
- *Disconnect* – przerwanie połączenia z NOLem (zamknięcie kanału asynchronicznego, wylogowanie użytkownika). W każdej chwili możemy wznowić połączenie (i aktualizacje stanu rachunków, notowań) klikając ponownie przycisk *Connect*.
- *M* – modyfikacja zlecenia (jedno z aktywnych zleceń wskazane przez nas w lewym okienku - *na załączonym obrazku aktywnych zleceń akurat nie ma, ale gdyby były... tam właśnie należy najpierw kliknąć :)*) Po wybraniu tej funkcji, program zapyta o nową cenę (można wpisać kwotę lub np. PKC) i po jej zaakceptowaniu – wyśle do systemu żądanie zmiany tego zlecenia. Biblioteka (jak i sama Bossa - z tego, co mi wiadomo, więcej opcji modyfikacji zleceń nie ma) udostępnia również możliwość zmiany daty ważności... tutaj to pominięto, żeby nie komplikować aplikacji.
- *Cancel* – anulowanie wskazanego zlecenia jw. *(uwaga: bez żadnych pytań! od razu wysyła anulatę...)*
- *Buy* – złożenie zlecenia kupna – dla instrumentu wskazanego w okienku po prawej stronie. Po wywołaniu przycisku, program najpierw zapyta o cenę (i jak przy *„M”* – możemy podać kwotę albo słówko: PKC, PCR, PCRO), przy czym domyślnie podstawi cenę z najlepszej aktualnie oferty sprzedaży... a następnie pyta o liczbę walorów, jaką chcemy kupić. Dopiero zatwierdzenie tej drugiej wartości spowoduje wysłanie zlecenia do systemu. Inne parametry zleceń (limit aktywacji, WiA, WuA, WUJ itp. itd.) znowu pominięto, by nie burzyć prostoty tej przykładowej aplikacji, ale biblioteka sama w sobie obsługuje je wszystkie.
- *Sell* – złożenie zlecenia sprzedaży – analogicznie jw.
- *Add Instrument* – pozwala dodać do tabelki kolejny instrument – włączając go w zakres papierów, dla których aktualizujemy notowania i pozwalając na składanie na nim zleceń.

Własna aplikacja

Gdy już na opisanym wyżej przykładzie przekonasz się, że „to działa”... czas spróbować wykorzystać bibliotekę do własnych zastosowań.

Potrzebna będzie oczywiście dll-ka *„pjank.BossaAPI.dll”*, którą dołączamy (w „References”) do swojego projektu. Warto też w tym samym katalogu (obok ww. dll-ki), umieścić również plik *„pjank.BossaAPI.xml”*, dzięki czemu uzyskamy w *Visual Studio* podpowiedzi (komentarze) do poszczególnych metod i właściwości większości klas biblioteki.

Kolejny krok to zaimportowanie „namespace” biblioteki do każdego pliku, w którym będziemy się odwoływać do oferowanych przez nią klas (krok opcjonalny, ale poprawia czytelność kodu). W przypadku języka **C#** dodajemy na początku pliku instrukcję:

```
using pjank.BossaAPI;
```

I to tyle! Już możemy korzystać - m.in. ze statycznej klasy `'Bossa'`, będącej niejako „punktem wyjścia” do reszty biblioteki (a przynajmniej tej prostszej w użyciu, „zewnętrznej” warstwy).

Poniżej kilka przykładów (**C#** 3.0)... a po szczegóły zapraszam do komentarzy w źródłach.

Najpierw należy się zalogować do rachunku na www, uruchomić NOL'a i nawiązać z nim połączenie:

```
// nawiązanie połączenia z aplikacją NOL3
Bossa.ConnectNOL3();
```

Po chwili powinniśmy mieć już dostęp do danych na listach Bossa.Accounts[] oraz Bossa.Instruments[]...

```
// odczyt stanu rachunku
var stan_konta = Bossa.Accounts["nr-rachunku"].PortfolioValue;
var wolne_srodki = Bossa.Accounts["nr-rachunku"].AvailableFunds;
```

Każde Accounts[] (klasa BosAccount) poza kwotami jw. udostępnia też listę papierów (BosPaper):

```
// sprawdzenie liczby otwartych pozycji danego kontraktu
// (przy okazji: jak w miejsce numeru rachunku wystarczy podać jego fragment)
var ile = Bossa.Accounts["00-22-"].Papers["FW20M11"].Quantity;
```

... oraz listę wszystkich bieżących zleceń (BosOrder):

```
// wylistowanie aktywnych zleceń z danego rachunku
foreach (var order in Bossa.Accounts["00-55-"].Orders)
  if (order.IsActive)
    Console.WriteLine("{0}: {1} {2} x {3} - {4}", order.Instrument,
      order.Side, order.Quantity, order.Price, order.StatusReport);
```

Lista instrumentów (BosInstrument) jest automatycznie uzupełniana przy pierwszym użyciu danego symbolu:

```
// szybki odczyt kursu danego instrumentu
var kurs = Bossa.Instruments["KGHM"].Trades.LastPrice;
```

Jeśli nie zdążył odebrać notowań, zwróci wyżej null. Możemy poczekać... albo podłączyć się pod Bossa.OnUpdate.

```
// tym razem po kodzie ISIN zamiast Symbolu - możemy ich używać zamiennie
var kghm = Bossa.Instruments.FindByISIN("PLKGHM000017");
// odczyt całej znanej historii notowań
foreach (var trade in kghm.Trades)
  Console.WriteLine("{0}  {1,7} x {2,-5}",
    trade.Time.TimeOfDay, trade.Price, trade.Quantity);
// odczyt najlepszych pozycji w tabeli ofert
var bid = kghm.BuyOffers.BestPrice;
var ask = kghm.SellOffers.BestPrice;
```

Zlecenia składamy najlepiej też spod obiektu klasy BosInstrument:

```
// złożenie zlecenia kupna 10 sztuk po 175.50 zł
Bossa.Instruments["KGHM"].Buy(175.50, 10);
// zlecenie sprzedaży 10 sztuk StopLoss z aktywacją po 170.00 zł
Bossa.Instruments["KGHM"].Sell(BosPrice.PKC, 170, 10);
```