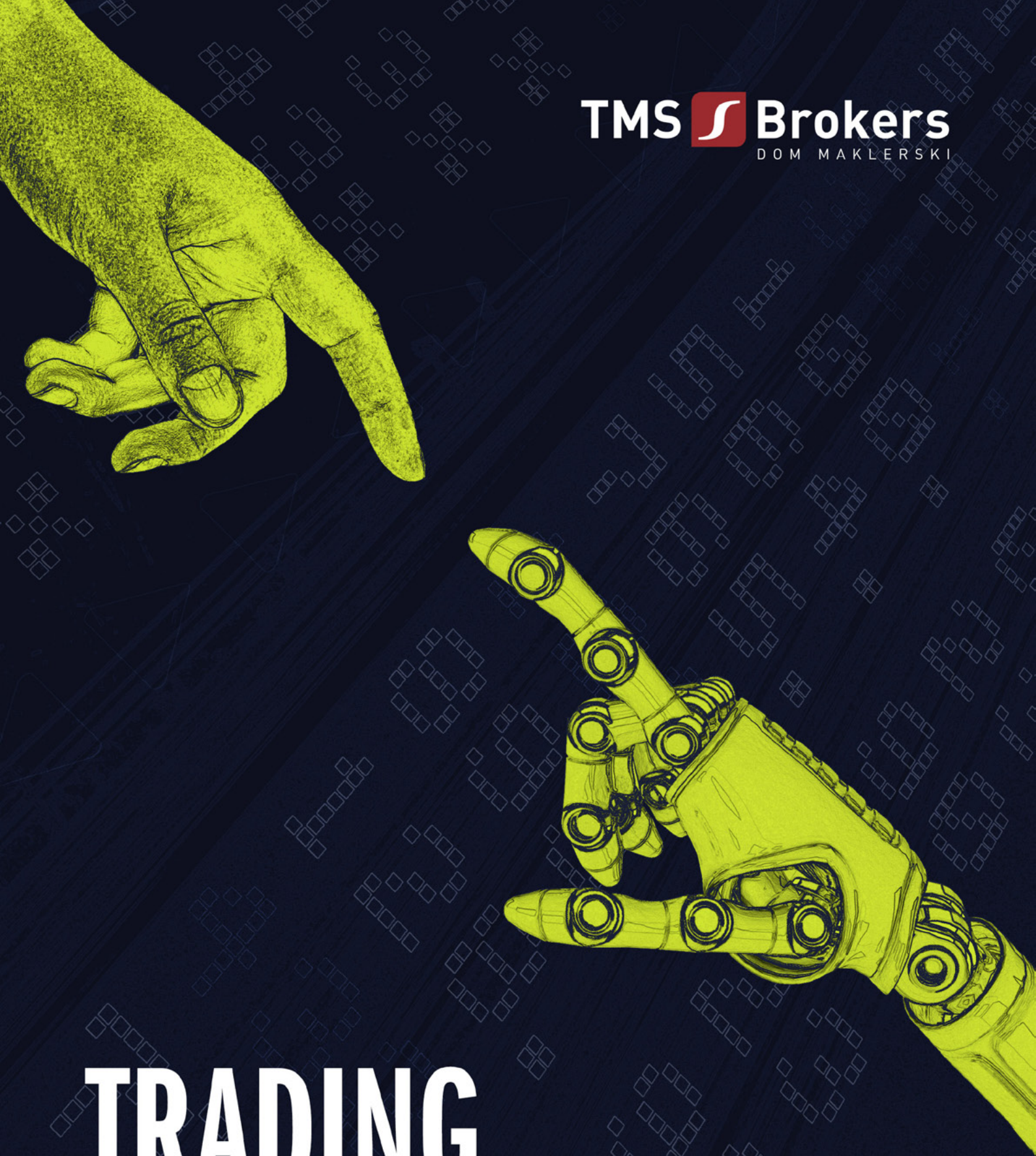
TMS Brokers
DOM MAKLERSKI



TRADING AUTOMATYCZNY

[...] WSPÓŁCZESNE TECHNOLOGIE WKRACZAJĄ W CORAZ TO NOWSZE OBSZARY NASZEGO CODZIENNEGO ŻYCIA [...]

WPROWADZENIE

CZĘŚĆ PODSTAWOWA

CZĘŚĆ ZAAWANSOWANA

ZAKOŃCZENIE

Tomasz Waszczyk

Absolwent Wydziału Automatyki, Elektroniki i Informatyki Politechniki Śląskiej w Gliwicach na kierunku Informatyka (studia prowadzone w języku angielskim).

Zawodowo zajmuję się wytwarzaniem oprogramowania oraz algorytmów samo decyzyjnych jak również wdrażaniem i utrzymywaniem zaimplementowanych rozwiązań w celu osiągnięcia jak najlepszych wyników. Do moich głównych zainteresowań należy rynek kontraktów CFD, amerykański rynek akcji oraz opcji. Codzienne obowiązki dzielę na programowanie (między innymi mobilnych aplikacji hybrydowych) oraz handel dla banku z grupy Tier 1. W czasie wolnym czytam książki o geopolityce, zarządzaniu projektami, psychologii. Jednocześnie stale zgłębiam zagadnienia współczesnej historii oraz sztuki. Uczestniczę w wielu konferencjach zarówno o tematyce oprogramowania jak i finansowej. Uważam, że nie można być długoterminowo zyskownym uczestnikiem rynku finansowego bez regularnej aktywności fizycznej - biegam, pływam oraz szukam nowych pomysłów inwestycyjnych w saunie. Dzięki systematycznemu wyszukiwaniu analogii pomiędzy światem wykresów a codzienną rzeczywistością udało mi się odkryć i wdrożyć z sukcesem parę zależności inwestycyjnych. Staram się kreować synergię pomiędzy światem wykresów a systemami informatycznymi które pozwalają na analizę dużo większej ilości danych niż mózg człowieka jest w stanie przetworzyć. Porozumiewam się w języku Mickiewicza, Szekspira oraz Schumanna.

Wprowadzenie

Każdego dnia możemy obserwować, jak wwspółczesne technologie wkraczają w coraz to nowsze obszary naszego codziennego życia. Jednym z takich obszarów są finanse. Dawniej, aby wysłać zlecenie, należało udać się w specjalne miejsce, założyć rachunek, przelać pieniądze, wypełnić specjalny blankiecik i z nim udać się do kasy naszego domu maklerskiego. Obecnie tą samą operację możemy zrobić siedząc przy własnym biurku, mając do dyspozycji jedynie komputer z dostępem do sieci Internet. Jeszcze przed paroma laty, aby móc zacząć myśleć o handlowaniu walutami na rynku międzybankowym, należało posiadać na swoim rachunku co najmniej kilkadziesiąt tysięcy waluty bazowej. Współczesne realia są zupełnie inne.

Głównym celem, który przyświecał mi przy pisaniu tej książki, było sprawienie, aby mój Czytelnik kończąc każdy podrozdział pomyślał sobie „Jakie to proste!". Jeszcze przed rozpoczęciem pisania musiałem się zastanowić, jakiej wersji Meta Trader będzie dedykowana ta publikacja, jako że obecnie na rynku mamy dwie najpopularniejsze platformy transakcyjne Meta Trader 4 oraz nowszą jego wersję, oznaczoną jako Meta Trader 5. Ostatecznie wybrałem wersję Meta Trader 4, uznając, że lepiej jest pisać książkę dla obowiązującego standardu. Nie odbiegniemy daleko od prawdy, stwierdzając, że nowsza wersja MT5 jest większą aktualizacją MT4. Z kolei język programowania MQL został rozwinięty o tzw. paradygmat programowania, czyli programowanie obiektowe, dzięki któremu możemy pisać kod źródłowy operując na większej abstrakcji, a jednocześnie za pomocą bardziej namacalnych obiektów.

Aby ułatwić wszystkim Czytelnikom możliwość eksperymentowania, pod adresem https://github.com/TomaszWaszczyk/Trading-Automatyczny-MQL znajdują się wszystkie kody źródłowe, dzięki czemu można uniknąć bezsensownego ich przepisywania do naszego Meta Editora.

W razie pytań zapraszam do kontaktu za pomocą platformy GitHub.

Przyjęte oznaczenia w książce

Aby zwiększyć czytelność treści, dla niektórych elementów, takich jak kod źródłowy, linki czy odnośniki do innych publikacji, przyjąłem następującą konwencję zapisu:

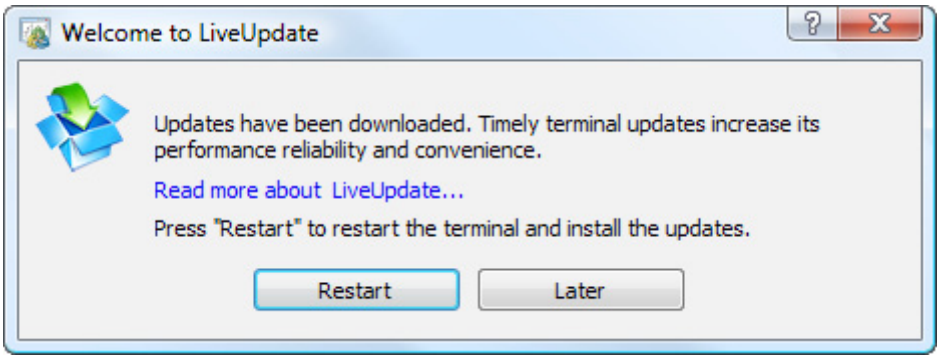
Przykładowy Kod Źródłowy:

```
int a = 50;
int b = 100;
MessageBox („Bardzo proste dodawanie. a+b=” + (a + b), „a+b=?”);
```

Ciekawostka: aby otworzyć Meta Editor 4, wystarczy, że naciśniesz F4 pracując w Meta Trader 4.

W związku z tym, że książka, którą, Drogi Czytelniku, trzymasz właśnie w ręku, pomyślana jest jako wstęp do programowania w MQL, chciałbym wyjaśnić, iż terminy program i system będę stosować w niej naprzemiennie. Gdy natomiast w tekście pojawi się termin strategia, przyjmuje on dwa możliwe znaczenia: po pierwsze, strategia jako ciąg wydarzeń występujących po sobie, spełniający nasze kryteria transakcyjne, czego efektem jest konkretna transakcja, po drugie strategia jako implementacja, czyli przełożenie naszych pomysłów na kod źródłowy, w celu napisania automatycznego systemu transakcyjnego. Odnosząc się do nazw Meta Trader 4 lub Meta Trader 5 stosować będę z kolei skróty MT4/MT5. Używam też tych powszechnie znanych, np. MQL – MetaQuotes Language.

Zapewne każdy z użytkowników, który na co dzień korzysta z oprogramowania Meta Trader w dowolnej jego wersji, przynajmniej raz podczas uruchomienia programu został uraczony następującym komunikatem (lub podobnym):



Jest to informacja o aktualizacji wersji MT, jaką mamy zainstalowaną u siebie na komputerze. Aktualizacje te pojawiają się stosunkowo często, co ma swoje zarówno dobre, jak i złe strony: z jednej strony powinniśmy być zadowoleni z tego, że oprogramowanie, za pomocą którego dokonujemy na co dzień decyzji inwestycyjnych, dotyczących konkretnych, w pocie czoła zarobionych przez nas pieniędzy, jest wolne od błędów. Nie chcielibyśmy przecież stracić zysków lub nawet pewnej ich części z powodu błędu w oprogramowaniu! Za pomocą aktualizacji programiści wraz z testerami wyszukują błędy, po czym je usuwają, aby każdy

z traderów mógł z zadowoleniem korzystać ze swojej platformy. Z drugiej jednak strony, czasami po aktualizacji przestają nam działać programy, jak i skrypty napisane przez nas lub innych programistów MQL. Dzieje się tak, ponieważ jedną z mniej przyjemnych konsekwencji częstych aktualizacji jest zmiana języka MQL. W efekcie, jeśli w naszym skrypcie wykorzystaliśmy funkcję, którą architekci MQL oznaczyli jako „deprecated", czyli przestarzała (dalej już niewspierana), najczęściej oznacza to konieczność przepisania naszego kodu w taki sposób, aby był kompatybilny z najnowszą wersją naszej platformy transakcyjnej.

W związku z tymi zmianami uznałem, że potrzebujemy pewnego punktu odniesienia, czyli punktu startowego do wszelkich rozważań oraz ćwiczeń. W związku z tym będę się opierać na aktualnie najnowszej wersji MT4 (Build 745). Zainstalowaną wersję programu możemy sprawdzić klikając w głównym oknie MT w Pomoc, a następnie w zakładkę „O Programie".

Dla kogo jest ta książka?

Zaczynając pisać tę książkę, już na samym początku musiałem odpowiedzieć sobie na bardzo ważne pytanie, kim będzie potencjalny Czytelnik. Uznałem, że Czytelnik ten jest zaznajomiony z takimi pojęciami jak cena Ask, cena Bid, spread, a także posiada podstawowe informacje na temat szeroko pojętego rynku Forex. Myślę, że krąg odbiorców można jednak poszerzyć o osoby początkujące, które znajdą w tej książce materiał do dalszego rozwoju.

Postawiłem sobie za cel, aby Czytelnik po skończeniu lektury był w stanie bez problemu napisać prosty automat i sprawnie poruszał się na platformie Meta Editor. Ma on też poznać pojęcia z dziedziny programowania MQL oraz otrzymać liczne wskazówki, zgodnie z którymi powinien dalej podążać tą drogą. Niestety książka ta ma ograniczoną objętość i nie jestem w stanie poruszyć wszystkich problemów, którym programista automatów wcześniej czy później będzie musiał stawić czoła. Wszelkie nowe pojęcia będę starał się omawiać na bieżąco, posiłkując się obrazkami i grafami, tak aby Czytelnik nie czuł się zagubiony już na samym początku swojej programistycznej drogi. Z doświadczenia wiem, że bardzo często pojawia się wiele pytań odnośnie programowania automatów. Chciałbym te szczególnie popularne kwestie wyjaśnić na łamach tej książki.

Myślę, że każdy zainteresowany znajdzie coś dla siebie, a być może nawet zarażę trochę osób tym fascynującym tematem. Już na wstępie chciałbym jednak jasno zasygnalizować, że budowa zyskownego automatu zabiera wiele czasu oraz energii i – mimo że często na początku udaje nam się zaprogramować automat zarabiający w testach setki procent miesięcznie – po bardziej wnikliwej analizie, nierzadko zajmującej dziesiątki godzin, znajdujemy tzw. problemy logiczne. Problemy te wydają się możliwe z technicznego punktu widzenia, jednak są niemożliwe w realnym handlu, na co warto uczulić każdego Czytelnika. Po przeczytaniu tej książki być może zaoszczędzicie ten czas, który ja straciłem, i który mogłem wykorzystać lepiej.

Zarys historii z zakresu automatycznych strategii transakcyjnych

Do napisania tego podrozdziału skłoniły mnie głównie dwie rzeczy, a jedną z nich jest z pewnością pasja do historii. Zawsze uważałem i uważam nadal, że w handlowaniu znajomość historii bardzo się przydaje, podobnie zresztą jak życiu codziennym. Drugim powodem jest często zadawane mi pytanie, czy zajmuję się programowaniem tzw. HFT Systems (High Frequency Trading), którymi są zazwyczaj komputery lub zbiór komputerów o ogromnej mocy obliczeniowej, ulokowane najczęściej w okolicach największych centrów finansowych, takich jak Londyn czy Hong Kong. Jako ciekawostkę podam przykład jednej z firm zajmującej się wytwarzaniem tego typu systemów, która na własny koszt zdecydowała się przeprowadzić łącza światłowodowe pomiędzy Londynem a Nowym Jorkiem. Inwestycja ta zwróciła się po upływie niewiele ponad roku. Odpowiadając na powyższe pytanie, zaznaczę, że nie będę opisywać tego typu programów z kilku powodów: średni budżet tego typu projektów zaczyna się od paru milionów euro i wymaga co najmniej 20-osobowego zespołu złożonego z wysoko wykwalifikowanych programistów, fizyków, statystyków oraz traderów, a i cały ten zespół niewiele zdziała, jeśli nie będzie miał dostępu do bardzo szybkich łączy pomiędzy ich komputerami (tzw. mainframe) oraz Tier 1[1] . Niech zatem sam Czytelnik odpowie na pytanie: co ja mogę zrobić w pojedynkę wobec tak ogromnych funduszy inwestycyjnych, bez milionowego budżetu, zespołu ludzi czy potężnych komputerów?

Wbrew pozorom obracanie takim ogromnym kapitałem bywa bardzo problematyczne i nierzadko dużo łatwiej jest handlować mniejszymi kwotami. I tutaj widzę właśnie punkt przewagi, jaką mają traderzy „detaliczni" nad tymi „instytucjonalnymi". Tym drugim jest bardzo trudno otworzyć właściwe pozycje, a kiedy już je otworzą, równie trudno je zamknąć, podczas gdy w przypadku automatów obracających stosunkowo małym kapitałem kapitałem problem płynności nie istnieje. Myślę, że jest to godne przemyślenia zagadnienie i skłania do działania, do czego gorąco zachęcam Cię, Drogi Czytelniku.

1 https://www.wikijob.co.uk/wiki/investment-banking

Część podstawowa

W tym rozdziale planuję zapoznać każdego z Czytelników z podstawowym nazewnictwem dotyczącym tematyki rynku finansowego Forex. Dzięki temu będzie nam się łatwiej rozmawiało przez następne strony niniejszej publikacji. Jeśli jesteś doświadczonym programistą, możesz jedynie przekartkować w celu upewnienia się, czy na pewno posługujemy się tym samym językiem.

Jednym z podstawowych pojęć, które chciałbym wyjaśnić, jest trading algorytmiczny. Istota tego zjawiska jest stosunkowa łatwa do uchwycenia. Wyobraźmy sobie, że po wielogodzinnych obserwacjach rynku walutowego zauważyliśmy cyklicznie występującą zależność. Dla lepszego zobrazowania, niech będzie to przecięcie się szybkiej średniej arytmetycznej z wolną średnią arytmetyczną od dołu, co oznacza dla nas sygnał kupna. Sygnałem sprzedaży będzie sytuacja odwrotna, to znaczy przecięcie średniej wolnej przez średnią szybką od góry. Mając tak zdefiniowane warunki handlu możemy sobie zaprogramować, parametryzując dokładnie nasze założenia, taki automat, który będzie wykonywać transakcje ściśle zgodnie z określonymi przez nas regułami. Ogólny schemat procesu decyzyjnego jest przedstawiony poniżej:

Widzimy, że na podstawie nowych danych, które trafiają do naszego algorytmu, mogą być już wykonywane konkretne zlecenia. Oczywiście podana przeze mnie przykładowa strategia jest bardzo prosta i nie możemy liczyć, że taki automat będzie zarabiać w dłuższej perspektywie.

Automat pozwala nam również na szybkie testowanie strategii. Po uprzednim wcześniejszym zdefiniowaniu konkretnych parametrów możemy obserwować, jak wynik strategii zmienia się na skutek dokonania zmiany jedynie w wartościach parametrów, wszystkich lub poszczególnych.

Wprowadzenie do platformy Meta Trader4

Chciałbym, aby na początku naszej wspólnej podróży każdy z Czytelników miał pojęcie o tym, jak działa platforma tradingowa MetaTrader 4 firmy MetaQuotes Software Corp., z czysto technologicznego punktu widzenia.

Instalując MetaTrader 4 otrzymujemy trzy ściśle współpracujące ze sobą moduły:

Meta Trader	Meta Editor	Tester Strategii

Warto uściślić, jaka jest relacja pomiędzy programami MetaTrader 4 a MetaEditor 4. Meta Trader 4 (w skrócie MT lub MT4) to środowisko pozwalające na uruchomienie naszych strategii automatycznych, a zarazem miejsce, w którym widzimy efekt zaprogramowanego przez nas robota. Warto dodać, że w nazewnictwie MT funkcjonuje jako Expert Advisor (w skrócie EA).

Meta Editor to tzw. Integrated Development Environment (IDE), czyli środowisko programistyczne dla programistów MQL'a, a zarazem program, w którym będziemy pisać kod źródłowy oraz kompilować go, a w którym nie zobaczymy jednak efektu naszej pracy. Zagadnieniu zintegrowanego testera strategii poświęcam oddzielny rozdział, w którym pokażę jego zalety, jak i wady, oraz omówię jego możliwości. Przetestujemy również strategię, którą wcześniej dokładnie objaśnię, z naciskiem na jej praktyczne zastosowanie.

Omówienie architektury klient-serwer

Platformy Meta Trader to nie tylko rozwiązania informatyczne, które każdy z klientów domów maklerskich widzi i instaluje na własnym komputerze lub też na urządzeniu mobilnym. Wręcz przeciwnie – struktura platformy transakcyjnej jest bardzo złożona, a składa się z dwóch głównych części:

1) **Backend**

- Meta Trader Manager,
- Meta Trader Administrator,
- Meta Trader Data Center,
- Meta Trader Server.

2) **Frontend**

- Meta Trader PC Client,
- Meta Trader Mobile Terminal.

Należy mieć na uwadze, że za każdym razem, kiedy uruchamiamy na swoim komputerze platformę, łączymy się z głównym serwerem, co możemy zobaczyć na schemacie obok.

Jest to dla nas o tyle istotne, o ile niektórzy brokerzy posiadają stosunkowo słabe sprzętowo serwery, przez co ulegają one częstym awariom. Dzięki temu, że wiemy, jak cała platforma działa wewnętrznie, możemy sobie napisać specjalny skrypt, za pomocą którego zbadamy tzw. responsywność serwera, czyli jak szybko nasz komputer łączy się z serwerem. W razie gdy stracimy połączenie, skrypt ten tworzy logi, które możemy użyć na wypadek reklamacji u naszego brokera.

Ogólnione przedstawienie architektury platformy Meta Trader.

Rodzaje plików w Meda Editor

Platforma tradingowa Meta Trader 4 pracuje głównie na dwóch rodzajach plików. Są to pliki z kodem źródłem *.mq4 (w przypadku MT5 – *.mq5) oraz skompilowane pliki wykonywalne *.ex4 (analogicznie dla MT5 – *.ex5), które bezpośrednio są wykonywane przez platformę. Główna różnica pomiędzy tymi dwoma rodzajami plików polega na tym, że pliki wykonywalne (*.ex4) nie mogą być edytowane. Wykonuje je zawsze procesor komputera, jako że zawierają kod binarny rozumiany tylko przez procesor.

Chciałbym w tym miejscu zwrócić uwagę na jedną bardzo ważną rzecz. Czasami jest mi wysyłany plik *.ex4 z prośbą o dokonanie „małej zmiany” w strategii, co jest technicznie bardzo trudnym zadaniem. Problem polega na tym, że aby dokonać zmiany mając do dyspozycji jedynie plik wykonywalny, najpierw musimy go poddać procesowi tzw. dekompilacji. Dekompilacja jest to odwrócenie procesu kompilacji plików źródłowych, czyli z pliku wykonywalnego otrzymujemy kod rozumiany przez człowieka. O ile mało wartościowe strategie w ogóle nie mają zabezpieczeń i proces dekompilacji okazuje się stosunkowo łatwy, to już w przypadku tych wartościowych staje się to dużo trudniejsze. Dzieje się tak, ponieważ autorzy dbają o swoją pracę (za kodem źródłowym kryją się pomysły, do których nierzadko dochodzą oni przez wieloletnie eksperymenty) i wiedzą, jak dobrze zabezpieczyć swój kod. Nie wdając się w szczegóły dodam, że najczęściej stosowane zabezpieczenia to biblioteki *.dll, napisane nisko poziomowo i zawierające tzw. salt code, czyli kod źródłowy, którego jednym celem jest „zaciemnienie” faktycznej logiki strategii. Innym bardziej zaawansowanym zabezpieczeniem okazuje się synchronizacja z zewnętrznym serwerem uwierzytelniającym.

W przypadku kiedy strategia nam się rozrasta do większych rozmiarów, posiadanie jednego ogromnego pliku źródłowego, zawierającego po 5000 linii, staje się bardzo niewygodne. Możemy wtedy wykorzystać pliki z rozszerzeniem *.mqh. Są to pliki, za pomocą których w bardzo wygodny sposób możemy zarządzać kodem źródłowym, poprzez podzielenie strategii na mniejsze funkcje, po czym poszczególne części logiki aplikacji umieścić w osobnych plikach *.mqh. Do tematu zarządzania oraz wersjonowania kodu źródłowego wrócę jeszcze w kolejnej części opracowania.

Ostatnim rodzajem plików, mniej związanych bezpośrednio z kodem źródłowym, są pliki *.tpl, w których Meta Trader przechowuje informacje o rozmieszeniu i poszczególnej konfiguracji środowiska tradera, tj. otwarte wykresy, otwarte wskaźniki na poszczególnych wykresach, domyślna wartość otwarcia lotu, wszystkie skróty klawiszowe danego użytkownika. Są to informacje o szablonach.

Lokalizacja plików źródłowych na dysku

W zależności od posiadanej przez nas wersji Meta Tradera, treść tego rozdziału może mniej lub bardziej pokrywać się z ogólnym widokiem waszej instalacji. Ogólnie, kiedy wejdziemy do katalogu, gdzie dokonaliśmy instalacji, zobaczymy najprawdopodobniej katalog o nazwie „MQL5", w którym znajdują się następujące podfoldery:

- **MQL5\Experts** – najbardziej interesujący dla nas katalog ponieważ znajdują się w nim wszystkie kody źródłowe oraz ich odpowiedniki w formie plików wykonawczych,
- **MQL5\Include** – znajdziemy tu wszystkie pliki nagłówkowe *.mqh i dobrą praktyką jest trzymać je właśnie w tym katalogu,
- **MQL5\Indicators** – znajdziemy tu kody źródłowy oraz pliki wykonywalne naszych wskaźników,
- **MQL5\Scripts** – podobnie jak w innych przypadkach, znajdują się tutaj pliki kodów źródłowych oraz ich skompilowane odpowiedniki.

Podkatalogów tych najprawdopodobniej będzie więcej, ale te powyższe są najważniejsze z punktu widzenia programowania strategii. Dla porządku wymienię te mniej istotne katalogi:

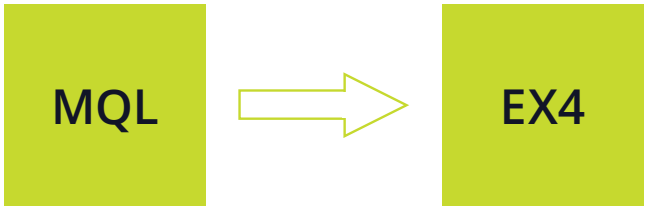
- **\MQL5\Files** – czasami potrzebujemy pracować na plikach zewnętrznych, np. w celu przechowania tymczasowych wyników potrzebnych do ostatecznego raportu,
- **\MQL5\Logs** – tutaj znajdziemy pliki tekstowe z informacją o funkcjonowaniu platformy,
- **\MQL5\Libraries** – do tego katalogu powinny zostać skopiowane biblioteki *.dll, jeśli wykorzystujemy w projekcie.

Podstawowe zagadnienia związane z programowaniem

Na wstępie chciałbym zasygnalizować, że przechodzimy właśnie do właściwej części książki. I od razu mała dygresja – kiedy studiowałem informatykę, przekonałem się, że większość prowadzących przekazywała wiedzę studentom w sposób bardzo skomplikowany, rzeczy proste były tłumaczone w sposób bardzo zawiły, z wykorzystaniem pojęć, które nadawały danej tematyce „magiczny" wydźwięk. Będąc na zagranicznych praktykach, a później już na stale pracując w dużej firmie, obserwując swoich współpracowników, nauczyłem się w sposób właściwy komunikować się, czyli w sposób maksymalnie prosty i przyswajalny. Pewnego dnia chciałem wytłumaczyć koledze pewne zawiłości bazy danych, na której pracowaliśmy. Po blisko dwugodzinnym spotkaniu, kiedy zostałem zmuszony odpowiadać na coraz to bardziej wyrafinowane pytania, uświadomiłem sobie jedną bardzo ważną dla mnie rzecz – jeśli nie umiem przekazać czegoś w sposób bardzo prosty, to znaczy, że nie rozumiem w pełni danego zagadnienia. Chciałbym, aby ta książka była bardzo łatwo przyswajalna i napisana w taki sposób, aby każdy był w stanie ją przeczytać i aby przyniosło to konkretny rezultat– Czytelnik nabędzie mocne podstawy programowania w MQL'u. Jeśli nie osiągnę tego celu, będzie to moja porażka, a nie Czytelnika.

Bez względu na to, jaki program piszemy, wcześniej czy później będziemy musieli go skompilować. Wspomogę się definicją z Wikipedii[2]:

> **Kompilator** to program służący do automatycznego tłumaczenia kodu napisanego w jednym języku (języku źródłowym) na równoważny kod w innym języku (języku wynikowym)[1]. Proces ten nazywany jest kompilacją.



Automat transakcyjny, jak każdy program komputerowy, pracuje na danych, które zazwyczaj są przechowywane w zmiennych. W związku z tym wyróżnić trzeba dwa pojęcia, które powinniśmy poznać: deklaracja zmiennej oraz inicjalizacja zmiennej. Warto je rozróżniać, aby uniknąć na wstępie nieporozumień. W MQL przed wykonaniem jakichkolwiek obliczeń każdą zmienną musimy zadeklarować.

2 https://pl.wikipedia.org/wiki/Kompilator

Deklaracja zmiennej jest to miejsce w kodzie, gdzie kompilator widzi zmienną, jednak jeszcze nie może określić jej wartości, z czym radzi sobie w ten sposób, że przypisuje jej wartość domyślną, zależną od typu danych:

```
int zmienna1; // jedna zmienna w jednej linijce kodu
int zmienna2, zmienna3, zmienna4; // parę zmiennych w jednej linii kodu
```

Inicjacja zmiennej oznacza zarówno określenie typu danej zmiennej, jak i również jej wartości początkowej: double zmienna5 = 5.2;

Za każdym razem, kiedy uruchomimy nasz program, kompilator będzie poinformowany o tym, że zmienna5 na samym początku uruchomienia automatu ma wynosić 5.2. Oczywiście w wyniku pewnych obliczeń (dalszego działania automatu) wartość ta może się zmieniać, jednak zawsze po ponownym uruchomieniu pozostanie taka rama, jak podczas inicjacji. Innymi słowami, typ danych zmiennej nie zmieni się nigdy.

CIEKAWOSTKA

Każdy typ danych w komputerze jest przechowywany inaczej, stąd każdy z operatorów (na przykład dodawania „+") inaczej wykonuje operacje dodawania na typie int, a inaczej na double.

Niezbędne pojęcia związane z programowaniem automatycznych strategii

Po wstępnym przedstawieniu programowania w MQL chciałbym zapoznać Czytelnika z pewnym z pozoru oczywistym zagadnieniem, jakim jest zmiana ceny. Temat ten, pozornie tylko oczywisty, wymaga rozwinięcia i wyjaśnienia. Jak wcześniej napisałem, Meta Trader, czyli środowisko, w którym będzie działał nasz automat, jest oparte na architekturze klient–serwer. Oznacza to, że serwer wysyła informacje do klienta, aby mógł on zobaczyć je na własnym komputerze i na ich podstawie był w stanie podejmować decyzje inwestycyjne, przy czym mam tu na myśli dane rynkowe dotyczące aktualnej ceny.

W tym momencie warto zadać pytanie, czym jest zmiana ceny z technologicznego punktu widzenia. Otóż zmiana ceny w kontekście programowania automatów transakcyjnych to tick (w MQL4 funkcja start () zostaje wywołana zawsze w momencie pojawienia się nowej ceny). Możemy powiedzieć, że tick to zmiana ceny w jednostce czasu, dokonana na podstawie informacji z serwera. W praktyce broker dysponujący u siebie w biurze serwerem, z którym każdorazowo łączymy się logując na nasz rachunek inwestycyjny, wysyła nam informacje o nowej cenie dla każdego z instrumentów, które mamy dostępne na platformie.

Typy danych w MQL

MQL jest językiem tzw. silnie typowanym . Ma to istotne konsekwencje dla pisanych przez nas programów, ponieważ musimy już na samym początku przemyśleć i przewidzieć, jaki typ danych będzie przechowywany w danej zmiennej. Może to mieć ogromne znaczenie choćby w przypadku ustalenia wielkości otwartej pozycji.

W MQL4 mamy do dyspozycji następujące typy danych:

- **Integer (int)** – liczba całkowita,
- **Boolean (boolean)** – logiczny typ danych, przechowuje dwie wartości: prawda lub fałsz,
- **Char (char)** – znakowy typ danych,
- **String (string)** – ciąg znaków,
- **Double (double)** – liczba zmiennoprzecinkowa,
- **Color (color)** – przechowuje informację o kolorze,
- **Data i czas (datetime)** – przechowuje informacje o dacie oraz czasie,
- **Enumerations (enum)** – typ wyliczeniowy.

Poniżej omówię pokrótce, czym charakteryzuje się każdy z nich. W celu zobrazowania, jak każdy z typów funkcjonuje w środowisku Meta Editor, do każdego z nich dołączę prosty przykład. Dzięki temu każdy będzie mógł poeksperymentować we własnym zakresie, do czego gorąco zachęcam.

Typ int – przedstawienie liczb całkowitych

W przypadku gdy chcemy wykorzystywać w swoim automacie liczby całkowite, będziemy definiować je w taki oto sposób:

```
int liczbaOtwartychPozycji = 5;
```

Powyżej zdefiniowałem zmienną o nazwie „liczbaOtwartychPozycji” oraz przypisałem do niej wartość 5. Ponieważ nie jest możliwe, aby uzyskać 5.5 lub 3.4 otwartej pozycji, dla tego typu zmiennej wybrałem typ całkowity – int.

MQL pozwala definiować liczby całkowite na dwa sposoby:

1. Dziesiętny (decimal) – wartości zmiennej reprezentowane są w sposób dziesiętny w zbiorze liczby od 0 do 10, mogą być to liczby dodatnie lub ujemne: 3, 5, –1, 7
2. Szesnastkowo (hexadecimal) – ta reprezentacja liczb jest już nieco mniej intuicyjna, jednak sposób rozpoznania jej bardzo łatwy – zawsze zaczyna się od 0x lub 0X, np. 0X1E. Osoby zainteresowane zachęcam do zapoznania się z tym systemem.

Dodam, że dokonałem w tym miejscu pewnego uproszczenia, z uwagi na rozmiary i cele tej pracy. MQL4 posiada kilka wariancji, który może przechowywać liczby całkowite. Więcej informacji znajdziecie w oficjalnej dostępnej online dokumentacji MQL4[4].

Typ bool – określenie prawdy lub fałszu

Typ logiczny bool może przechowywać jedną z dwóch wartości prawda (true) lub fałsz (false). Wewnętrznie typ bool jest przechowywany jako integer, czyli w postaci liczby. Są to odpowiednio wartości 1 oraz 0.

Przykład użycia:

```
Bool prawda = true;
Bool falsz = false;
Bool prawda = 1;
```

Stałe znakowe – char

Czasami istnieje potrzeba przechowania jednego znaku alfanumerycznego w zmiennej. Jednym z przykładów praktycznego użycia typu char jest sprawdzenie, czy w ciągu znaków znajduje się wcześniej zdefiniowana litera, co dokonuje się za pomocą jednej z pętli, które opiszę w kolejnych rozdziałach. Na tę chwilę wystarczy wiedzieć, że: mamy do dyspozycji taki typ, wewnętrznie MQL4 przechowuje tą wartość jako liczba z przedziału od -128 do 127 oraz zajmuje on 1 bajt w pamięci komputera.

Przykład użycia:

```
char szukanaLitera = 'A';
char dolar = '$';
```

Należy wspomnieć, iż komputer nie potrafi przechowywać w swojej pamięci jakiejkolwiek litery czy znaku ‚$'. Wszelkie znaki w pamięci komputera funkcjonują w postaci liczby, stąd też wszystkie litery alfabetu oraz znaki specjalne są po prostu ponumerowane. To ten numer (kod) danego znaku jest właśnie przechowywany w pamięci.

WSKAZÓWKA

Jest dostępnych wiele sposobów numerowania (kodowania) znaków. Jednym z najszerzej stosowanych jest kod ASCII[3].

Typ String – ciąg znaków

Jak zapewne się domyślacie, typ string służy do przechowywania ciągu znaków. Stosując termin ciąg znaków mam na myśli kolejno ustawione znaki ASCII zapisane w podwójnym cudzysłowie, np. „Witaj drogi Czytelniku". Wewnętrznie MQL4 przechowuje zmienną łańcuchową jako tablicę znaków zawartych w cudzysłowie, która zajmuje 8 bajtów pamięci komputera. Z danych typu string będziemy korzystać bardzo często, czego przykładem są chociażby wszystkie informacje, jakie są wyrzucane z platformy do programisty:

```
String potwierdzenieZlecenia = „Złożono zlecenie kupna EURUSD 1.35 lot po cenie 1.23435";
```

Ponieważ string to ciąg znaków rozpoczynający się oraz kończący znakiem cudzysłowu, moglibyśmy sobie zadać zatem pytanie, co dzieje się w przypadku, kiedy chcemy wyświetlić cudzysłów wewnątrz stringu, który chcemy wyświetlić. Naturalnie jest taka możliwość, a żeby to objaśnić, wprowadzę pojęcie *znaku specjalnego*[4]. Znakiem specjalnym jest właśnie znak cudzysłowu. Aby wyświetlić znak cudzysłowu w stringu, należy przed każdym znakiem specjalnym napisać znak backslashu w pojedynczym cudzysłowie:

```
string znakSpecjalny = „Przykładowy znak specjalny ,\"'";
```

Rozróżniamy ponadto trzy główne grupy typów danych w MQL: int, double oraz string. Pozostałe, np. bool, color oraz date, stanowią typ integer, czyli mają postać liczbową.

3 http://articles.mql4.com/457

4 http://docs.mql4.com/basis/types/integer/symbolconstants

Typ double / float – liczby zmiennoprzecinkowe

MQL daje nam do dyspozycji dwa typy liczb: double oraz float, przydatne w sytuacji, kiedy potrzebujemy wprowadzić zmienną, która prawdopodobnie będzie przyjmować wartości z częścią dziesiętną po przecinku. Różnica pomiędzy nimi jest taka, że double może zawierać szerszy przedział liczbowy oraz zajmuje więcej miejsca w pamięci komputera. W środowisku programistów o typie double czasami mówi się jako o zmiennej zawierającej podwójną precyzję. Gdy istnieje więc potrzeba skorzystania ze zmiennej, od której wymagamy dużej precyzji przechowywanych wartości, zalecam korzystanie z typu double. Z własnego doświadczenia wiem, że w przypadku gdy prowadzimy bardziej skomplikowane obliczenia, typ float nie wystarcza, co może przysporzyć nam mnóstwa bardzo trudnych do wykrycia problemów. Niestety kompilator w MQL nie jest bowiem na tyle inteligentny, aby ostrzec nas przed niedostateczną dokładnością wykonywanych przez nas operacji, co najwyżej obetnie on część, której już nie może przechować, bez żadnego komunikatu. Takie działanie kompilatora może nas zmylić i sprawić, że wyciągniemy z tego błędne wnioski. Skorzystanie z typu double chroni nas przed tego typu niebezpieczeństwem.

Aby zdefiniować zmienną typu double, wystarczy napisać taki oto kod:

```
double wielkoscPozycji = 0.7;
```

Należy podkreślić, że część dziesiętną oznaczamy za pomocą kropki, a nie przecinka.

Typ color

Bardzo często wykorzystywanym typem danych jest **color**. Typ ten przechowuje informacje o kolorze, dzięki czemu możemy „pokolorować" wcześniej wyliczony poziom cenowy w taki sposób, aby informacja, jaką chcemy uzyskać, była intuicyjna i szybko osiągalna dla tradera. Przykładowo, przy wyznaczaniu poziomów cenowych oporu oraz wsparcia przyjęło się, że opór oznaczamy kolorem czerwonym, a wsparcie kolorem zielonym. Zmienne typu color mogą być przechowywane na trzy sposoby: explicite jako kolor, jako liczba int lub otrzymywana za pomocą **rzutowania**, co wyjaśnię poniżej.

Typ color zajmuje 4 bajty w pamięci komputera.

Bardzo często potrzebujemy przekonwertować jeden typ danych na inny. W takim przypadku należy wykonać tzw. rzutowanie, czyli konwersję jednego typu na inny. Rozróżniamy dwa typy **rzutowania**: z utratą informacji oraz bez utraty. Kiedy wykonujemy rzutowanie pierwszego typu, musimy pogodzić się z faktem, że utracimy część danych, jaką przechowujemy w zmiennej, czego przykładem jest konwersja zmiennej

z typu double do typu int – stracimy wtedy informacje będące po przecinku w zmiennej typu double:

```
doublezmiennaTestowa = 5.4;
intnaszaNowaZmienna = MathFloor(zmiennaTestowa);
MessageBox(„Zmienna typu int:” + naszaNowaZmienna); // efektem wykonania kodu jest konwersja liczby 5.4 do liczby typu int czyli 5.
```

Na powyższym przykładzie dokonaliśmy rzutowania za pomocą funkcji ***MathFloor***[5] . Możemy również korzystać z rzutowania w sposób, do którego jesteśmy przyzwyczajeni, w takich językach jak C/C++ lub młodszych, takich jak Java. Mam tu na myśli tzw. rzutowanie jawne, gdzie ogólny schemat wygląda następująco:

```
zmienna1 = (typNaJakiChcemyDokonacRzutowania) zmienna2;
```

lub

```
zmienna1 = typNaJakiChcemyDokonacRzutowania(zmienna2);
```

Zachęcam do czytania dokumentacji na ten temat pod linkiem[6].

Typ Datetime– czyli jak przechować datę i czas

Może się zdarzyć, że w naszym automacie będzie konieczne przypisanie do zmiennej konkretnej daty, np. w przypadku kiedy będziemy chcieli znaleźć maksimum danego waloru w określonym czasie, po implementacji stosunkowo łatwego algorytmu wyszukania maksimum (iteracja po cenach zamknięcia w określonym przedziale czasowym, zakładając że każda kolejna wartość wyznacza maksimum, pod warunkiem, że wartość poprzednia jest mniejsza). Do tego celu będziemy potrzebować sposobu na przechowanie danych czasowych, do czego służy typ datetime. Zmienne, które oznaczymy jako datetime, zawsze będą składać się z sześciu części:

- roku
- miesiąca
- dnia
- godziny
- minuty
- sekundy

5 http://docs.mql4.com/math/mathfloor

6 http://docs.mql4.com/basis/types/casting

Istnieje również możliwość zastosowania innego formatu daty w następującym układzie: dzień, miesiąc, rok, godzina, minuta, sekunda. Wybór zależy od naszych potrzeb lub upodobań. Należy tylko podkreślić ważną kwestię, że każda z dat musi zostać poprzedzona wielką literą D, tak jak na poniższych przykładach:

```
datetimenowyRok=D'2015.01.01 00:00';    //    Data rozpoczęcia Nowego Roku 2015
datetime data1 = D'1980.07.19 12:30:27'; //    Rok Miesiąc Dzień Godzina Minuta Sekunda
datetime data2 = D'19.07.1980 12:30:27'; //    Alternatywny sposób zapisu daty
datetime data3 = D'19.07.1980 12';       //    Identyczne znaczenie co D'1980.07.19 12:00:00'
datetime data4 = D'01.01.2004';          //    Identyczne znaczenie co D'01.01.2004 00:00:00'
datetime compilation_date = __DATE__; //    Data kompilacji
```

Typ ten zajmuje 8 bajtów w pamięci komputera. Wykorzystywany przez nas zakres daty i czasu powinien się zawierać pomiędzy 1 stycznia 1979 roku a 31 grudnia 2037 roku. Wewnętrzna reprezentacja datetime to zmienna integer zajmująca 4 bajty. Wartość datetime określona jest z kolei jako liczba sekund, jaka upłynęła od godziny 00:00 1 stycznia 1970.

Typ Enum– zachowujemy porządek w kodzie

Dobrą praktyką podczas pisania oprogramowania jest dążenie do tego, aby kod był przede wszystkim łatwy w czytaniu oraz sprzyjał zrozumieniu logiki danego fragmentu. Świetnym narzędziem do tego celu jest typ enum. Ogólny schemat jego wykorzystania wygląda następująco:

```
enum nazwa typu wyliczeniowego
 {
lista wartosci
 };
```

Praktyczne zastosowanie typu wyliczeniowego może z kolei wyglądać następująco:

```
enum dzienTygodnia

M=1,    // poniedziałek
T=2,    // wtorek
W=3,    // środa
Th=4,   // czwartek
Fr=5,   // piątek
St=6,   // sobota
S=7,    // niedziela
 };
```

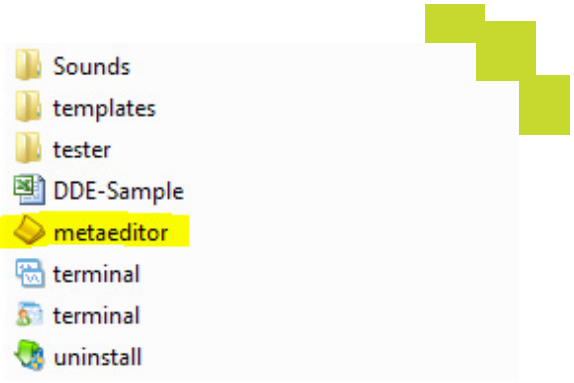
Do zagadnienia typu wyliczeniowego wrócę w dalszych rozdziałach pracy[7].

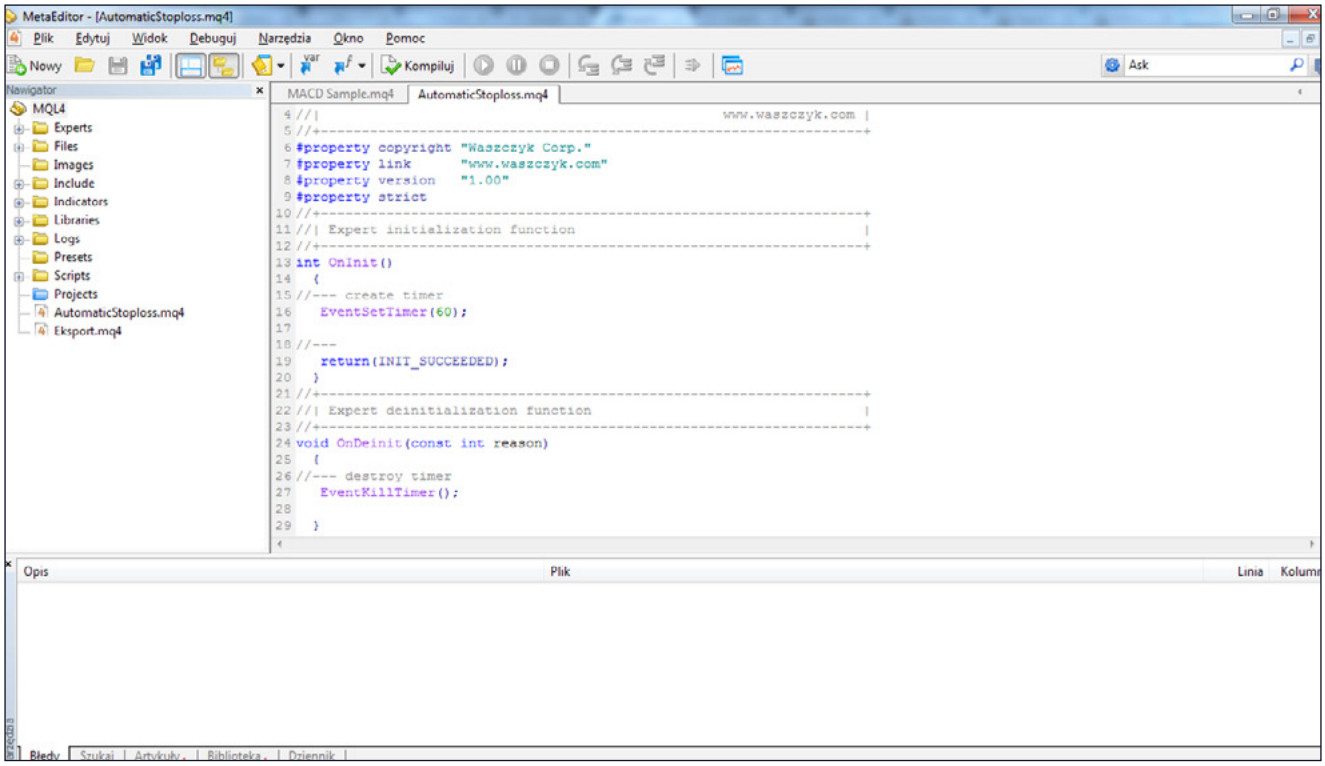
7 http://docs.mql4.com/basis/variables/externvariables

Meta Editor jako nasze podstawowe środowisko pracy

Celem tego rozdziału będzie zapoznanie programisty ze środowiskiem. Opiszę pokrótce korzyści z korzystania z Meta Editora, jego zalety i wady. Należy wiedzieć, że instalując Meta Trader instalujemy również oprogramowanie służące do programowania w MQL. Możemy je uruchomić na dwa sposoby:

1. Wchodząc do Meta Trader, a następnie nacisnąć przycisk F4,
2. Bezpośrednio z katalogu instalacyjnego: klikając na zaznaczoną na żółto ikonę podpisaną „metaeditor". Bez względu na wybrany sposób uruchomienia, naszym oczom powinien ukazać się podobny obraz:



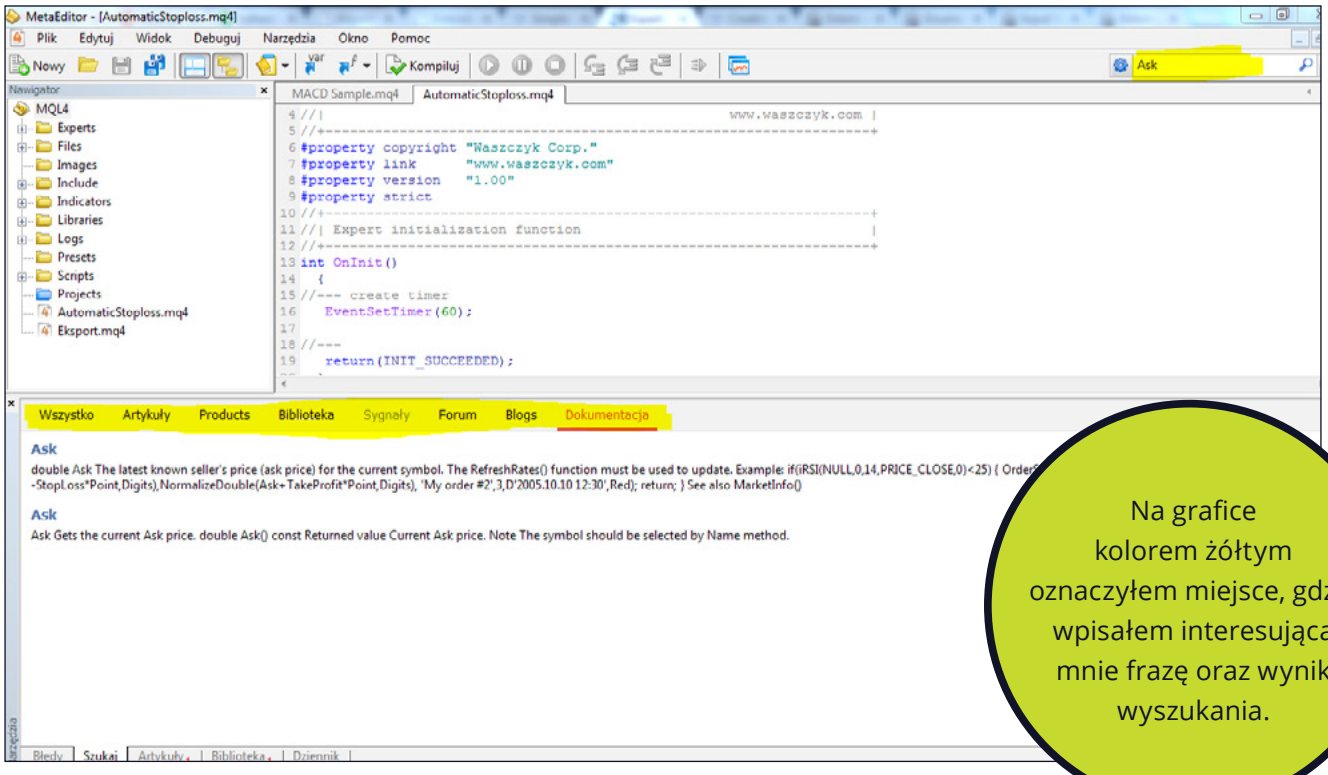


MetaEditor to zintegrowane środowisko programistyczne (Integrated Development Environment- IDE) dla języka MQL, w którym będziemy programować nasze strategie automatyczne. Korzystając z tego

środowiska podczas programowania mamy do dyspozycji kilka narzędzi, które umożliwiają programiście bardziej efektywną, a do tego przyjemniejszą pracę, z uwagi na szereg udogodnień, jakimi są:

- możliwość posiadania otwartych kilku plików źródłowych,
- narzędzie wyszukiwania konkretnej linijki kodu (CTRL + G),
- autouzupełnianie kodu źródłowego (CTRL + Spacja),
- szybki dostęp do pełnej dokumentacji MQL, przy czym aby z niej skorzystać, należy zaznaczyć interesującą nas funkcję oraz wcisnąć przycisk F1.

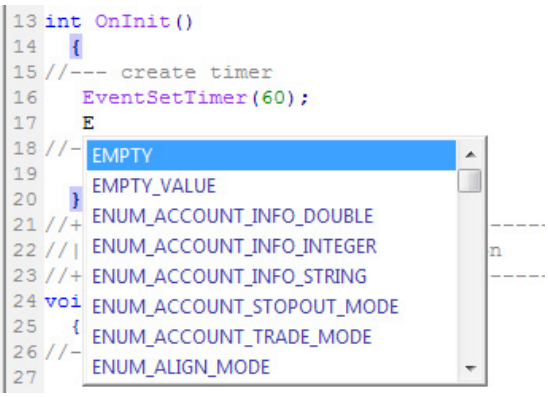
Za pomocą zakładki „Błędy", znajdującej się oknie oznaczonym jako „Narzędzia", programista odczytuje wszelkie komunikaty dotyczące kompilacji kodu. Dzięki temu możemy w łatwy sposób dowiedzieć się, w której linii popełniliśmy błąd, oraz znaleźć sugestię, jak go naprawić. Z poziomu zakładki „Szukaj" mamy dostęp do globalnej bazy danych – wystarczy wpisać np. „Ask" i potwierdzić klawiszem Enter, aby przeszukać wszystkie bazy: Artykuły, Produkty, Biblioteka itp. Wymagany jest przy tym dostęp do sieci Internet:



Na grafice kolorem żółtym oznaczyłem miejsce, gdzie wpisałem interesującą mnie frazę oraz wynik wyszukania.

Kiedy klikniemy na zakładkę „Dziennik" (ang. Journal), uzyskamy dostęp do logów Meta Editora. To właśnie tu możemy wysyłać wszelkiego rodzaju komunikaty oraz informacje z naszego automatu. Bardzo przydatnym narzędziem są również podpowiedzi kontekstowe wewnątrz edytora, dzięki którym możemy szybciej pisać kod źródłowy.

Może to również pomóc w przypomnieniu sobie nazwy funkcji, jeśli jej zapomnimy. Aby zaakceptować podpowiedź, należy potwierdzić przyciskiem Enter.

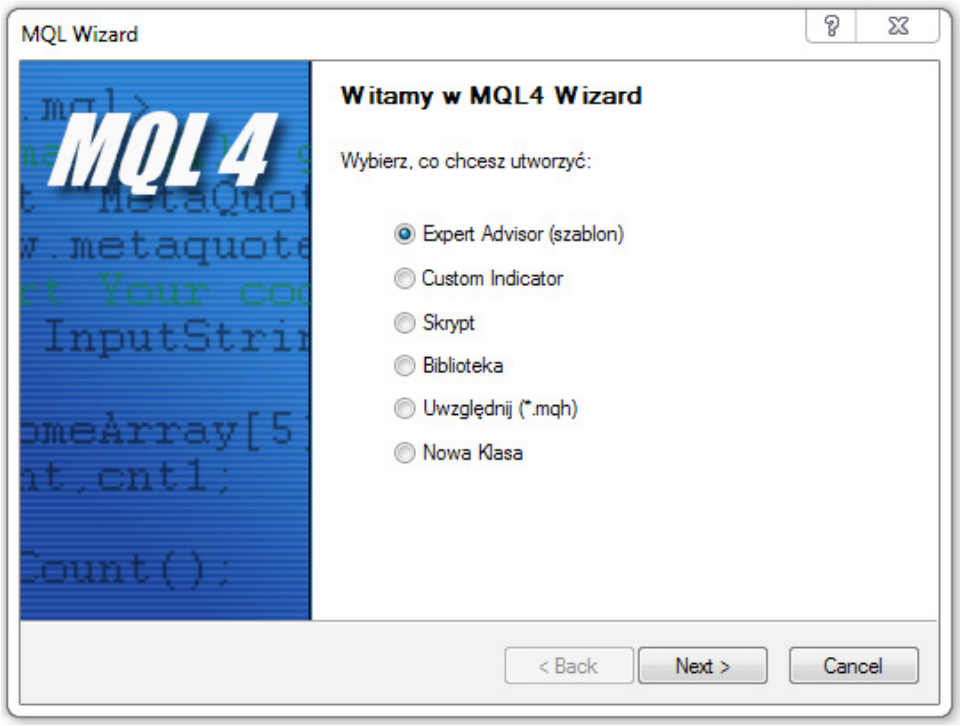


Wprowadzenie do programowania w MQL- pierwszy Expert Advisor

W tym rozdziale przejdziemy przez cały proces tworzenia automatu od początku, po czym skompilujemy go i uruchomimy w obrębie MT. Efektem tych działań będzie możliwie najprostszy Expert Advisor, który po skompilowaniu i uruchomieniu na wykresie pokaże okienko z napisem „Witaj Świecie”.

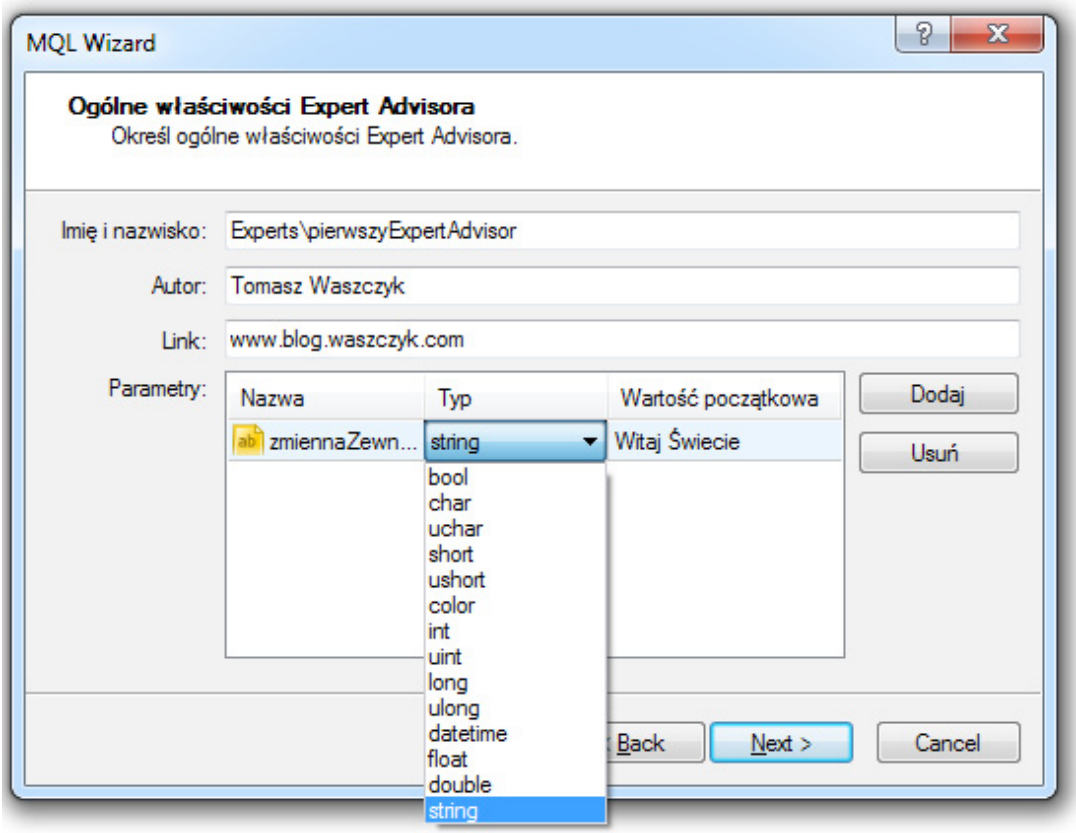
Kiedy mamy już uruchomiony MetaEditor, należy kliknąć przycisk 

, po czym powinniśmy zobaczyć następujące okno:



Najprawdopodobniej będziemy mieć automatycznie wybraną opcję „Expert Advisor (szablon)”. Jeśli jednak tak nie jest, proszę ją wybrać zgodnie z instrukcją pokazaną na obrazku powyżej oraz kliknąć przycisk „Next>”.

Efektem tego powinno być pojawienie się poniższego okienka, które proszę wypełnić w następujący sposób:

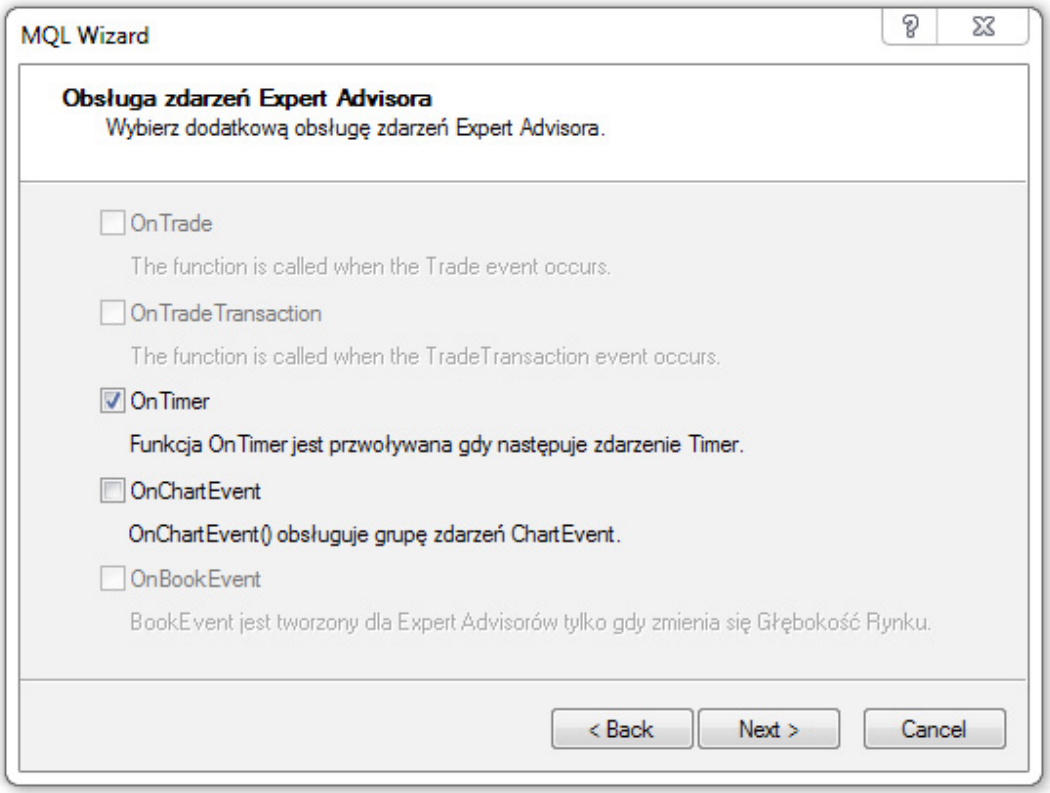


Pole podpisane „Imię i nazwisko" okazuje się tu mylące i błędnie opisane, ponieważ w tym polu wpisujemy nazwę naszego automatu – w powyższym przypadku wpisałem nazwę „pierwszyExpertAdvisor". Napis „Experts\" poprzedzający moją nazwę oznacza katalog, gdzie ma być umieszczony nasz automat.

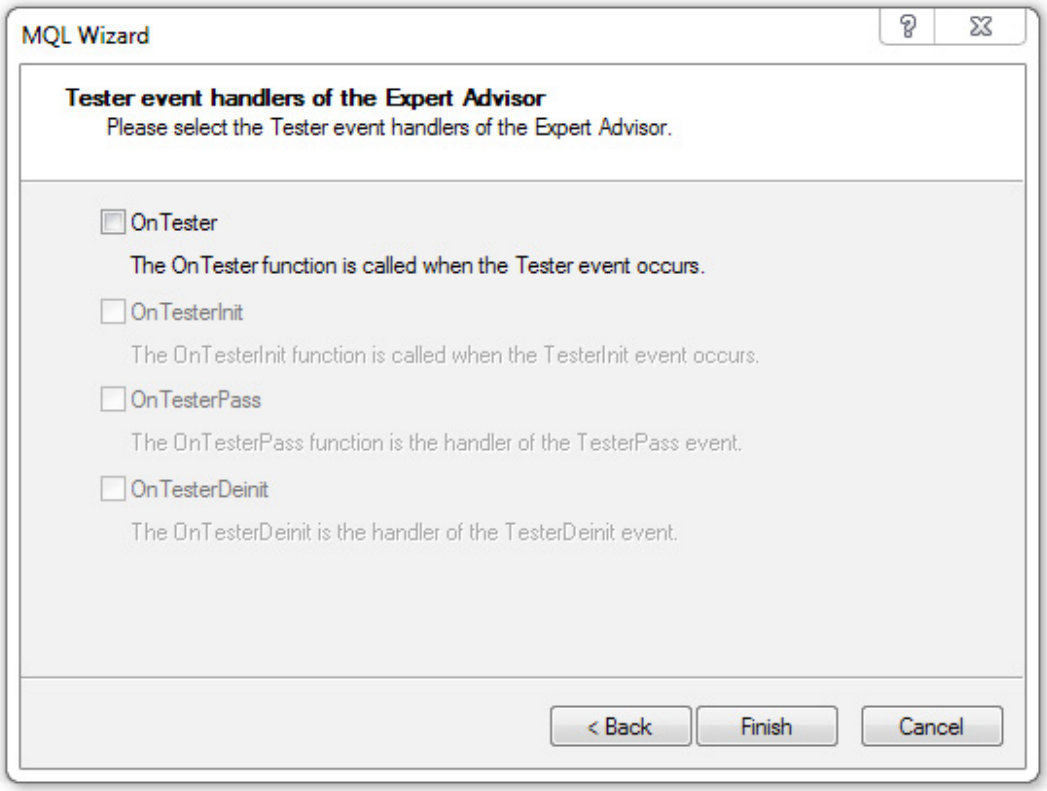
Interesującym polem jest również to oznaczone jako „Parametry", w którym określamy tzw. **zmienne zewnętrzne**[8] lub **zmienne wejściowe**[9]. Na powyższym obrazku widać, że kolumna „Typ" rozwija się, ukazując różne typy danych do wyboru, co stanowi praktyczny sposób zastosowania typu enum. Jeśli określamy typy zewnętrzne, po przeciągnięciu strategii na wykres pojawia się okienko, w którym możemy ustawić wartości początkowe tych zmiennych. Pokażę to podczas uruchamiania naszego pierwszego automatu.

8 http://docs.mql4.com/basis/variables/externvariables
9 http://docs.mql4.com/basis/variables/inputvariables

Tymczasem klikamy przycisk „Next>„ i pokazuje nam się takie oto okno:



Pozostawiamy domyślne ustawienia oraz raz jeszcze klikamy „Next>„:

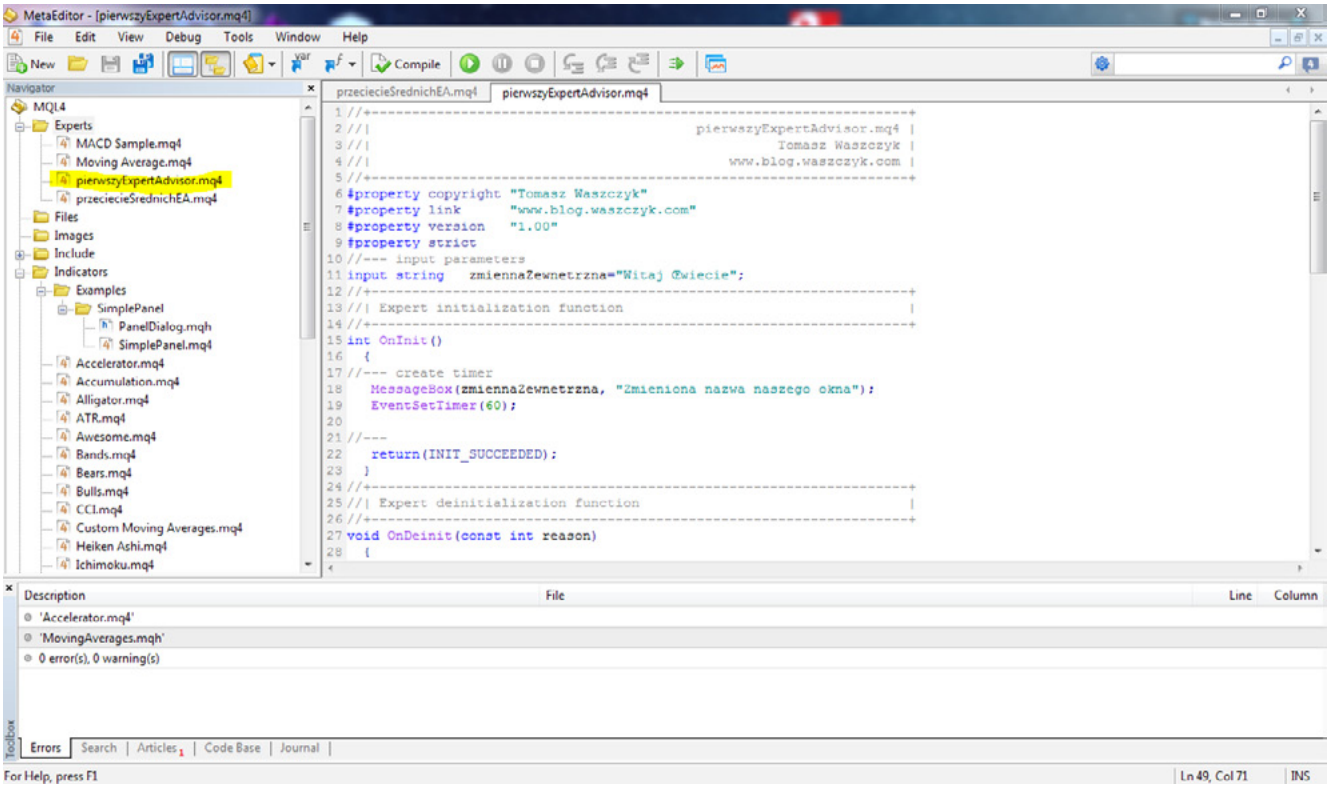


Wreszcie, pozostawiając domyślne ustawienia, klikamy przycisk „Finish”, w efekcie czego powinno pokazać się otwarte okno z plikiem „pierwszyExpertAdvisor.mq4”, z następującym kodem źródłowym:

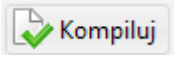
```
//+------------------------------------------------------------------+
//|                                    pierwszyExpertAdvisor.mq4 |
//|                                                 Tomasz Waszczyk |
//|                                       www.blog.waszczyk.com |
//+------------------------------------------------------------------+
#property copyright „Tomasz Waszczyk”
#property link      „www.blog.waszczyk.com”
#property version   „1.00”
#property strict
//--- inputparameters
input string   zmiennaZewnetrzna=”Witaj Świecie”;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
 {
//--- create timer
EventSetTimer(60);

//---
   return(INIT_SUCCEEDED);
 }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(constint reason)
{
//--- destroy timer
EventKillTimer();
 }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
{
//---

 }
```

```
//+------------------------------------------------------------------+
//| Timer function                                                   |
//+------------------------------------------------------------------+
void OnTimer()
{
//---

 }
//+------------------------------------------------------------------+
```

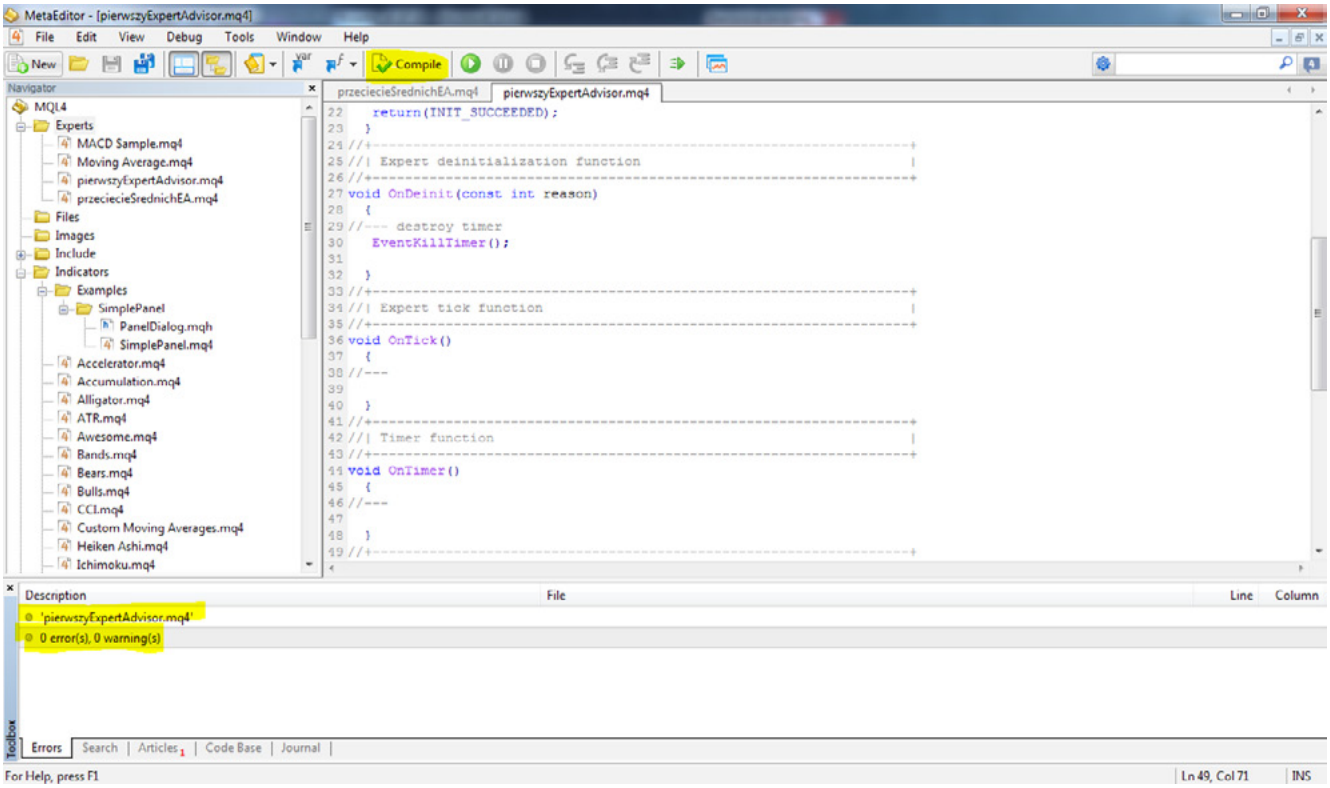
Efekt, jaki powinniśmy otrzymać na naszych komputerach, prezentuje się następująco:



Proszę zwrócić uwagę, że plik „pierwszyExpertAdvisor.mq4” znajduje się w katalogu „Experts”, czyli tam, gdzie być powinien. Czasami jednak z nieznanych mi powodów pojawia się na samym dole drzewa katalogów – w tym przypadku byłby pod plikiem „Eksport.mq4”. Chciałbym także zwrócić uwagę na fakt, że na tym etapie pracy nie ma jeszcze pliku wykonywalnego (z rozszerzeniem *.ex4), a to dlatego, że nie dokonaliśmy jeszcze kompilacji naszego kodu źródłowego. Dopiero po skompilowaniu możemy oczekiwać, że pojawi się plik „pierwszyExpertAdvisor.ex4”.

Kompilacja jest bardzo prostym procesem, pod warunkiem, że nasz kod źródłowy nie zawiera błędów. Wystarczy nacisnąć przycisk 

lub użyć skrótu klawiszowego F7. Po wykonaniu tej operacji widzimy następujący obraz:

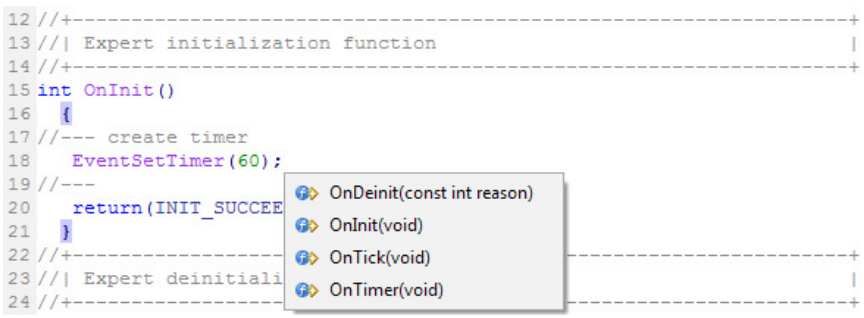


Udana kompilacja szablonu EA.

Możemy być z siebie dumni! Właśnie stworzyliśmy pierwszy automat, który został poprawnie skompilowany, potwierdzeniem czego są komunikaty widoczne w zakładce „Błędy” (Errors):

,pierwszyExpertAdvisor.mq4’ pierwszyExpertAdvisor.mq4
0 error(s), 0 warning(s)

Komunikat wyraźnie informuje, że podczas kompilacji nie wystąpił żaden błąd ani ostrzeżenie. Po wciśnięciu CTRL + M widzimy listę standardowych funkcji, które na początku są puste:



Lista funkcji do implementacji w standardowym szablonie Expert Advisor.

Jak widać, Meta Editor wygenerował nam cztery funkcje:

1. OnDeinit(),
2. OnInit(),
3. OnTick(),
4. OnTimer().

Pojęcie funkcji, a także sposoby jej implementacji i wywoływania opiszę w następnym podrozdziale. Naszym celem jest wyświetlenie komunikatu w oknie platformy transakcyjnej. Aby to zrobić, należy skorzystać z funkcji MessageBox , która pozwala na utworzenie własnego okna dialogowego. Funkcję tę implementuję w funkcji OnInit().

Aby wyświetlić nasze okno dialogowe, musimy to zaimplementować wewnątrz funkcji OnInit() w następujący sposób:

MessageBox (zmiennaZewnetrzna, „Nazwa naszego nowego okna”);

WSKAZÓWKA

Każda linijka kodu musi być zakończona średnikiem, z wyjątkiem deklaracji funkcji, a także otwarć oraz zamknięć funkcji. W poniższym kodzie źródłowym są to linie nr 15, 16 oraz 22.

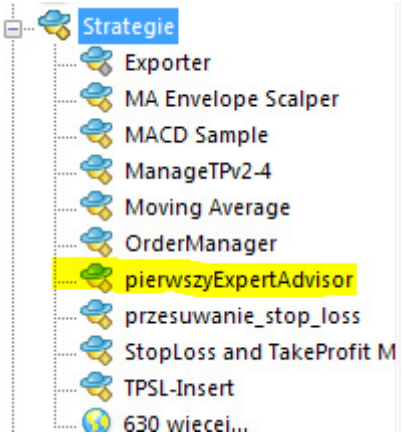
```
12 //+------------------------------------------------------------------+
13 //| Expert initialization function                                   |
14 //+------------------------------------------------------------------+
15 int OnInit()
16   {
17 //--- create timer
18    EventSetTimer(60);
19    MessageBox(zmie
20 //---                zmiennaZewnetrzna
21    return(INIT_
22   }
```

Kiedy już uzupełniliśmy nasz kod o funkcję MessageBox, kompilujemy raz jeszcze.

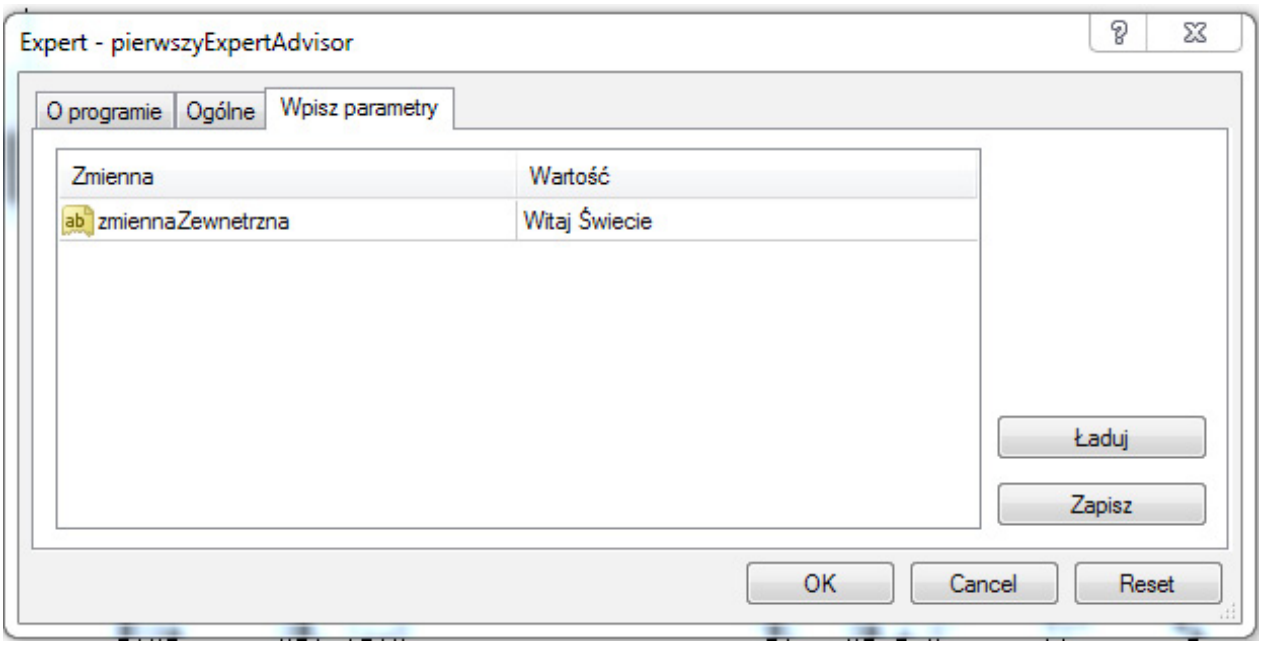
WSKAZÓWKA

Po każdej zmianie kodu musimy przeprowadzać kompilację.

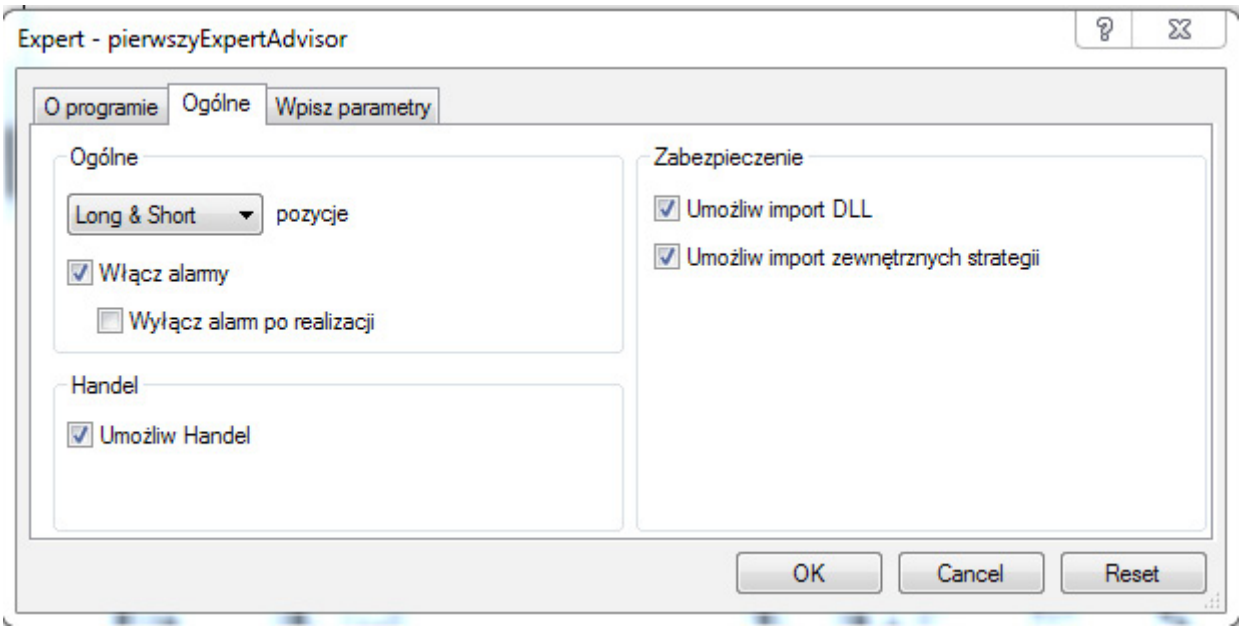
Kiedy ponownie ujrzymy komunikaty potwierdzające bezproblemową kompilację, jesteśmy gotowi do tego, by uruchomić nasz automat na wykresie. W tym celu przełączamy się do Meta Tradera oraz wyszukujemy na liście „Strategie" nazwy naszego programu typu Expert Advisor



Aby zakończyć procedurę uruchomienia w pełni działającego, niezawierającego na razie żadnych transakcji automatu, należy metodą złap i upuść (drag and drop) przenieść nasz plik na wybrany przez nas wykres waloru, na którym chcemy uruchomić automat. Efektem tej operacji będzie pojawienie się poniższego okienka:



Komunikat wewnątrz okienka kieruje do nas pytanie, czy na pewno chcemy uruchomić EA. Mamy też możliwość wprowadzenia początkowych wartości wcześniej zdefiniowanych zmiennych. Zakładka „Ogólne” powinna być skonfigurowana w sposób następujący:

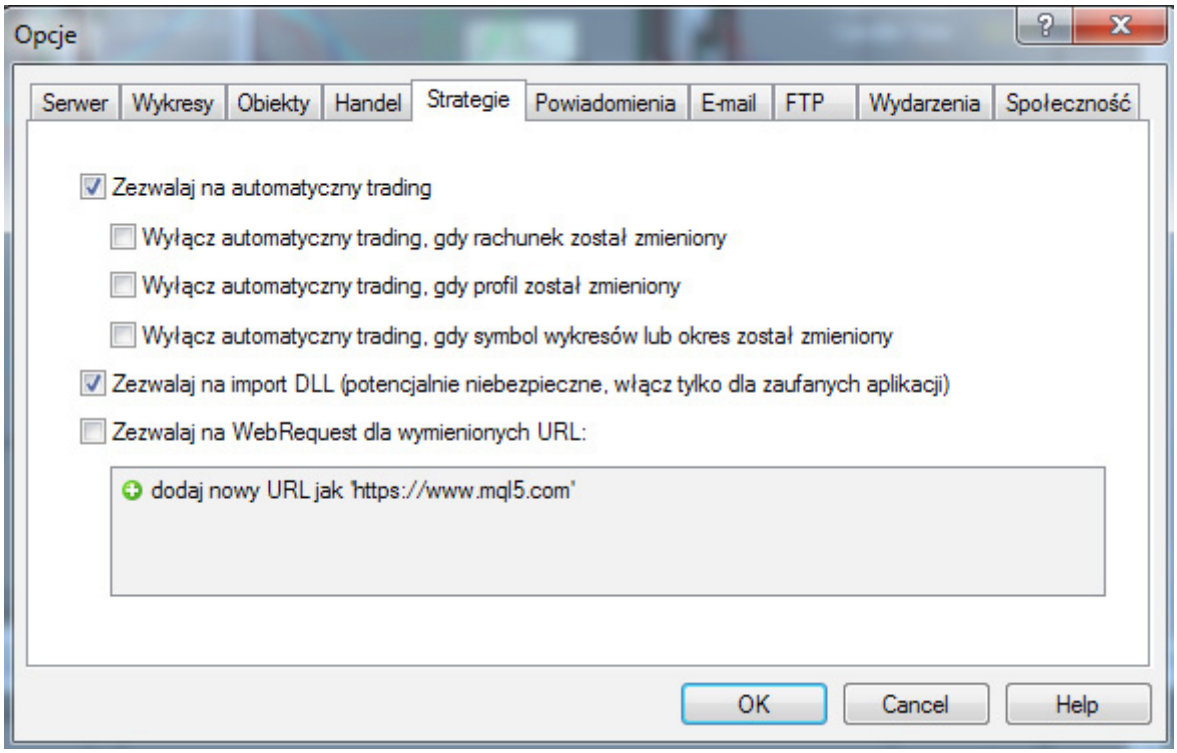


Opisy dostępnych ustawień wydają się dość czytelne i nie trzeba objaśniać ich znaczenia. Ostatecznie potwierdzamy klikając przycisk „OK”. Oto, co jest efektem naszego wcześniej skompilowanego programu:

Jak widzimy, pokazało nam się okienko z napisem „Witaj Świecie”, co jest zgodne z nazwą zdefiniowaną w kodzie źródłowym. Bardziej istotny wydaje się sposób, w jaki platforma powiadamia nas, że strategia została uruchomiona na wykresie. Odbywa się to za pomocą

Uwaga: Aby uruchomić jakikolwiek Expert Advisor w programie Meta Trader, najpierw musimy pozwolić platformie na handel automatyczny. W celu aktywowania możliwości handlu należy:

1. Uruchomić program Meta Trader.
2. Z menu kliknąć przycisk „Narzędzia”.
3. Z listy rozwijanej wybrać opcje.
4. Przejść do zakładki „Strategie” oraz skonfigurować ją w następujący sposób:

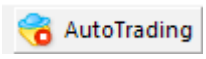


Jest to poprawna minimalna konfiguracja, niezbędna do tego, aby MT mógł handlować automatycznie (z użyciem automatu).

W momencie kiedy robot handlujący działa i jest aktywny, pojawia się oznaczenie tego faktu w prawym górnym rogu MACD Sample☺ . Widoczny tekst przed uśmiechniętą „buźką” powinna stanowić nazwa „pierwszyExpertAdvisor” – w podanym przykładzie jest to standardowy robot. Jeśli natomiast z jakichś przyczyn chcemy robota handlującego zatrzymać, powinniśmy kliknąć przycisk AutoTrading . Potwierdzeniem, iż automat został wyłączony, jest oznaczenie w prawym górnym roku ekranu danego waloru MACD Sample☹ . Aby ponownie uruchomić wcześniej załadowaną strategię, klikamy AutoTrading .

Poniżej widzimy zaznaczone w odpowiednich miejscach poszczególne przyciski oraz oznaczenia:

WSKAZÓWKA

W momencie kiedy klikamy przycisk 

,po zatrzymaniu aktywnej strategii w zakładce musi zostać wygenerowany log o następującej treści: „Automated trading disabled”, co pokazuje powyższy zrzut ekranu. Wystarczy w tym celu prześledzić logi platformy, które znajdują się pod zakładką „Strategie”.

Podstawy programowania

Opowiedzieliśmy już sobie o różnych dostępnych typach danych w MQL, nie wspomnieliśmy jednak o tzw. zakresie zmiennych, który określa, jak długo kompilator będzie trzymać zmienną wraz z jej wartością w pamięci. W MQL, podobnie jak w innych językach programowania, wyróżniamy dwa rodzaje zasięgu zmiennych:

1. zmienne lokalne,
2. zmienne globalne.

Zmienne lokalne

Zmienna lokalna to taka, która jest zadeklarowana wewnątrz funkcji, a której zasięg obejmuje ciało funkcji. Zmienne lokalne są przechowywane w pamięci zarezerwowanej dla tej funkcji, wewnątrz której dana zmienna została zadeklarowana.

Przykład:

```
int naszaFunkcja()
{
int naszaZmiennaLokalna = 0;
  ...
  return(naszaZmiennaLokalna);
}
```

W momencie, kiedy kompilator wychodzi poza zasięg funkcji, wartości zmiennych są usuwane z pamięci, co ilustruje poniższy kod:

Przykład:

```
voidOnStart()
{
int i=1;     // Zmienna lokalna zdefiniowana wewnątrz funkcji
  {
int i=10;  // Zmienna wewnątrz bloku kodu
Print(„Wewnątrz bloku wartość i wynosi = „, i);   // Wynik działania  i=10;
  }
Print(„Na zewnątrz bloku wartość i wynosi = „, i);  // Wynik działania  i=1;
}
```

Zmienne globalne

Zmienna globalna to taka, w przypadku której deklaracja dokonuje na zewnątrz jakiejkolwiek funkcji. Zmienne tego typu definiuje się na takim samym poziomie w kodzie źródłowym co funkcje. Nie są lokalne w żadnym z bloków kodu źródłowego:

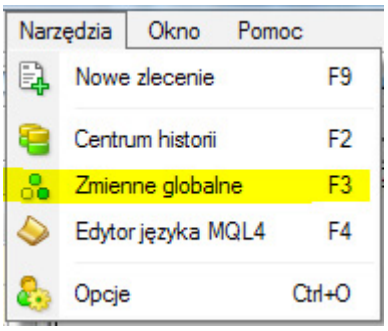
Przykład:

```
int naszaZmiennaGlobalna = 1;   // Zmienna globalna
int OnStart()
{
  ...
}
```

Zasięgiem każdej zmiennej globalnej jest cały program, przy czym zasięg oznacza, z jakiego miejsca w kodzie mamy dostęp do tej zmiennej. Co istotne, wewnątrz każdej funkcji możemy się odnieść do zmiennej globalnej. W momencie gdy nie deklarujemy, jaka ma być wartość zmiennej globalnej, domyślnie wynosi ona 0.

WSKAZÓWKA

Zmienne globalne możemy również stworzyć, a następnie przejrzeć sobie po naciśnięciu przycisku F3 z poziomu Meta Trader lub klikając zakładkę „Narzędzia", a następnie wybierając „Zmienne globalne":



Komentarze

Komentarz jest bardzo użytecznym narzędziem stosowanym w celu dokumentacji kodu źródłowego. Często komentuje się też dany fragment kodu, który z jakichś przyczyn tymczasowo wolimy nie kompilować. Aby zakomentować tylko jedną linijkę kodu, używamy dwóch znaków „/".

Przykład:

```
// To jest nasz komentarz
```

Aby zakomentować więcej niż jedną linię kodu, możemy wykorzystać tzw. komentarz blokowy, który wygląda następująco:

Przykład:

```
/* To jest
nasz długi komentarz
blokowy
*/
```

Komentarz blokowy rozpoczynamy znakiem „/*" a kończymy „*/". Możemy go też rozszerzać na dowolną ilość linii kodu.

Nazwy zmiennych oraz funkcji - identyfikatory

W MQL istnieją pewne ograniczenia odnośnie nazw zmiennych oraz funkcji. Mianem identyfikatorów określa się nazwy zmiennych oraz implementowanych przez nas funkcji, które mogą się składać z dowolnej kombinacji liter, cyfr oraz znaku podkreślenia „_". Długość identyfikatora jest ograniczona do 63 znaków oraz nie może mieć takiej samej nazwy jak **nazwa zastrzeżona**[10].

Funkcje

Bazą do dalszych rozważań będzie ponadto proces kompilacji oraz techniczne zagadnienia, których znajomość pozwala na uruchomienie „czegoś, co działa", przedstawione we wcześniejszych podrozdziałach. Odtąd teorii będzie mniej, a my skupimy się na omawianiu poszczególnych zagadnień wraz z przykładowymi kodami źródłowymi.

Zapewne zauważyłeś, że opisując proces uruchomienia najprostszego EA parokrotnie użyłem określenia ***funkcji***. W niemalże każdym języku programowania jednym z podstawowych składników są właśnie funkcje. Jest tak przynajmniej z kilku przyczyn:

1. Możliwość podzielenia większego zadania na łatwiejsze podzadania, które ułatwiają implementację wymaganej logiki,
2. Możliwość szybkiego wykorzystania kodu źródłowego napisanego przez innego programistę,
3. Łatwość podczas dokonywania zmian w kodzie źródłowym.

10 https://www.mql5.com/en/docs/basis/syntax/reserved

Kiedy nasz automat staje się coraz to bardziej skomplikowany, najlepiej jest podzielić go na mniejsze fragmenty. Rozwiązanie to ułatwia nam przede wszystkim implementację złożonego algorytmu, jak i sprawia, że utrzymanie kodu źródłowego staje się tańsze. Poprzez ***tańsze utrzymanie kodu źródłowego*** mam na myśli zdarzenia w przyszłości, gdy będziemy zmieniać kod oraz czytać go, a ***czas***, jaki na to poświęcimy, jest naszym kosztem. W momencie kiedy nasz kod staje się jedną wielką funkcją, każda taka zmiana wymaga od nas zrozumienia całej domeny problemu, a więc jesteśmy zmuszeni do zrozumienia np. 5000 linii kodu. Nietrudno sobie wyobrazić, że wymaga to ponownego „wgryzania się" w problem, co zabiera bardzo dużo czasu, zwłaszcza jeśli kod ten pisaliśmy pół roku temu. Jak wiadomo, programista bardzo często zapomina dokładną logikę już po dwóch tygodniach pracy. Dlatego dobrą praktyką jest pisanie kodu, który jest łatwy w czytaniu i zrozumieniu. Wtedy nanoszenie ewentualnych poprawek lub zmian okazuje się łatwe w implementacji, co osiągamy za pomocą funkcji. Ogólna struktura funkcji w kodzie źródłowym wygląda następująco:

Przykład:

```
1: typ_zwracany_funkcji nazwaFunkcji(typ parametryFunkcji)
2: {
3: polecenia do wykonania składające się na funkcję
4: }
```

Kod zawarty w linii numer 1 jest nazywany nagłówkiem funkcji, który składa się kolejno z typu, jaki funkcja zwraca za pomocą słowa kluczowego **return**[11], będącego operatorem. Przyjrzymy mu się dokładniej w jednym z najbliższych rozdziałów. Każdej funkcji musimy przypisać pewną nazwę, co widzimy już w linii pierwszej. Proponuję więc, żeby zarówno funkcje, jak i zmienne nazywać opisowo oraz w formacie CamelCase, tzn. jako „mojaPierwszaFunkcja", a nie „mojapierwszafunkcja". Myślę, że zgodzicie się ze mną, że dużo łatwiej czyta się pierwszą nazwę.

Wskazówka: Z nazwami zarówno funkcji, jak i zmiennych wiąże się jeszcze jeden problem, jakim jest język naturalny wykorzystywany podczas programowania. Naturalnie od nas zależy wybór języka, jednak trzeba dobrze go przemyśleć przed rozpoczęciem programowania, aby później zachować konsekwencję. Osobiście wykorzystuję i polecam angielski, a powodem mojego wyboru jest fakt, że jest to język, którym większość programistów biegle się posługuje. Okazuje się to bardzo przydatne wtedy, gdy chcę komuś udostępnić swój kod lub jego część, bo nie muszę go przepisywać, co musiałbym zrobić, gdybym posługiwał się w pracy językiem polskim. Ze względu na to, że końcowy odbiorca tej książki mówi po polsku, zdecydowałem, że na potrzeby opracowania lepiej będzie wykorzystać język rodzimy.

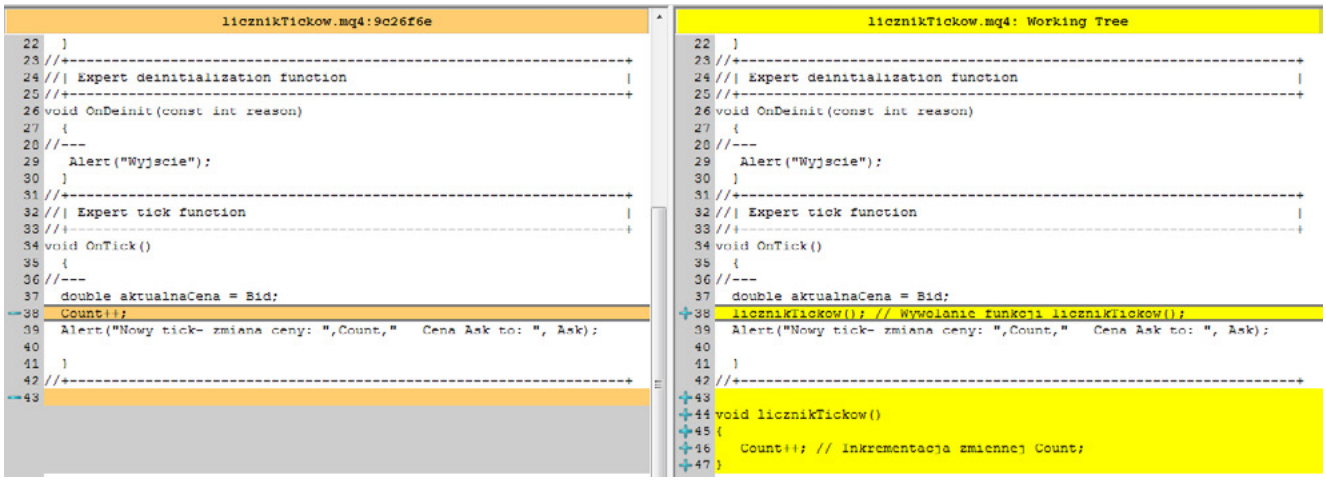
Po nazwie funkcji w nawiasach znajdują się parametry. W zależności od potrzeb funkcja może nie przyjmować w ogóle żadnego parametru, albo też go przyjąć, przy czym można wyróżnić aż 64 różne parametry. Cały kod, który mieści się pomiędzy nawiasami klamrowymi, stanowi tzn. ciało funkcji. Są tam zawarte wszystkie operacje, jakie chcemy wykonywać na przesyłanych do funkcji parametrach. Poniżej znajduje się dłuższy praktyczny przykład[15] :

11 Poprawna wymowa to 'rytern', ale zadziwiająco mało ludzi wymawia w sposób poprawny ten wyraz.

```
//+------------------------------------------------------------------+
//|                                          licznikTickow.mq4 |
//|                                          Tomasz Waszczyk |
//|                                          www.blog.waszczyk.com |
//+------------------------------------------------------------------+
#property copyright „Tomasz Waszczyk”
#property link      „www.blog.waszczyk.com”
#property version   „1.00”
#property strict
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
intCount = 0;

intOnInit()
 {
//---
  Alert („Funkcja init() wywolana na poczatku”);    // Alert
//return(0);                                 // Wyjscie z init()
//---
  return(INIT_SUCCEEDED);
 }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(constint reason)
{
//---
  Alert(„Wyjscie”);
 }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
 {
//---
 double aktualnaCena = Bid;
 Count++;
 Alert(„Nowy tick- zmianaceny: „,Count,”   Cena Ask to: „, Ask);
}
//+------------------------------------------------------------------+
```

Następnie w funkcji OnTick() znajdziemy linię, w której znajduje się kod odpowiedzialny za inkrementację[12], czyli dodanie do bieżącej wartości jedynki. Postanowiłem część kodu odpowiedzialną za operację inkrementacji zmiennej Count przenieść do własnej funkcji.

Lewa strona ekranu to wersja przed zmianami, prawa to postać kodu po zmianach:



Wskazówka: Operacja dodania jedynki do zmiennej nazywana jest **inkrementacją**. Operacja polegająca na usunięciu od zmiennej jedynki to z kolei **dekrementacja**.

W prawej części ekranu widzimy, że linia 38. jest zastąpiona wywołaniem[13] naszej funkcji licznikTickow():

```
void licznikTickow()
{
Count++; // Inkrementacja zmiennej Count;
}
```

Jak widzimy, w miejscu, w którym, jak wcześniej ustaliliśmy, znajduje się typ, jaki zwraca funkcja, widnieje teraz void[14]. Oznacza to, że funkcja oznaczona tym typem nie zwraca żadnej wartości, może jedynie zmieniać jakąś zmienną (w naszym przypadku globalną). W przypadku gdy spróbujemy jednak wykorzystać operator return w naszej funkcji licznikTickow:

```
void licznikTickow()
{
Count++; // Inkrementacja zmiennej Count;
  return count; - ten kod powoduje błąd na poziomie kompilacji.
}
```

12 http://docs.mql4.com/basis/operations/mathoperation
13 https://www.mql5.com/en/docs/basis/function/call
14 https://www.mql5.com/en/docs/basis/types/void

Kod ten się nie skompiluje a kompilator wypisze nam następujące błędy:

Opis	Plik	Li...	Ko...
'licznikTickow.mq4'			
'count' - undeclared identifier	licznikTickow.mq4	48	11
'return' - 'void' function returns a value	licznikTickow.mq4	48	4
2 error(s), 0 warning(s)			

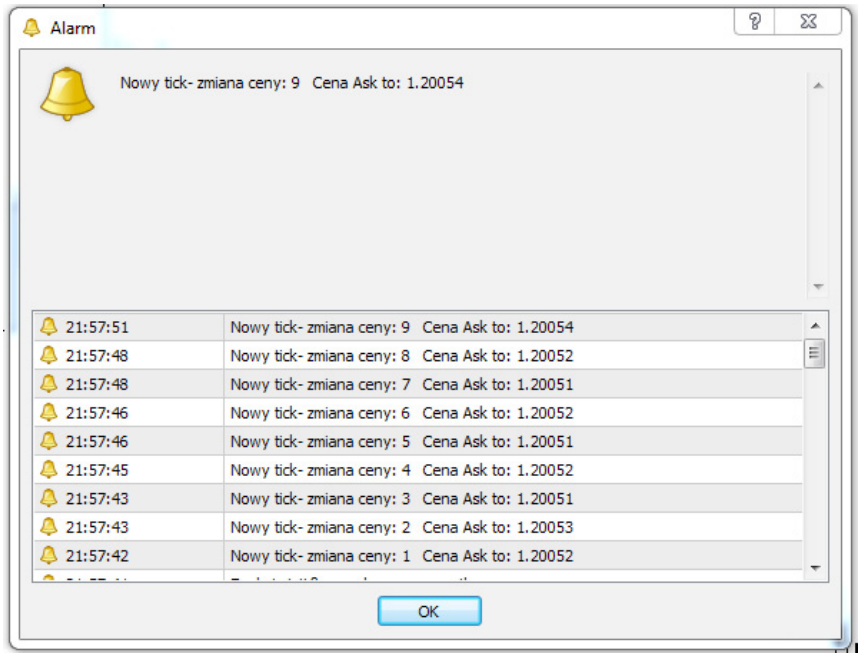
WSKAZÓWKA

Warto zwrócić uwagę na jeszcze jedną rzecz – jak widzimy, w kodzie powyżej jest napisane:

```
return count;
```

W MQL4/MQL5 zmienna count oraz Count to dwie różne zmienne, a kompilator MQL bierze pod uwagę wielkość wykorzystanych w zmiennej liter. Podczas programowania musimy o tym pamiętać.
W wyświetlanych w programie błędach możemy przeczytać: „return' - ‚void' functionreturns a value", co jest potwierdzeniem tego, że funkcja oznaczona jako „void" nie może zwracać żadnej zmiennej.

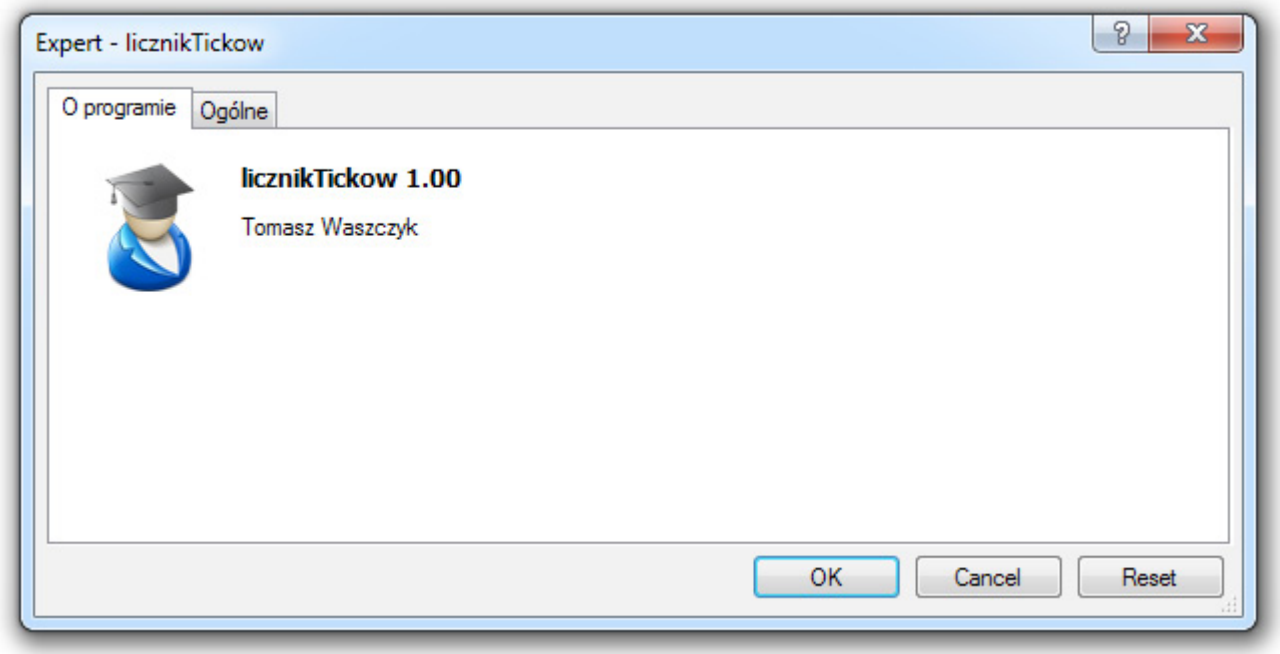
Efekt kompilacji oraz uruchomienia powyższego EA prezentuje się następująco:



Po zapoznaniu Czytelnika z całym kodem źródłowym, a następnie efektem prac programistycznych, chciałbym wyjaśnić krok po kroku, dlaczego za każdym razem, kiedy pojawi się na wykresie waloru nowa cena, widzimy okno Alarm. Jak już wspomniałem, wszystkie linie, które rozpoczynają się od znaku „//", stanowią komentarze, niebrane w ogóle pod uwagę przez kompilator i traktowane jako informacja dla czytających kod. Stąd też je pominę. Następnie mamy taki oto kod:

```
#property copyright „Tomasz Waszczyk"
#property link      „www.blog.waszczyk.com"
#property version   „1.00"
#propertystrict
```

Linie te odpowiadają za okno, które pojawia się przy inicjalizacji strategii na wykresie:



Jest to pewnego rodzaju kosmetyka, która daje nam możliwość wprowadzenia za pomocą dyrektywy „property" wielu ustawień dla danej strategii, takich jak: autor, wersja czy prawa autorskie. Możemy też ustawić naszą ulubioną ikonę jako domyślną – zainteresowanych tematem odsyłam do dokumentacji[15]. Zmienną globalną, jak wcześniej powiedzieliśmy, inicjujemy i deklarujemy na zewnątrz wszystkich funkcji: intCount = 0;

15 http://docs.mql4.com/basis/preprosessor/compilation

Aby zainicjować strategię, musimy wysłać odpowiedni komunikat do platformy, potwierdzający, że inicjacja przebiegła bezbłędnie. Robimy to za pomocą operatora return:

```
intOnInit()
 {
//---
  Alert („Funkcja init() wywolana na poczatku”);    // Alert

//---
  return(INIT_SUCCEEDED);                    // Wyjscie z init()
}
```

Następnie pozostaje nam określić, co ma się wydarzyć w momencie, kiedy przychodzi do platformy informacja o nowej cenie. Jest to bardzo ważny element naszego EA, ponieważ w funkcji tej definiujemy całą logikę. Aby oprogramować wydarzenie przyjścia nowego ticku do platformy, musimy skorzystać z funkcji obsługujących wydarzenia platformy (Event Handling Functions[16]). W tym przypadku wykorzystujemy funkcję OnTick():

```
void OnTick()
{
//---
 double aktualnaCena = Bid;
licznikTickow(); // Wywolanie funkcji licznikTickow();
 Alert(„Nowy tick- zmiana ceny: „,Count,”   Cena Ask to: „, Ask);

}
```

W powyższej funkcji najpierw tworzymy lokalną zmienną o nazwie „aktualnaCena” o typie „double”:

```
doubleaktualnaCena = Bid;
```

Uważny czytelnik zwróci uwagę na pewną niejasność, wynikająca z tego, że jakkokolwiek zdefiniowanie zmiennej lokalnej wydaje się zrozumiałe, to już nie wiadomo, skąd pobierana jest wartość tej zmiennej – przecież nigdzie nie ma zdefiniowanej zmiennej o nazwie „Bid”. Chciałbym w tym miejscu pogratulować tym, którzy zauważyli ten problem. Wyjaśniam, że w MQL mamy w każdym miejscu kodu dostęp do tzw. zmiennych predefiniowanych , obejmujących między innymi wartość, dzięki której otrzymujemy cenę, po której otwierane są zlecenia sprzedaży oraz zamykane zlecenia kupna, czyli ceny Bid. Jest to bardzo przydatna cecha języka MQL, z której będziemy bardzo często korzystać.

16 http://docs.mql4.com/basis/function/events

Następna linia zawiera nazwę naszej funkcji oraz nawias otwarcia oraz zamknięcia „()". Oznacza to, że funkcja ta nie przyjmuje żadnych parametrów:

```
licznikTickow();
```

Jest to informacja dla kompilatora, aby na liście wszystkich funkcji w kodzie źródłowym znalazł właśnie tą oraz wywołał kod w niej zawarty:

```
Count++;
```

Następuje więc dodanie jedynki do zmiennej Count – licznika tickόw. Kiedy kompilator widzi, że wykonał już całe działo funkcji, wraca do miejsca, skąd miało miejsce odwołanie do funkcji. Następnie dalej wykonuje kod. W naszym przypadku będzie to:

```
Alert(„Nowy tick- zmianaceny: „,Count," Cena Ask to: „, Ask);
```

W linii tej wywołujemy funkcję MQL'a Alert[17], która odpowiada za wygenerowanie oraz pokazanie okna alertu. Z względu na to, że operację tą wykonujemy podczas każdego nowego ticku, możemy powiedzieć, że program działa na zasadzie swego rodzaju pętli.

Na koniec omówimy ostatnią funkcję, jaką jest Deinit():

```
voidOnDeinit(constintreason)
 {
//---
  Alert(„Wyjscie");
 }
```

Funkcja OnDeinit stanowi przeciwieństwo funkcji OnInit. Jest wywoływana podczas deinizjalizacji strategii w trzech przypadkach:

1. Przed ponowną inicjalizacją z powodu zmiany waloru na wykresie lub interwału, do którego dołączona był strategia.
2. Przed ponowną inicjalizacją z powodu zmiany parametrów strategii.
3. Przed „odłączeniem" strategii z wykresu danego waloru.

Funkcja ta musi być zadeklarowana typem „void" i powinna mieć jeden parametr typu „const int", którego zadaniem jest oznaczenie przyczyny deinicjalizacji[18]. W praktyce wystarczy, jeśli jako parametr przekażemy stałą typu integer.

17 http://docs.mql4.com/common/alert

18 http://docs.mql4.com/constants/namedconstants/uninit

WSKAZÓWKA

Kiedy w kodzie źródłowym widzimy przed typem parametr „const”, oznacza to, że mamy do czynienia ze zmienną, która jest stałą[19] [20]. Ma to swoje istotne konsekwencje, gdyż w czasie wykonywania strategii (ang. runtime) zmiennej tej nie można przypisać innej wartości, niż ta, którą określiliśmy podczas deklaracji.

Jak się zapewne domyślacie, ciało funkcji, analogicznie do poprzedniego kodu, powoduje pojawienie się okienka Alarm podczas deinicjalizacji strategii. W ten sposób trader powiadamia nas, że właśnie usuwa strategię z wykresu. Jeśli nie otrzymamy żadnego potwierdzenia deinicjalizacji strategii, nie mamy całkowitej pewności, że operacja przebiegła pomyślnie.

To już koniec tego rozdziału. Zapoznaliśmy się z przykładową strukturą Expert Advisora napisanego w MQL, poznaliśmy wiele funkcji, dowiedzieliśmy się, co to są stałe i jak wygląda proces kompilacji. Wiemy też, gdzie znajdziemy informacje o błędach podczas kompilacji. W kolejnej części omówimy bardziej zaawansowane zagadnienia, a także zaimplementujemy w pełni działającą strategię, która będzie zawierać konkretne transakcje. Na końcu zaś obejrzymy wyniki testów, które przeprowadzimy za pomocą wbudowanego testera strategii w programie Meta Trader.

Operatory logiczne

Operatory logiczne służą do budowania wyrażeń logicznych w celu ich dalszego sprawdzenia i w zależności od ich stanu wykonania. Jednym z najczęstszych miejsc w kodzie, w których okazują się one bardzo użyteczne, są pętle. W każdej pętli możemy napisać własne warunki i połączyć je logicznie w następujący sposób:

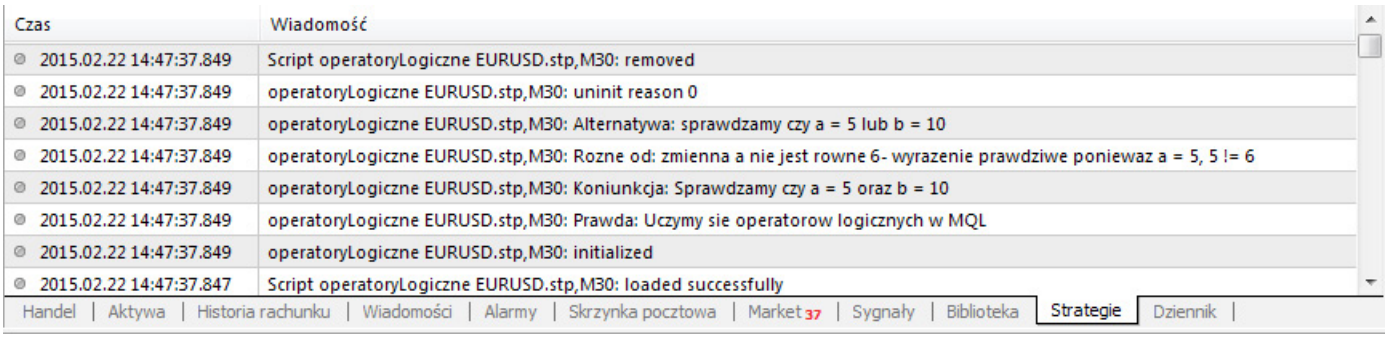
- alternatywa,
- koniunkcja,
- zaprzeczenie.

19 http://docs.mql4.com/basis/variables
20 http://book.mql4.com/basics/vars

W celu przeprowadzenia demonstracji działania na żywo wszystkich operatorów utworzyłem skrypt o nazwie operatoryLogiczne.mql , który pokazuje możliwości operatorów logicznych:

```
if(true){
   Print(„Prawda: Uczymy sie operatorow logicznych w MQL”);
}
if(false){
   Print(„Falsz: Tego tekstu nie zobaczymy w konsoli Meta Trader”);
}
```

W skrypcie tym widzimy pętlę „if”, dla której pierwszy z warunków został spełniony (pętla „if” jest wykonywana, gdy jej wartość logiczna jest prawdą). Aby zweryfikować i naocznie sprawdzić, uruchamiamy załączony skrypt i przeglądamy zakładkę „Strategie”:



W zakładce tej, czytając od dołu do góry, możemy śledzić poszczególne zdarzenia: najpierw skrypt został załadowany do pamięci, następnie zainicjowany, po czym wykonany kod źródłowy, zgodnie z opisem. W miejscu, gdzie przekazaliśmy pętli „if” wartość „false”, kompilator ominął ciało pętli.

Następnie tworzymy sobie dwie zmienne w celu demonstracji:

```
int a = 5;
int b = 10;
```

Operator „&&” oznacza koniunkcję. Aby kompilator wykonał funkcję „Print”, w ciele pętli oba warunki muszą zostać spełnione:

```
if(a == 5 && b == 10){
  Print(„Koniunkcja: Sprawdzamy czy a = 5 oraz b = 10”);
}
```

Aby udowodnić, że koniunkcja zachodzi tylko w przypadku, gdy oba warunki są spełnione, piszemy następujący warunek:

```
if(a == 6 && b == 10){
  Print(„Koniunkcja: Sprawdzamy czy a = 5 oraz b = 10”);
}
```

Naturalnie, warunek ten nie zostanie spełniony, przez co nie zobaczymy tego tekstu w zakładce „Strategie” (zobaczymy go tylko jeden raz).

Kolejnym operatorem logicznym jest operator „!=”, który oznacza „różny od” lub „nie równy”. Nasza zmienna a jest równa 5, czyli różna od 6, w związku z czym poniższe wyrażenie okazuje się prawdziwe:

```
if(a != 6){
  Print(„Rozne od: zmienna a nie jest rowne 6- wyrazenie prawdziwe poniewaz a = 5, 5 != 6”);
}
```

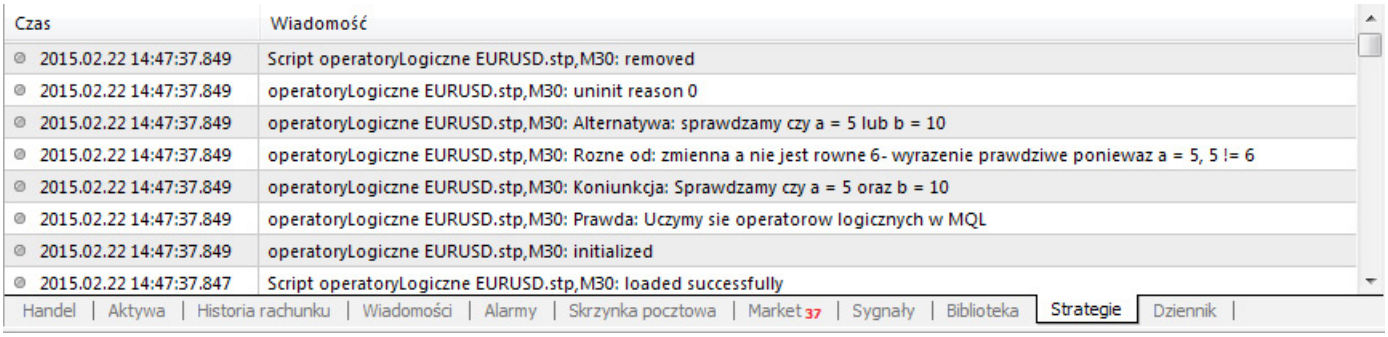
Ostatnim operatorem logicznym, z jakim się spotkamy, jest alternatywa, oznaczona w następujący sposób „||”. W tym przypadku wystarczy, że tylko jeden warunek będzie spełniony, aby tym samym całe wyrażenie zostało uznane za prawdziwe. W naszym przykładzie pierwszym warunkiem staje się sprawdzenie, czy zmienna a jest równa 6 (nieprawda) oraz czy b jest równe 10 (prawda), w wyniku czego całe wyrażenie staje się prawdziwe:

```
  // alternatywa czyli aby petla sie wykonala jedno z wyrazen musi byc spelnione
  if(a == 6 || b == 10){
    Print(„Alternatywa: sprawdzamy czy a = 5 lub b = 10 „);
  }
}
```

Jak zawsze zachęcam do własnych eksperymentów. Im więcej pracy z kodem, tym szybciej staniemy się biegli w programowaniu.

Operatory arytmetyczne

Każdy program operuje na liczbach, wykonując wszelkiego rodzaju obliczenia na zmiennych, które reprezentują liczby. W celu przedstawienia operatorów arytmetycznych, jakie oferuje nam MQL, stworzyłem prosty skrypt o nazwie „operatoryArytmetyczne.mq4". Kiedy uruchomimy go na wykresie, zaobserwujemy taki oto efekt:




```
int a = 2;
int b = 10;
```

Ponieważ kod źródłowy jest bardzo intuicyjny i nie wymaga objaśnień, przejdę do omówienia operacji arytmetycznych wykonywanych na dwóch zmiennych a oraz b. Zaczynam od operatora sumy:

```
//suma
int suma = a + b;
Print(„Suma „ + suma);
```

Następnie wykonuję operację odejmowania:

```
//roznica
int roznica = a - b;
Print(„Roznica „ + roznica);
```

Spodziewam się, że wynikiem odejmowania zmiennej a od b będzie -8, co mogę zweryfikować sprawdzając wyniki w zakładce. Pozostałe operatory są równie intuicyjne:

```
//iloczyn
int iloczyn = a * b;
Print(„Iloczyn „ + iloczyn);
```

```
//iloraz
int iloraz = b / a;
Print(„Iloraz „ +iloraz);


//zmiana znaku
int zmianaZnaku = -a;
Print(„Zmiana znaku „ + zmianaZnaku);
```

Ostatnim operatorem jest operator modulo, który służy do wyznaczenia reszty z dzielenia:

```
//reszta z dzielenia - modulo
int resztaZDzielenia = b % a;
Print(„Reszta z dzielenia „ + resztaZDzielenia);
```

Kiedy podzielimy liczbę 10 przez dwa, otrzymamy 5 oraz 0 reszty. Aby upewnić się czy jego działanie jest w pełni dla nas zrozumiałe, podam inny przykład:
10 mod 3 = 1 ponieważ 10 podzielone na 3 daje 3 oraz reszty 1.

Wskazówka: Kiedy korzystamy z operatorów warto używać nawiasów, aby mieć pewność, że kolejność wykonania operacji będzie taka, jaką chcemy otrzymać.

Typy programów w Meta Editor

Język MQL jest wykorzystywany nie tylko do handlu automatycznego. Dzięki jego rozbudowanej strukturze możemy tworzyć narzędzia, które będą ułatwiać nam codzienny handel. W tym podrozdziale opiszę różne typy narzędzi, które możemy programować w MQL. Z uwagi na funkcjonalności poszczególnych programów, możemy podzielić je na trzy kategorie:

1. **Automatyczna strategia inwestycyjna (Expert Advisor)** – najbardziej złożone programy, które mogą obejmować cały proces handlu na dostępnych instrumentach, począwszy od analizy wykresu, poprzez wykrycie sygnału ewentualnego wysłania maila lub innego typu powiadomienia oraz otwarcia transakcji wraz z prowadzeniem pozycji, a kończąc na zamknięciu pozycji.
2. **Wskaźnik (Custom Indicator)** – służy zazwyczaj do prezentacji na wykresie lub w osobnym oknie danego algorytmu w sposób graficzny. Za pomocą tego typu programów możemy na bieżąco wyliczać wszelkiego rodzaju średnie, bazując na najnowszych danych, ale również zaimplementować (lub zmienić istniejący) swój własny oscylator.

3. **Skrypt (Script)** – jest to najprostszy typ programu, którego tzw. cykl życia (life cycle) jest najkrótszy, ponieważ jego działanie polega na sekwencyjnym wykonaniu kilku instrukcji, po czym samoistnie wyłącza się. Zaletą tego typu programów jest to, że pozwalają na maksymalnie szybkie wykonanie paru operacji, które wykonywane manualnie zajęłyby dużo więcej czasu – a jak wiadomo, czas ten jest dla nas bardzo cenny.

Osobiście bardzo lubię skrypty i niemalże codziennie używam jednego z nich, który pozwala zamknąć wszystkie transakcje za pomocą jednego skrótu klawiszowego. Do każdego skryptu możemy bowiem „podpiąć" skrót klawiszowy, dzięki czemu jego uruchomienie trwa ledwie chwilę i nie musimy wyszukiwać danego skryptu w oknie „Nawigator". Drugim moim ulubionym skryptem jest zmiana interwału wszystkich wykresów. Zamiast klikania w poszczególne ikonki, wystarczy w moim przypadku użyć następujących skrótów klawiszowych, kolejno: CTRL + 1 dla interwału M1, CTRL + 2 dla M5 itd.

Bardziej doświadczony Czytelnik po zapoznaniu się z szeregiem możliwości, które daje nam MQL, zapyta: „Chwilka, a gdzie jest handel półautomatyczny?". Pojęcie handlu półautomatycznego oznacza program, który na bieżąco analizuje wykres, jednak nie zawiera żadnych transakcji, dzięki czemu możemy obserwować bardzo dużo wykresów jednocześnie, nie tracąc na to czasu (pod warunkiem, że oprogramowaliśmy odpowiedni algorytm), a decyzje inwestycyjne pozostawiamy swojemu doświadczeniu traderskiemu. Naturalnie jest możliwość pisania tego typu rozwiązań na platformę Meta Trader w postaci EA, jednak w strategii tej nie wykorzystujemy funkcji OrderSend(), lecz np. funkcję SendMail() , dzięki której otrzymujemy maila na swój smartfon z informacją, że cena spełniła właśnie wszystkie warunki zadane w warunkach wysłania alarmu, czyli potencjalnej transakcji.

Co zrobić, kiedy projekt nam się rozrasta?

Immanentną częścią programowania jest zmiana. Kod źródłowy ciągle ulega modyfikacjom, z uwagi na to, że pojawiają się w naszych głowach coraz to nowe pomysły. Pytanie brzmi, jak sobie radzić z tego typem środowiskiem? W tym podrozdziale chciałbym pokazać, w jaki sposób rozwiązać problemy, takie jak:

- zarządzanie wersjami automatów, które tworzymy,
- zachowanie historii zmian naszego oprogramowania,
- łatwy i szybki dostęp do zmian danej części kodu źródłowego,
- możliwość szybkiego porównania dwóch wersji systemu,
- tworzenie kopii zapasowej naszego kodu źródłowego.

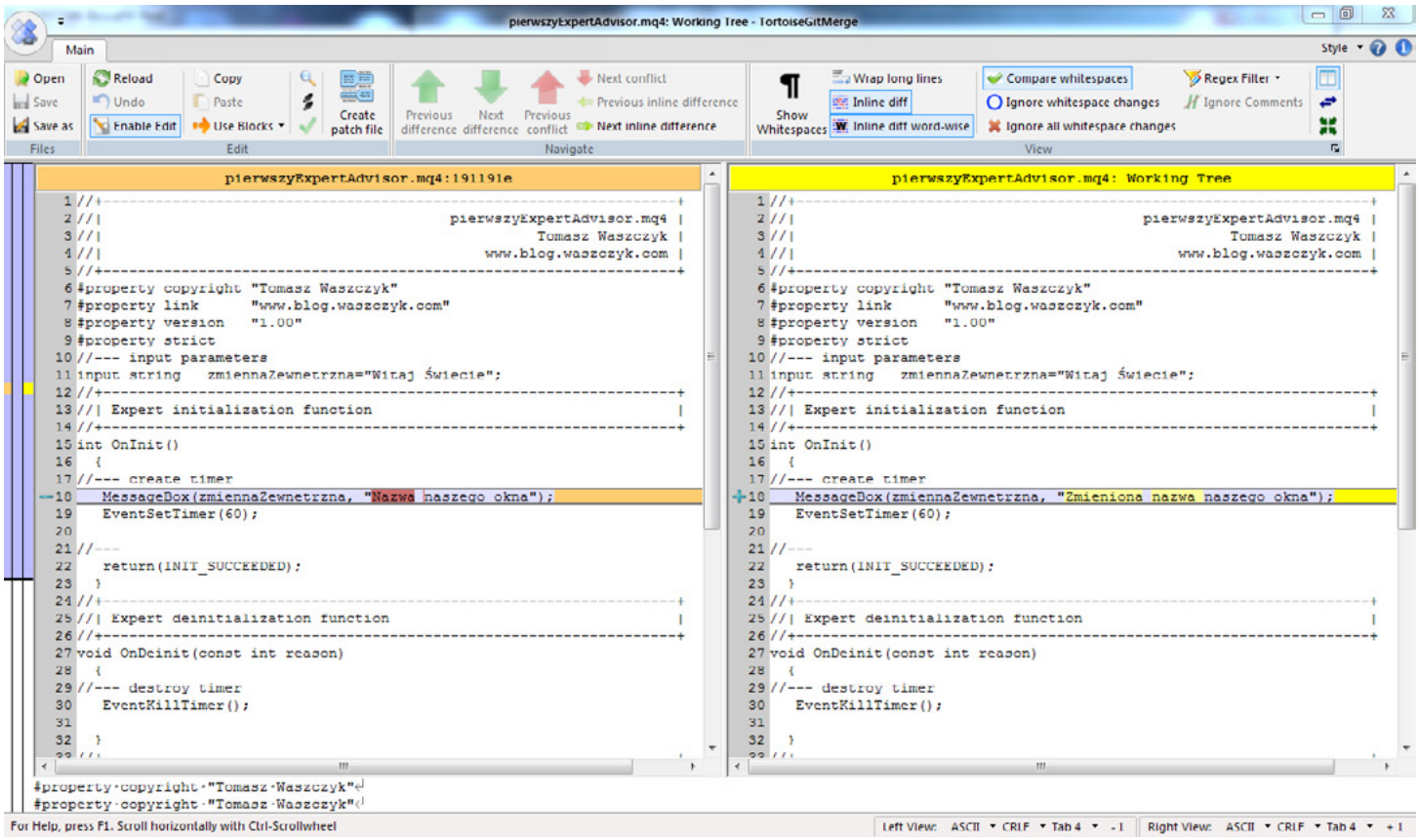
Ze względu na to, że domyślnie Meta Trader nie daje nam możliwości rozwiązania powyższych problemów, a dla osób, którzy programowaniem zajmują się zawodowo, są to niezbędne funkcjonalności, musiałem znaleźć na to sposób. Niestety, jeśli chcemy mieć dostęp do tych udogodnień, musimy doinstalować dodatkowe oprogramowanie na naszym komputerze.

Dalej rozwinę ten temat, zakładając, że każdy z Czytelników ma zainstalowane następujące oprogramowanie:

- **Klient Git:** http://git-scm.com/
- **TortoiseGit:** https://code.google.com/p/tortoisegit/wiki/Download

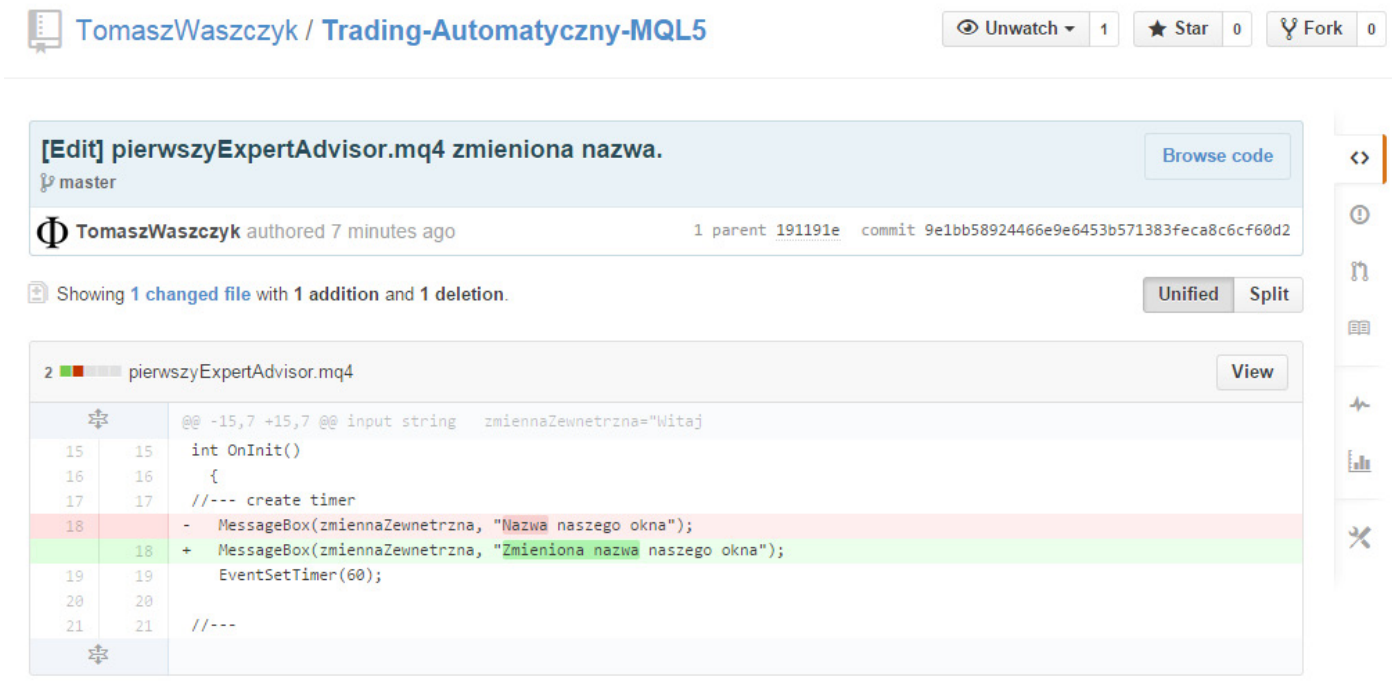
Dodatkowo każdy z Czytelników powinien posiadać darmowe konto w serwisie BitBucket: https://bitbucket.org lub GitHub- *https://github.com/*[21] oraz znać zagadnienia związane z systemami kontroli wersji (materiały na ten temat powszechnie dostępne są w sieci Internet). Zakładam też, że każda osoba, która biegle posługuje się komputerem, będzie w stanie dokonać pełnej instalacji oraz konfiguracji wyżej wymienionych narzędzi.

Poniżej chciałbym zaprezentować nowe możliwości po skonfigurowaniu systemu wersjonowania kodów źródłowych:

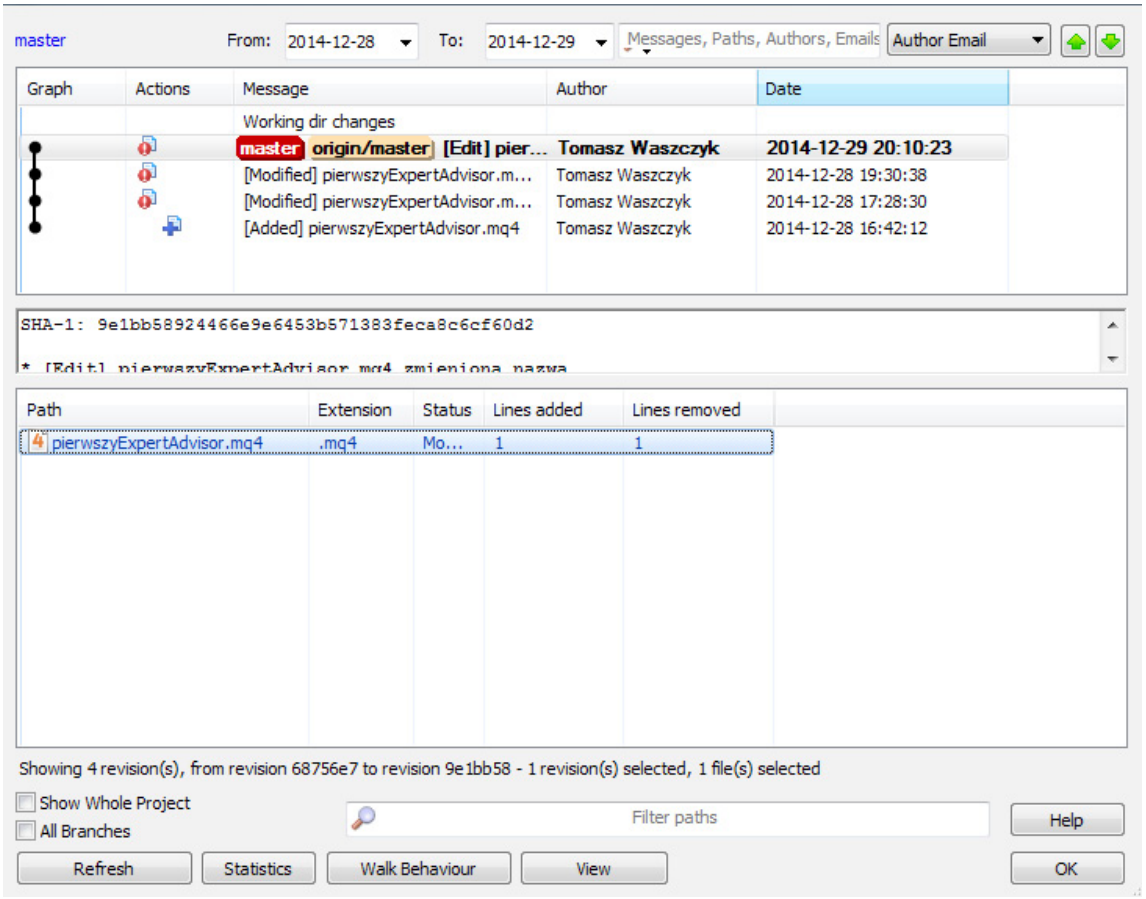


Jak widzimy na zamieszczonym zrzucie ekranu (przykład pochodzi z naszego pierwszego programu), na interfejsie graficznym można dostrzec zmiany, jakie zostały wprowadzone ostatnio w kodzie. Po lewej stronie widzimy wersję starszą, a przy linii numer 18 znak minusa, który znaczy, że jakiś fragment kodu został usunięty. Natomiast w tym samym miejscu po prawej stronie widzimy znak plusa, oznaczający, że w tej linii nastąpiły zmiany. Powyższy zrzut ekranu prezentuje TortoiseGit. Poniżej możemy zobaczyć to samo, jednak w postaci webowego interfejsu serwisu GitHub.

21 https://github.com/TomaszWaszczyk/Trading-Automatyczny-MQL



Natomiast na poniższym obrazie możemy zobaczyć równie przydatny element systemu kontroli wersji, jakim jest drzewo rozwoju oprogramowania, które możemy podzielić na tzw. branche[22] [23].



22 http://git-scm.com/book/pl/v1/Ga%C5%82%C4%99zie-Gita-Zarz%C4%85dzanie-ga%C5%82%C4%99ziami

23 http://git-scm.com/book/pl/v1/Ga%C5%82%C4%99zie-Gita-Podstawy-rozga%C5%82%C4%99ziania-i-scalania

Jak widzimy, daje to bardzo łatwy i szybki dostęp do wszystkich zrobionych przez nas commitów[24], przy czym commit to zapisanie zmian w repozytorium. Klikając w każdy commit prawym przyciskiem myszy możemy bardzo szybko sprawdzić, jakie zmiany zawierał w stosunku do poprzedniego, jak i każdego innego, dowolnie przez nas wybranego.

WSKAZÓWKA

Proszę zwrócić uwagę, że opisane przeze mnie rozwiązanie ma wiele zalet, ale też wady – musimy być świadomi, że robiąc operację „Push" w systemie kontroli Git wersji, wysyłamy naszą pracę na obce serwery i **nie możemy być pewni** tego, co z efektami naszej pracy się wydarzy dalej. Nie chodzi tu o możliwość utraty naszej pracy z powodu wadliwego działania komputerów w serwisie BitBucket, a raczej o wątpliwości, czy aby na pewno chcemy wysłać gdzieś efekty naszej pracy, nie wiedząc, kto ma do nich dostęp. Rozwiązaniem tego problemu są prywatne repozytoria Git przechowywane na własnym serwerze lub własnym dysku[25].

24 https://www.atlassian.com/git/tutorials/saving-changes/git-commit/

25 https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config

Część zaawansowana

Czytając poprzednie rozdziały z pewnością czułeś niedosyt informacji, ponieważ we wcześniej zaprogramowanym EA ani nie handlowaliśmy, ani nawet nie przetestowaliśmy naszego pomysłu na handel. Nie zrobiliśmy zatem tak podstawowej rzeczy, jaką jest zaprogramowanie naszego pomysłu. W tym rozdziale napiszemy wspólnie EA, które będzie handlować. Pokażę i pomogę rozwiązać problemy, z jakimi trader spotyka się podczas handlowania za pomocą strategii automatycznych, takich jak handel, zarówno manualnie, jak i z użyciem EA. Zrobimy to w taki sposób, aby nasz automat nie „widział" transakcji zrobionych manualnie. Dzięki temu nie dojdzie do np. przypadkowego zamknięcia lub otwarcia transakcji, którą otworzyliśmy uznając, że jest to dobry moment, a nasze EA nie miało zaprogramowanego tego wejścia. Na koniec przetestujemy nasz automat za pomocą wbudowanego testera strategii, jaki znajdziemy w każdym pakiecie oprogramowania Meta Trader. Ze względu na to, że nasz automat liczy ok. 200 linii kodu, aby zaoszczędzić czas każdego z Czytelników, cały kod źródłowy jest dostępny pod następującym linkiem:

https://github.com/TomaszWaszczyk/Trading-Automatyczny-MQL/blob/master/przeciecieSrednichEA.mq4

Ma to także skłonić Was do przetestowania i eksperymentów na własną rękę.

Implementacja Expert Advisor

Poniżej opiszę, funkcja po funkcji, linia po linii, bardziej skomplikowany automat, przy czym zachęcam Czytelnika do zapoznania się z innymi aspektami programowania tej strategii, eksperymentów i własnych ćwiczeń. Należy pamiętać, że jedynie za sprawą praktyki stajemy się coraz lepszymi programistami, co przekłada się z czasem na coraz lepsze wyczucie, jakie rozwiązanie jest lepsze, a jakie gorsze w danym przypadku. Dodam, że kod jest modyfikacją strategii standardowo dodawanej przy każdej instalacji EA.

Wskazówka: Przy instalacji platformy MT dostajemy dwie standardowe strategie: MACD Sample oraz MovingAverage. Uważam, że stanowią one bardzo dobry materiał do analiz. MovingAverage jest podobna do tej, którą przedstawiam poniżej, a MACD Sample, jak sama nazwa podpowiada, bazuje na wskaźniku MACD.

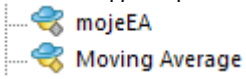
Zaczynajmy.

```
//+------------------------------------------------------------------+
//|                                    przeciecieSrednichEA.mq4 |
//|                                                  Tomasz Waszczyk |
//|                                             www.blog.waszczyk.com |
//+------------------------------------------------------------------+
#include<stderror.mqh>
#include<stdlib.mqh>
//---------------------------------------
```

Dyrektywa „include”[26] odpowiada za dostarczenie kodu źródłowego z zewnętrznego pliku o rozszerzeniu *.mqh. Gdybyśmy potrzebowali zaimportować kod z innego pliku binarnego, takiego jak *.ex4 lub biblioteki DLL (*.dll), powinniśmy wykorzystać tzw. funkcje importu[27].

```
#property copyright „Tomasz Waszczyk”
#property link      „www.blog.waszczyk.com”
#property version   „1.00”
#property strict
```

Myślę, że powyższe linie wydają się zrozumiałe. Chciałbym wspomnieć tylko o jednej zmianie MQL4. Czytelnicy, którzy korzystają od jakiegoś czasu z platformy MT, zapewne będą kojarzyć dobrze tę historię. W pewnym momencie firma MetaQuotes postanowiła zrobić aktualizację dla języka MQL4, w następstwie czego zapewne wielu z Was widziało na własnej platformie tę zmianę, poprzez brak „podświetlenia”



nego (lub kupionego) oprogramowania napisanego w MQL, co wyglądało tak:

W praktyce zmiana ta miała spore konsekwencje dla użytkownika, ponieważ znaczną część kodu źródłowego należało napisać od nowa. Więcej o tym możecie przeczytać w Internecie[28].

```
//--- oznaczenie „magiccode” po ktorymmozemyrozrozniac transakcje naszego EA;
#define MAGIC_CODE 20150104
//---------------------------------------
```

Często podczas rozmowy z osobą, która potrzebuje strategii automatycznej, prosi mnie ona o gwarancję, że napisany przeze mnie kod będzie działać na wykresie danego waloru i nie będzie interferować z innymi transakcjami (manualnymi). Podam taki przykład. Klient prosi mnie o implementację własnego algorytmu do prowadzenia stop loss dla transakcji długoterminowych, ale jednocześnie chce handlować podczas sesji na tym samym instrumencie. Dzięki „MAGIC_CODE” możemy rozróżnić, która pozycja jest pod „restrykcją”

26 http://docs.mql4.com/basis/preprosessor/include
27 http://docs.mql4.com/basis/preprosessor/import
28 http://docs.mql4.com/mql4changes#compiler_difference

naszego EA, a która powinna zostać pominięta przy automatycznej analizie zleceń.

```
extern int stopLoss                = 500;
extern int takeProfit              = 500;
extern double loty                 = 0.1;
extern double maksymalneRyzyko     = 0.02;
extern double wspolczynnikPowiekszenia  = 3;
extern int okresSrednichCenyOtwarcia  = 12;
extern int okresSrednichCenyZamkniecia  = 21;
extern int przesuniecieSredniej    = 1;
extern color   kolorKupna          = clrAliceBlue;
extern color   kolorSprzedazy      = clrSandyBrown;
//-------------------------------------
double SL = 0, TP = 0;
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
```

Powyżej widzimy deklarację oraz inicjalizację szeregu zmiennych, przy czym słowo kluczowe „extern" przed deklaracją oznacza, że zmienna ta pojawi się jako możliwa do zmiany przy inicjalizacji strategii przed uruchomieniem na wykresie.

```
//--- Funkcja startowa naszego EA;
void start()
{
//--- Jesli nie sredniaKroczacadostepnych 100 swieczek nie sredniaKroczacamozliwosci handlu;
if(Bars <100 || IsTradeAllowed() == false)
return; // Wyjscie z EA, brakhandlu;
//--- Jesli obliczony wolumen trnsakcji jest zgodny z aktualnym depozytem rachunku
if(policzAktualneTransakcje(Symbol())==0)
sprawdzOtwarcieTransakcji();  // EA zaczyna sprawdzac czy jego warunki zawierania transakcji saspelnione;
else
sprawdzZamkniecieTransakcji();  // W przeciwnym razie zamknij transakcje;
}
```

Funkcja „start()" w MQL4 to tzw. funkcja specjalna[29], stanowiąca odpowiednik funkcji „OnTick()" w MQL5, która jest wywoływana zaraz po pojawieniu się nowego ticku na wykresie. Funkcja ta jest punktem startowym naszej strategii, w której, za pomocą pętli „if"[30] oraz zawartego w niej operatora logicznego alternatywy „||"[31] , sprawdzamy czy możemy handlować. Na podstawie definicji alternatywy możemy wywnioskować, że jeśli jeden z warunków jest niespełniony, wychodzi z EA (odpowiada za to „return;" co jest zakomentowane).

29 http://book.mql4.com/programm/special
30 http://book.mql4.com/operators/if
31 http://book.mql4.com/basics/expressions#Log

Poniżej widzimy funkcję, której zadaniem jest obliczenie, ile otwartych transakcji kupna oraz sprzedaży posiadamy:

```
//+------------------------------------------------------------------+
//| Ustalenie otwartych transakcji;                                  |
//+------------------------------------------------------------------+
intpoliczAktualneTransakcje(string symbol)
 {
int kupno = 0, sprzedaz = 0;
for(int i=0; i<OrdersTotal(); i++)
{
if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES)==false)
break;
if(OrderSymbol()==Symbol() &&OrderMagicNumber()==MAGIC_CODE)
      {
if(OrderType()==OP_BUY) kupno++;
if(OrderType()==OP_SELL) sprzedaz++;
}
   }
//---- Zwrocenie wolumenu transakcji- (ilosci);
if(kupno >0)
return(kupno);
else
return(-sprzedaz);
}
```

Widzimy, że funkcja ta przyjmuje jeden parametr typu string (ciąg znaków), co będzie nam potrzebne, aby ustalić, jakim walorem handlujemy w funkcji „egzekucjaZlecen". W ciele funkcji możemy z kolei sprawdzić, jakie transakcje obecnie mamy w portfelu. Robimy to za pomocą pętli „for", wewnątrz której mamy zadeklarowaną oraz zainicjalizowaną zmienną pomocniczą „i". Następnie dzięki funkcji wbudowanej „OrdersTotal()" możemy otrzymać informację o ilości zleceń złożonych, jak i oczekujących. Informacja ta jest nam potrzebna, ponieważ musimy wiedzieć, ile razy pętla „for" ma się wykonać, aby nie ominąć przy analizie żadnego zlecenia. Przy pomocy funkcji „OrderSelect()" możemy wskazać konkretne zlecenie w celu sprawdzenia, czy w atrybutach wybranego przez pętlę „for" zlecenia znajduje się nasz wcześniej zdefiniowany „magic code". Jeśli dane zlecenie nie zawiera naszego kodu, omijamy je poprzez wyjście z pętli „if". W przeciwnym razie, w zależności od typu zlecenia, inkrementujemy odpowiednie zmienne, a w celu sprawdzenia typu danego zlecenia korzystamy z funkcji „OrderType()" [40].

Mamy do dyspozycji 6 typów zlecenia:

1. **OP_BUY** – zlecenie kupna,
2. **OP_SELL** – zlecenie sprzedaży,
3. **OP_BUYLIMIT** – zlecenie oczekujące typu buy limit,
4. **OP_BUYSTOP** – zlecenie oczekujące typu buy stop,
5. **OP_SELLLIMIT** – zlecenie oczekujące typu sell limit,
6. **OP_SELLSTOP** – zlecenie oczekujące typu sell stop.

Na nasze potrzeby wystarczą dwa pierwsze typy. Jeśli dane zlecenie spełnia nasze warunki, inkrementujemy odpowiednie zmienne dokładnie w tych linijkach:

Na nasze potrzeby wystarczą dwa pierwsze typy. Jeśli dane zlecenie spełnia nasze warunki, inkrementujemy odpowiednie zmienne dokładnie w tych linijkach:

```
if(OrderType()==OP_BUY) kupno++;
if(OrderType()==OP_SELL) sprzedaz++;
```

Ze względu na to, iż nasza funkcja jest oznaczona jako ta, która zwraca typ int (i takie właśnie zadanie chcemy jej przypisać), na końcu za pomocą „return” zwracamy odpowiednie wartości. Zapewne każdy z traderów zgodzi się, że wolumen transakcji jest ściśle powiązany z naszym depozytem oraz aktualną (podczas handlu) ilością gotówki, dlatego dobrą praktyką jest implementacja funkcji, która na bieżąco jest w stanie wyliczyć wolumen ewentualnej transakcji:

```
//+------------------------------------------------------------------+
//| Wyliczenie optymalnej wielkosci transakcji                       |
//+------------------------------------------------------------------+
double optymalizacjaWielkosciLotu()
{
double lot = loty;
int zlecenia = HistoryTotal(); // Calahistoriatransakcji;
int zleceniaStratne = 0;           // Liczba transakcji stratnych;
//---- Wyliczenie wielkosci lotu;
  lot = NormalizeDouble(AccountFreeMargin() * maksymalneRyzyko / 1000.0,1);
//---- Obliczenie liczby transakcji stratnych
if(wspolczynnikPowiekszenia>0)
    {
for(int i=zlecenia-1;i>=0;i--)
{
if(OrderSelect(i,SELECT_BY_POS,MODE_HISTORY)==false)
        {
Print(„Error in history!”);
break;
```

```
        }
if(OrderSymbol()!=Symbol() || OrderType()>OP_SELL)
continue;
//----
if(OrderProfit()>0)
break;
if(OrderProfit()<0)
zleceniaStratne++;
      }
if(zleceniaStratne>1)
       lot=NormalizeDouble(lot-lot*zleceniaStratne/wspolczynnikPowiekszenia,1);
   }
//---- Zwroceniewielkosci lotu
if(lot <0.1)
    lot = 0.1;
return(lot);
 }
```

Powyższa funkcja zwraca typ „double”, dlatego na początku potrzebujemy zmiennej lokalnej, czyli „lot”. Następnie musimy pobrać liczbę transakcji, jakie zawarliśmy w historii, abyśmy mieli argument pętli „for”. Dzięki temu będziemy wiedzieć, ile razy pętla ma się wykonać. Następnie za pomocą omówionej już funkcji „OrderSelect” sprawdzamy, czy podczas pobierania historii nie wystąpił błąd. Zmienna, która ma służyć nam jako wolumen transakcji w MQL, musi być przed użyciem (wykorzystaniem funkcji „OrderSend”) znormalizowana, co robimy za pomocą funkcji NormalizeDouble , która po prostu zaokrągla liczbę zmiennoprzecinkową – nie możemy przecież złożyć zlecenia o wolumenie 1.22324243 lot. Przy zwracaniu wartości raz jeszcze zabezpieczamy się za pomocą pętli „if”. Gdy nasz obliczony wolumen jest mniejszy niż 0.1, zastępujemy go wcześniej obliczonym, najmniejszym dostępnym wolumenem (to moje założenie, które na platformach z możliwości handlu mikrolotami okazuje się błędne).

Zbliżamy się powoli do krytycznej części kodu źródłowego, jaką jest składanie zleceń oraz sprawdzanie warunków składania zleceń:

```
//+------------------------------------------------------------------+
//| Funkcji otwierajaca zlecenia ! !                                 |
//+------------------------------------------------------------------+
voidsprawdzOtwarcieTransakcji()
 {
doublesredniaKroczaca;
int    res;
//---- Handluj tylko przy pierwszym ticku nowej swieczki;
if(Volume[0]>1)

return;
```

```
//---- Pobranie sredniejkroczacej;
sredniaKroczaca = iMA(NULL, 0, okresSrednichCenyOtwarcia, przesuniecieSredniej, MODE_SMA,
PRICE_CLOSE, 0);
//---- warunki sprzedazy
if(Open[1] >sredniaKroczaca&& Close[1] <sredniaKroczaca)
{
if(stopLoss>0)
      SL = Bid + Point*stopLoss;
if(takeProfit>0)
       TP = Bid - Point*takeProfit;
   res = egzekucjaZlecen(Symbol(), OP_SELL, optymalizacjaWielkosciLotu(), Bid, 3, SL, TP,"Moving Average",
MAGIC_CODE, 0, kolorSprzedazy);
if(res <0)
      {
Print(„Wystapil blady przy otwieraniu zlecenia sprzedazy #",GetLastError());
Sleep(10000);
return;
      }
   }
//---- warunki kupna
if(Open[1] <sredniaKroczaca&& Close[1] >sredniaKroczaca)
{
    SL=0;TP=0;
if(stopLoss>0)
       SL = Ask-Point*stopLoss;
if(takeProfit>0)
       TP = Ask+Point*takeProfit;
   res = egzekucjaZlecen(Symbol(), OP_BUY, optymalizacjaWielkosciLotu(), Ask, 3, SL,TP, „Moving Average",
MAGIC_CODE, 0, kolorKupna);
if(res <0)
      {
Print(„Wystapil blady przy otwieraniu zlecenia kupna #",GetLastError());
Sleep(10000);
return;
      }
   }
//----
 }
```

Chcemy handlować jedynie na pierwszym ticku nowej świeczki, co możemy osiągnąć dzięki założeniu:

```
if(Volume[0]>1)
```

Pętlę „if" już poznaliśmy i wiemy, że jeśli warunek jest spełniony (true) wewnątrz nawiasów, pętla jest wykonywana. To, co może nas zdziwić w powyższym kodzie, to nawiasy kwadratowe przy zmiennej „Volume[0]" oraz fakt, że nigdzie wcześniej nie zadeklarowaliśmy tej zmiennej. Zmienna Volume to tzw. tablica[32], czyli struktura danych. Aby się do niej odwołać, musimy wewnątrz nawiasów kwadratowych wpisać indeks danych, do których chcemy uzyskać dostęp. Zmienna Volume przechowuje liczbę ticków z danej świeczki (niektórzy ją mylą z wolumenem transakcji). Aby dostać się do wolumenu „najświeższej" świeczki, potrzebujemy odwołać się do zmiennej o indeksie 0. Następnie, za pomocą funkcji wbudowanej iMA[33], będziemy obliczać na bieżąco wartość średniej kroczącej. Następnie definiujemy konkretnie warunki transakcji sprzedaży:
if(Open[1] >sredniaKroczaca&& Close[1] <sredniaKroczaca)

Powyższy warunek wykorzystuje, podobnie jak w przypadku Volume, zmienną predefiniowaną Open dzięki której mamy dostęp do cen otwarcia kolejnych świec, przy czym indeks 1 oznacza, że jest to przedostatnia świeca. Analogiczna sytuacja ma miejsce w przypadku zmiennej Close, która przechowuje cenę zamknięcia dla poszczególnych świec. Podwójny znak pomiędzy tymi dwoma warunkami oznacza koniunkcję, a zatem warunki pętli zostały spełnione i oba warunki muszą zwrócić wartość true. W następnych linijkach obliczamy odpowiedni stop loss oraz take profit, do czego wykorzystujemy predefiniowane zmienne Ask[34] oraz Point[35]. Dla ostatecznego potwierdzenia, że zlecenie zostało złożone bez jakichkolwiek błędów, wartość, jaką zwraca funkcja „egzekucjaZlecen", przypisujemy do zmiennej res. Jeśli wszystko poszło w porządku, funkcja OrderSend, w momencie, kiedy składa zlecenie, zwraca tzw. ticket zlecenia. W przeciwnym razie zwraca wartość -1, stąd pętla:

```
if(res <0)
    {
Print(„Wystapil blady przy otwieraniu zlecenia kupna #",GetLastError());
Sleep(10000);
return;
    }
```

Dzięki tej pętli mamy pewność, że zostaniemy poinformowani o ewentualnym błędzie. Wbudowana funkcja Sleep po prostu zatrzymuje EA na wskazaną w nawiasie ilość milisekund.

W przypadku warunków kupna kod jest analogiczny:

```
if(Open[1] <sredniaKroczaca&& Close[1] >sredniaKroczaca)
```

Jedyną zmianą w warunkach pętli są przeciwstawne wartości względem sprzedaży wartości zmiennych

32 http://docs.mql4.com/predefined/volume
33 http://docs.mql4.com/indicators/ima
34 http://docs.mql4.com/predefined/ask
35 http://docs.mql4.com/predefined/pointvar

Open oraz Close.

Bez zamknięcia transakcji nie ma zysku – tak można by opisać funkcję „sprawdzZamkniecieTransakcji". Jej zadaniem jest uchwycenie momentu zdefiniowania warunków, kiedy poszczególne transakcje mają zostać zamknięte. Podobnie jak w przypadku otwierania transakcji, wszystkie pozycje chcemy zamykać na pierwszym ticku nowo otwartej świeczki:

```
//+------------------------------------------------------------------+
//| Funkcja zamykajaca transakcje                                    |
//+------------------------------------------------------------------+
voidsprawdzZamkniecieTransakcji()
 {
doublesredniaKroczaca;
//---- Zacznij handlowac na pierwszym ticku nowej swieczki;
if(Volume[0] >1) return;
```

Analogicznie do poprzednich linii kodu obliczamy wartość średniej:

```
//---- Pobranie sredniejkroczacej;
sredniaKroczaca = iMA(NULL,0,okresSrednichCenyZamkniecia,przesuniecieSredniej,MODE_SMA,PRI-
CE_CLOSE,0);
//----
```

Poniższa pętla odpowiada bezpośrednio za zamykanie zleceń. Dzięki pętli „for" analizujemy wszystkie otwarte zlecenia, a następnie sprawdzamy, które z nich zawiera nasz wcześniej zdefiniowany „magic code" i na podstawie typu zlecenia zamykamy tylko te zlecenia, które zostały otworzone przez naszego robota, korzystając przy tym z wbudowanej funkcji OrderClose[36]:

```
for(inti=0; i<OrdersTotal(); i++)
   {
if(OrderSelect(i, SELECT_BY_POS, MODE_TRADES) == false) break;
if(OrderMagicNumber() != MAGIC_CODE || OrderSymbol()!=Symbol()) continue;
//---- sprawdz typ zlecenia
if(OrderType() == OP_BUY)
      {
if(Open[1] >sredniaKroczaca&& Close[1] <sredniaKroczaca)
OrderClose(OrderTicket(), OrderLots(), Bid, 3, kolorKupna);
break;

      }
```

36 http://docs.mql4.com/trading/orderclose

```
    if(OrderType() == OP_SELL)
    {
    if(Open[1] <sredniaKroczaca&& Close[1] >sredniaKroczaca)
    OrderClose(OrderTicket(), OrderLots(), Ask, 3, kolorSprzedazy);
    break;
          }
       }
}
```

WSKAZÓWKA

Jedną ze zmiennych funkcji OrderClose jest tzw. poślizg ceny (ang. slippage). Ten istotny parametr określa maksymalną różnicę faktycznego zamknięcia pozycji z trzecim z kolei parametrem, jakim jest konkretna cena. Wspominam o tym dlatego, ponieważ kiedy piszemy robota przygotowanego pod dane fundamentalne i kiedy cena może bardzo się zmieniać, warto powiększyć parametr slippage, aby mieć pewność, że zlecenie zamknięcia zrealizuje się (czyli zwiększyć prawdopodobieństwo).

W funkcji „sprawdzOtwarcieTransakcji" znalazło się wywołanie poniższej funkcji:

```
//+------------------------------------------------------------------+
//| Otwarcie zlecen po biezacych cenach                              |
//+------------------------------------------------------------------+
integzekucjaZlecen(string    symbol,
intcmd,
double    volume,
double    price,
int       slippage,
doublestoploss,
doubletakeprofit,
               string    comment,
int       magic,
datetime  expiration,
               color     arrow_color)
 {
int ticket = OrderSend(symbol,cmd,volume,price,slippage,0,0,comment,magic,expiration,arrow_color);
```

```
int check = -1;
if(ticket >0&& (stoploss != 0 || takeprofit != 0))
   {
if(!OrderModify(ticket,price,stoploss,takeprofit,expiration,arrow_color))
      {
       check = GetLastError();
if(check != ERR_NO_MQLERROR)
Print(„OrderModifyzglosilblad: „,ErrorDescription(check));
      }
   }
else
   {
    check=GetLastError();
if(check!=ERR_NO_ERROR)
Print(„OrderSendzglosilblad: „,ErrorDescription(check));
}
return(ticket);
 }
//+------------------------------------------------------------------+
```

Powyższy kod odpowiada za umieszczenie zlecenia na rynku. Dobrą praktyką jest wydzielać pewne funkcjonalności do osobnej funkcji, aby kod stał się bardziej przyjazny podczas czytania. W języku programistów mówimy nawet o czystym kodzie (clean code). Pętla „if" odpowiada za sprawdzenie (i ewentualną zmianę), czy zajęta pozycja posiada stop loss oraz take profit. W celu ewentualnej zmiany pozycji wykorzystujemy funkcję OrderModify[37], natomiast aby pobrać ostatni błąd, jaki wystąpił, korzystamy z funkcji[38].

Właśnie zakończyliśmy omawianie naszej prawdziwej strategii. Zachęcam gorąco do dalszych eksperymentów, pamiętając, że to dopiero początek przygody z programowaniem tego typu. Musimy też pamiętać o tym, że rynek nieustannie się zmienia i największym wyzwaniem jest dostosowanie naszych strategii do ciągłych zmian warunków, w których nasze oprogramowanie będzie funkcjonować.

Testowanie strategii automatycznych w MQL

Rozdział ten będzie poświęcony jest testerowi strategii, dostępnemu w każdej instalacji Meta Tradera, oraz jego praktycznemu zastosowaniu, czyli przetestowaniu wcześniej omówionej strategii. Należy być świadomym, że jest to krytyczny etap opracowywania strategii automatycznej, a często niestety bardzo zaniedbywany i bagatelizowany, zwłaszcza przez bardziej niecierpliwych traderów.

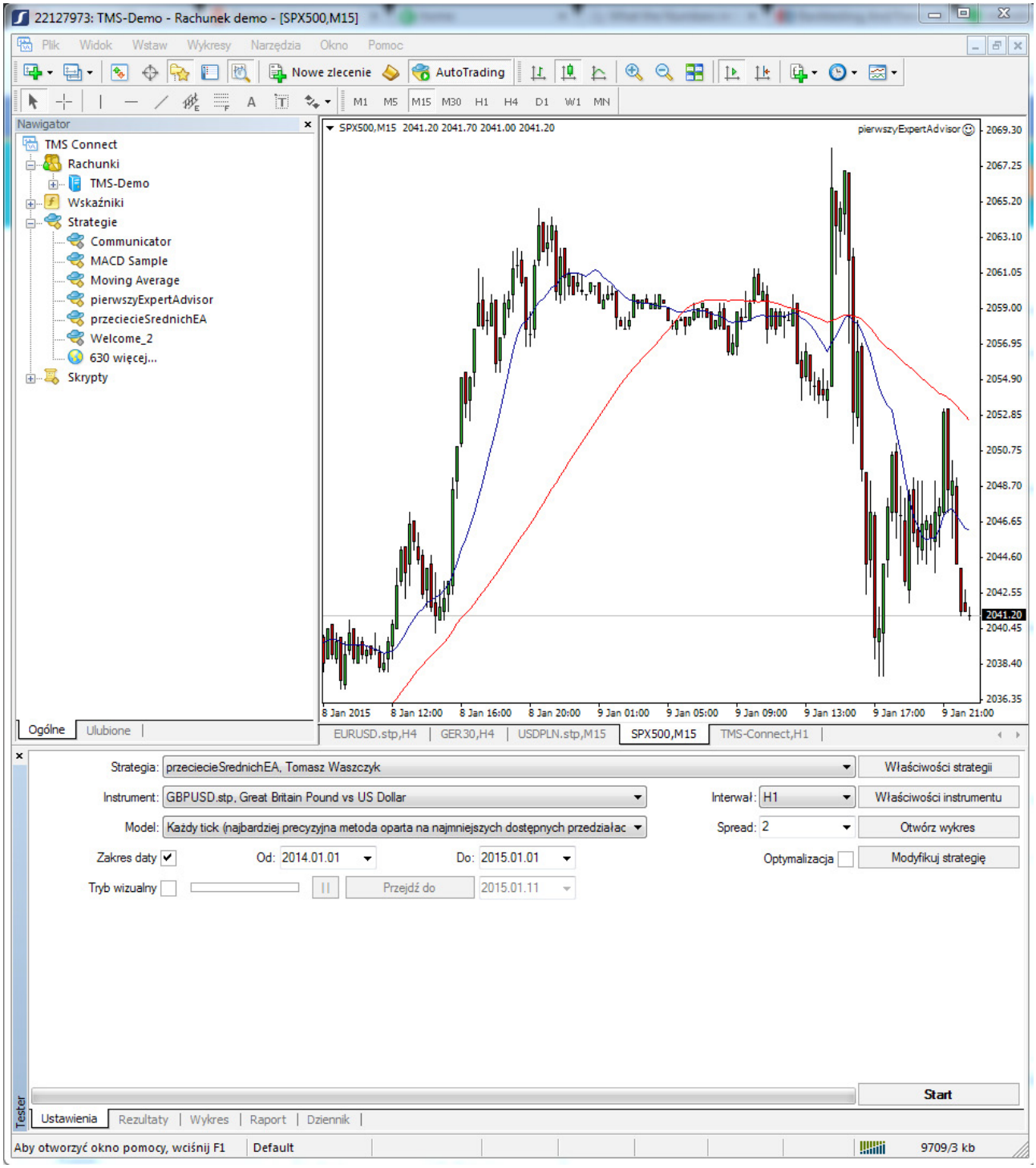
Tester standardowo wbudowany w naszą platformę dokonuje transakcji na podstawie danych historycznych

37 http://docs.mql4.com/trading/ordermodify

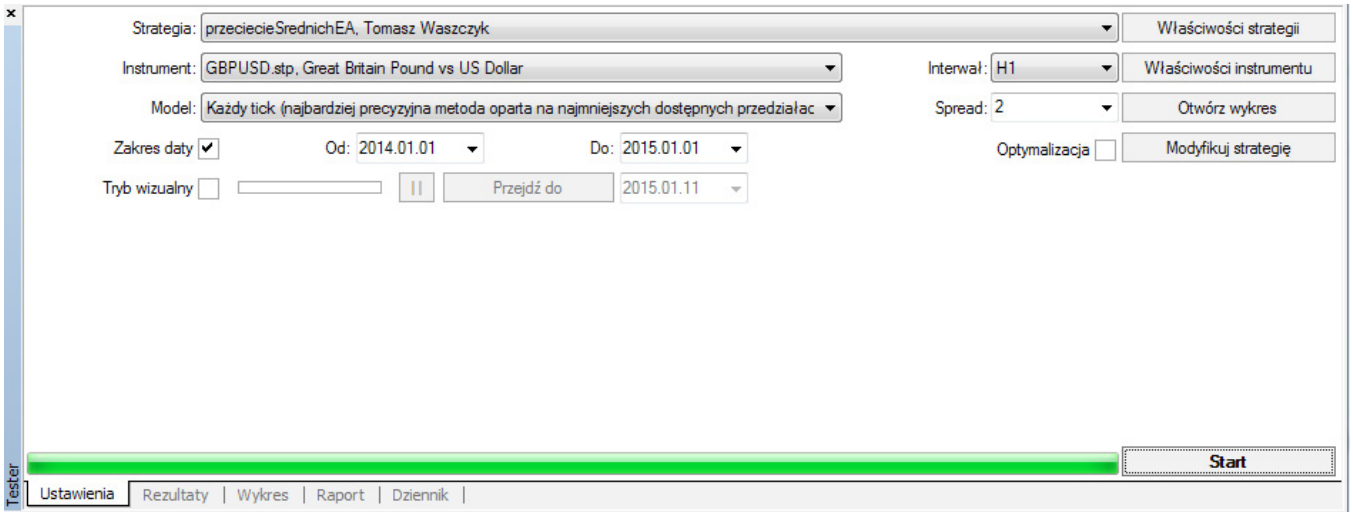
38 http://docs.mql4.com/check/getlasterror

(back testing), dlatego bardzo ważną sprawą jest jakość tych danych. Niestety dane dostępne domyślnie na naszej platformie nie są najlepszej jakości. Warto więc skorzystać z usług firm, które profesjonalnie zajmują się gromadzeniem notowań oraz ich sprzedażą. Można albo zaimportować zakupione dane do naszej platformy, albo – jeśli znamy język C++/Java lub inny o podobnych możliwościach – napisać sobie własny tester, w którym będziemy wykonywać wszystkie obliczenia, a platforma MT4 będzie służyć nam jedynie do wykonywania zleceń.

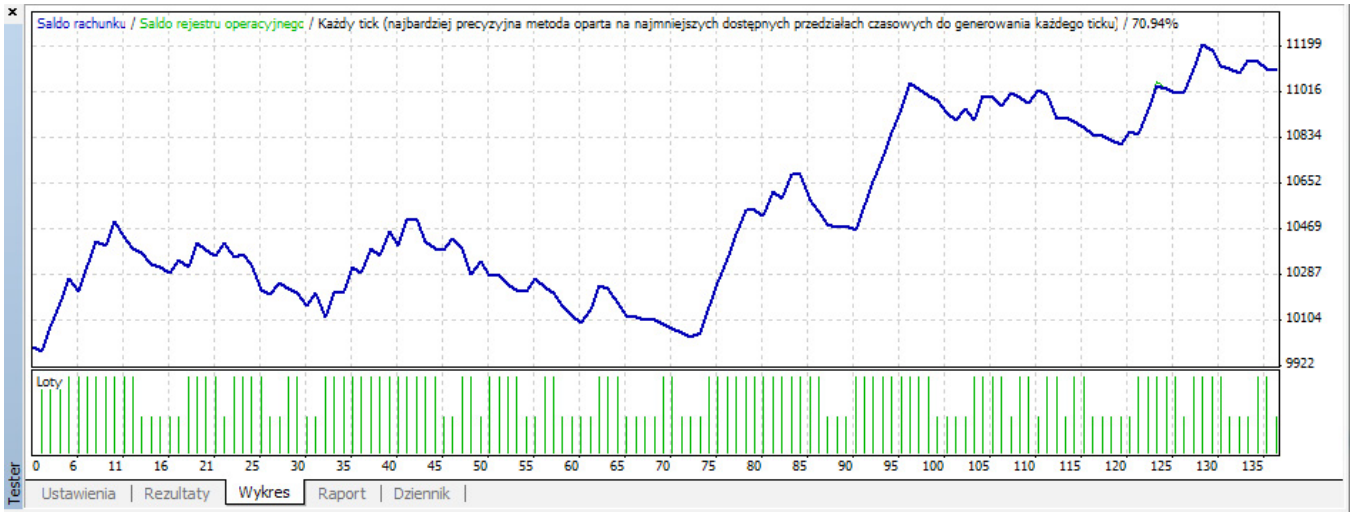
Aby uruchomić tester, należy z zakładki „Widok” wybrać „Tester Strategii”:



W polu „Strategia” wybieramy nazwę strategii, jaką chcemy przetestować, a następnie walor, na jakim test ma zostać przeprowadzony. Pole „Model” daje nam możliwość określenia, w jaki sposób dane historyczne mają zostać wymodelowane, stąd lepiej zainwestować w oryginalne dane historyczne, niż polegać na danych, które są pewnego rodzaju pochodną rzeczywistych danych. Z pola „Interwał” wybieramy interwał waloru, na jakim ma odbywać się test. Aby „obciążyć kosztami” test, należy określić spread, jaki będzie dodany do każdej transakcji, najlepiej ok. 10-15% większy od tego, który rzeczywiście mamy na platformie, w celu urealnienia wyniku. Aby rozpocząć test, klikamy przycisk „Start”, a po niedługim czasie, zależnie od szybkości naszego komputera, zobaczymy potwierdzenie ukończenia testu w postaci zielonego paska postępu:



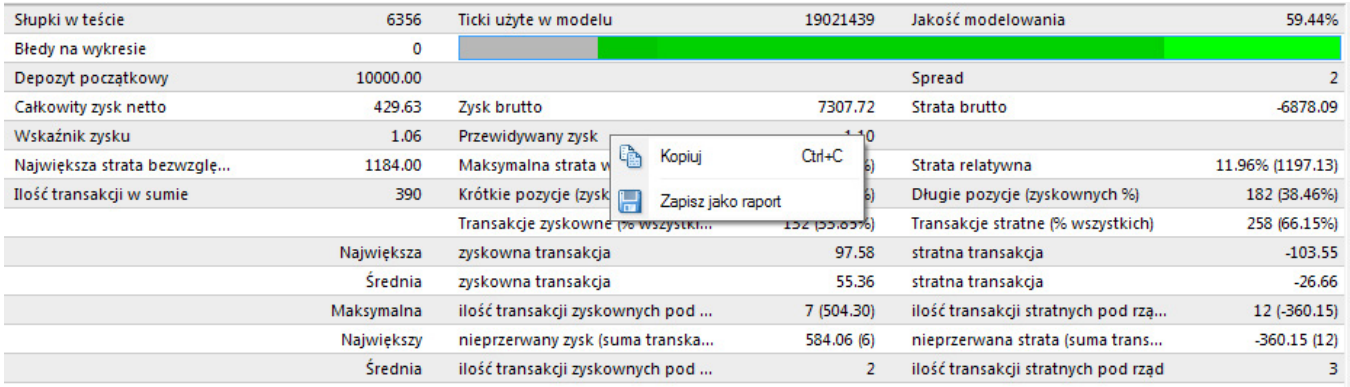
Następnie klikamy zakładkę „Wykres”, gdzie zobaczymy, jak nasza strategia pracowała w zadanym przez nas zakresie czasu:



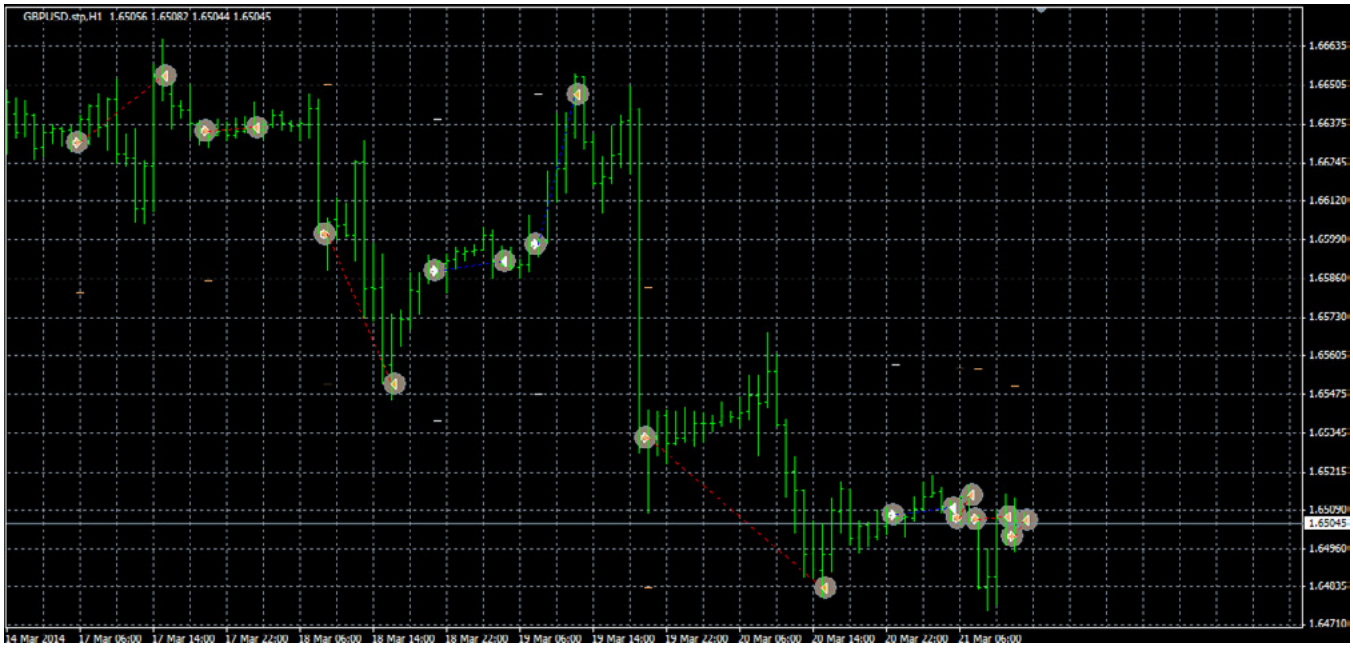
Kiedy przełączymy się do zakładki „Raport”, uzyskamy dostęp do bardziej szczegółowych informacji:

Słupki w teście	6356	Ticki użyte w modelu	19021439	Jakość modelowania	59.44%
Błędy na wykresie	0				
Depozyt początkowy	10000.00			Spread	2
Całkowity zysk netto	429.63	Zysk brutto	7307.72	Strata brutto	-6878.09
Wskaźnik zysku	1.06	Przewidywany zysk	1.10		
Największa strata bezwzględ...	1184.00	Maksymalna strata względna	1197.13 (11.96%)	Strata relatywna	11.96% (1197.13)
Ilość transakcji w sumie	390	Krótkie pozycje (zyskownych %)	208 (29.81%)	Długie pozycje (zyskownych %)	182 (38.46%)
		Transakcje zyskowne (% wszystki...	132 (33.85%)	Transakcje stratne (% wszystkich)	258 (66.15%)
	Największa	zyskowna transakcja	97.58	stratna transakcja	-103.55
	Średnia	zyskowna transakcja	55.36	stratna transakcja	-26.66
	Maksymalna	ilość transakcji zyskownych pod ...	7 (504.30)	ilość transakcji stratnych pod rzą...	12 (-360.15)
	Największy	nieprzerwany zysk (suma transka...	584.06 (6)	nieprzerwana strata (suma trans...	-360.15 (12)
	Średnia	ilość transakcji zyskownych pod ...	2	ilość transakcji stratnych pod rząd	3

Niestety ze względu na to, że dane historyczne są modelowane w odmienny sposób, wyniki tej samej strategii po wykonaniu kilkunastu testów będą się różnić, co wynika z modelowania danych opartego na aproksymacji, gdzie wartości nie będą ciągle takie same. Wynik testu możemy wyeksportować do pliku *.html. Aby to zrobić, należy kliknąć prawy przycisk myszy na wyniku:



Następnie klikamy „Zapisz jako raport”, a po wybraniu miejsca, gdzie chcemy zapisać nasz raport, klikamy „Zapisz”[39]. Istnieje również tryb wizualny testowania, podczas którego możemy wizualnie zaobserwować wykonywane zlecenia na wykresie:



Na koniec chciałbym dodać, że najlepszym sposobem testowania strategii jest handel na koncie demo, czyli tzw. forward-testing. Niestety jest on najbardziej czasochłonny, z własnego doświadczenia jednak przekonałem się, że długoterminowe testy są najbardziej realne.

39 Polecam bardzo interesujący artykuł, w którym jest objaśnione dokładnie, co oznaczają poszczególne wartości widoczne w wyniku testu: http://articles.mql4.com/83.

Funkcja automatycznej zmiany interwału czasowego

Rozdział ten chciałbym poświęcić implementacji możliwości automatycznej zmiany interwałów czasowych na otwartych wykresach. Zazwyczaj podczas handlu obserwujemy więcej niż jeden walor, przez co nasz ekran przypomina ten poniżej:

Często jednak mamy otwarte więcej niż cztery wykresy jednocześnie, przez co sprawdzenie innych interwałów staje się problematyczne i czasochłonne. Standardowo, aby zmienić interwał, najpierw musimy na niego kliknąć, a następnie wybrać ten nas interesujący. W momencie kiedy mamy dziewięć otwartych wykresów, taka operacja może nam zająć dłuższą chwilę. Proponowane przeze mnie rozwiązanie posiada kilka istotnych zalet:

- redukcja do minimum czasu, jaki potrzebujemy na przełączenie interwału czasowego,
- nie ważne, ile wykresów mamy otwartych – na każdym z nich zostanie dokonana zmiana wybranego przez nas interwału czasowego,
- prosta implementacja rozwiązania,
- oszczędność czasu, jaki potrzebujemy na przełączanie się pomiędzy różnymi interwałami.

Zanim przejdziemy do analizy kodu źródłowego, muszę poruszyć temat reprezentacji okienek w systemie MS Windows i zaimplementowania tego mechanizmu w MQL, co pomoże nam w łatwiejszym zrozumieniu załączonego kodu źródłowego.

Przede wszystkim potrzebna jest nam wiedza na temat działania tzw. uchwytów okien (ang. window handler). Podstawowym obiektem systemu Windows są okna, z którymi pracujemy na co dzień. Aby móc przesyłać komunikaty do określonego przez nas okna, musimy mieć dostęp do jego uchwytu, tzw. obiektu HWND, jako że każde okno ma swój unikalny uchwyt i to właśnie dzięki temu mechanizmowi możemy jedno okienko rozróżnić od drugiego oraz przesłać do niego określony komunikat, powodując tym samym np. zmianę interwału wykresu w nim znajdującego się. Za obsługę okien w systemie Windows odpowiada biblioteka „user32.dll", którą wykorzystamy w naszym skrypcie.

Pojawia się teraz pytanie, w jaki sposób możemy wykorzystać wiedzę o uchwytach w języku MQL. Otóż jeśli chcemy przesyłać komunikaty do okien Meta Tradera, musimy skorzystać z funkcji WindowHandle i to właśnie dzięki niej możemy otrzymać uchwyt do okna.

Po tym niezbędnym wprowadzeniu możemy przejść do implementacji naszej funkcji, która będzie zmieniać interwał czasowy[51].

```
// import biblioteki
#import „user32.dll"
  int PostMessageA(int hWnd,int Msg,int wParam,int lParam);
  int GetWindow(int hWnd,int uCmd);
  int GetParent(int hWnd);
#import
```

Powyższy kod odpowiada za zaimportowanie już wcześniej wspomnianej biblioteki DLL „user32.dll", dzięki której możemy pracować z oknami Meta Trader'a. Informację o tym, na jaki interwał czasowy nasz skrypt będzie przełączał wszystkie wykresy, przechowywujemy w poniższej zmiennej:

```
// zmienną naszInterwalCzasowy mozemy dowolnie modyfikować
int naszInterwalCzasowy = PERIOD_M30;
```

Niewątpliwie najważniejszą częścią skryptu dającą nam możliwość przełączania się pomiędzy interwałami jest ciało funkcji start():

```
int start()
{
  bool changeContinue = true;
  int intParent = GetParent(WindowHandle(Symbol(), Period()));
  int intChild = GetWindow(intParent, 0);  // pobranie okien nizej w hierarchii
  int intCmd;
```

Dzięki temu fragmentowi kodu mamy dostęp do głównego okna (zmienna intParent). Kiedy natomiast posiadamy dostęp do okna rodzica wystarczy, że wywołamy funkcję GetWindow z parametrem rodzica, a otrzymamy okienka dzieci. Aby móc obsłużyć wszystkie interwały czasowe, korzystamy z instrukcji switch, w której określamy, w zależności od zmiennej „naszInterwalCzasowy", wartość przesyłanego komunikatu do okna. W naszym przypadku komunikatem jest zmienna intCmd, która zawiera kod interwału czasowego:

```
switch(naszInterwalCzasowy)
{
  case PERIOD_M1:   intCmd = 33137;  break;
  case PERIOD_M5:   intCmd = 33138;  break;
  case PERIOD_M15:  intCmd = 33139;  break;
  case PERIOD_M30:  intCmd = 33140;  break;
  case PERIOD_H1:   intCmd = 35400;  break;
  case PERIOD_H4:   intCmd = 33136;  break;
  case PERIOD_D1:   intCmd = 33134;  break;
  case PERIOD_W1:   intCmd = 33141;  break;
  case PERIOD_MN1:  intCmd = 33334;  break;
}
```

Może zdarzyć się sytuacja, kiedy okno rodzic staje się jedynym otwartym oknem. Musimy przewidzieć taką sytuację i w tym celu sprawdzamy, czy okno, które pełni funkcję dziecka (intChild), jest zarazem rodzicem (intParent), a zatem sprawdzamy typ okna dziecka i rodzica. Jeśli te dwa okna nie są tymi samymi oknami, przesyłamy do nich komunikat z wartością „0" funkcją PostMessageA:

```
PostMessageA(intChild, 0x0111, intCmd, 0);
```

Jak widzimy, wywołanie funkcji okazuje się dość złożone. Drugi parametr 0x0111 oznacza, że wysyłamy do okna komunikat o nazwie WM_COMMAND . Najbardziej interesujący wydaje się parametr ostatni „0". Jest to zakodowana wartość komunikatu GW_HWNDFIRST, która operując na tzw. Z-order sprawdza, czy dany typ okna to taki sam jak typ okna położonego najwyżej w hierarchii (okno rodzic). W ten sposób udało nam się przesłać komunikat do okna, które jest niżej w hierarchii. Na końcu zmieniamy wartość zmiennej „changeContinue" na wartość false, ponieważ nie ma już więcej okien, których interwał moglibyśmy zmienić:

```
if (intChild > 0)
{
  if (intChild != intParent)
    PostMessageA(intChild, 0x0111, intCmd, 0);
}
else
  changeContinue = false;
```

Poniższy kod wydaje się bardzo podobny do tego już omówionego. Jedna zasadnicza zmiana dotyczy tego, że w momencie, kiedy zmienna „changeContinue" jest prawdziwa, wykonujemy pętlę while, pobierając okna znajdujące się niżej w hierarchii i za pomocą funkcji GetWindow przekazując parametr „2". Jest to zakodowana wartość uchwytu do okna „GW_HWNDNEXT", która, jak nazwa wskazuje, szuka następnych okien o takim uchwycie. Jeśli takie okno istnieje, wykonujemy takie same operacje jak w przypadku instrukcji warunkowej „if". Cały czas sprawdzamy, czy istnieje okno, do którego powinniśmy przesłać odpowiedni komunikat, tzn.:

```
if (intChild > 0)
```

Całość kodu możemy prześledzić poniżej:

```
while(changeContinue)
{
  intChild = GetWindow(intChild, 2);

  if (intChild > 0)
  {
    if (intChild != intParent)
      PostMessageA(intChild, 0x0111, intCmd, 0);
  }
  else
```

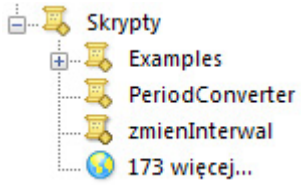
W przeciwnym przypadku zmieniamy zmienną sprawdzającą, czy mamy okna do przeprocesowania na false.:

```
    changeContinue = false;
}
```

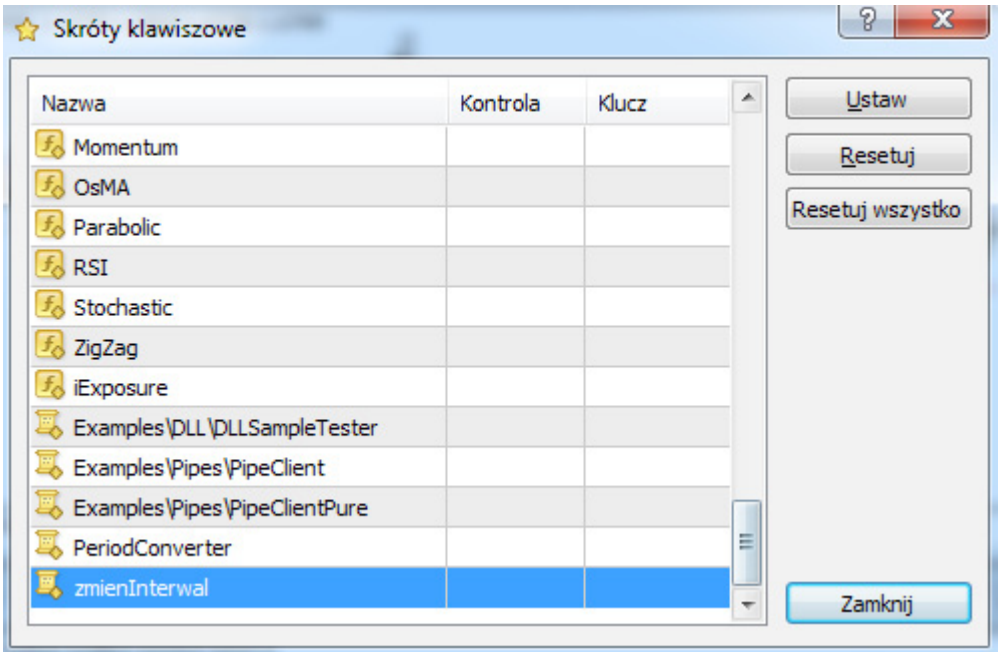
Nie możemy też zapomnieć o zmianie interwału rodzicu:

```
  // Przeslij komunikat do aktualnie otwartego okna
  PostMessageA( intParent, 0x0111, intCmd, 0 );
}
```

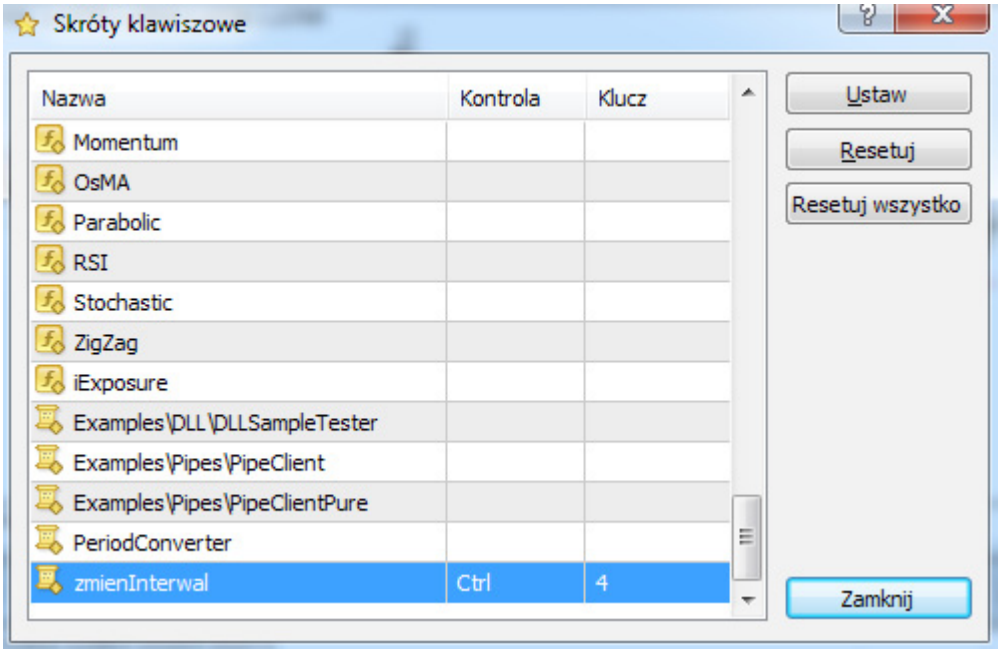
Po omówieniu kodu pokażę, jak go wykorzystać na naszej platformie.
Najpierw tworzymy sobie nowy wskaźnik, następnie kopiujemy kod źródłowy do naszej platformy i zapisujemy, w efekcie czego powinniśmy zauważyć nasz skrypt na platformie:



W moim przypadku jest to skrypt o nazwie „zmienInterwał”. Następnie do naszego skryptu przypisujemy skrót klawiszowy, poprzez kliknięcie prawym przyciskiem na nasz skrypt, a następnie wybranie „Skrót klawiszowy”. Pojawi nam się takie okno:



Przypisanie zależy od naszych preferencji. W moim przypadku wybrałem skrót CTRL + 4:



Z uwagi na to, że w kodzie mamy zmienną, która określa nam interwał, zgodnie z którym skrypt będzie przełączał nam interwały czasowe, proponuję utworzyć następujące skróty:

- CTRL + 1 jako zmiana interwału M1
- CTRL + 2 jako zmiana interwału M5

Plik ze skryptem kopiujemy z odpowiednią nazwą, np. zmienInterwalM30.

Analogicznie wykonujemy taką operację dla wszystkich interwałów, które nas interesują. Efektem końcowym jest możliwość przełączania się bardzo szybko pomiędzy interwałami jednym skrótem klawiszowym (jeden skrót klawiszowy na jeden interwał).

Zakończenie

Właśnie dotarłeś, Drogi Czytelniku, do końca mojej książki. Zapewne teraz czujesz wielki głód informacji, dlatego chciałbym Ci przekazać kilka rad i wskazówek, co zrobić dalej po skończeniu mojej lektury. Osobiście uważam, że rozwój kompetencji w przypadku programisty automatycznych systemów transakcyjnych odbywa się dwoma nurtami: „rynkowym" oraz „technologicznym". W związku z tym, że tematem przewodnim tej publikacji było programowanie strategii, doradzę Ci kierunek technologiczny. Jak zapewne zauważyłeś podczas lektury, do tekstu dodawałem odnośniki do oficjalnej i szeroko dostępnej dokumentacji MQL, z uwagi na to, że nie byłem w stanie zamieścić wszystkich informacji na stronach tej książki. Niestety osoby nie znające języka Szekspira będą mieć trudniej, jako że zdecydowana większość dostępnej literatury oraz dokumentacja i inne materiały są dostępne w języku angielskim. Co ciekawe, istnieje również bardzo szeroka literatura przedmiotowa w języku rosyjskim.

Poruszyłem, niestety dość pobieżnie, parę problemów, takich takich chociażby opis wersjonowania naszego oprogramowania. Jeśli czujesz, że informacje są niepełne albo coś jest niezrozumiałe, proszę, wyślij mail na mój adres: tomasz@waszczyk.com. Postaram się też napisać parę artykułów jako uzupełnienie tej publikacji. Chciałbym poprosić każdego z Czytelnika, aby podzielili się ze mną wszelkimi uwagami. To dla mnie o tyle ważne, iż jest to moja pierwsza tego typu publikacja.

Dodatki

VPS czyli jak mieć stale uruchomiony automat mając wyłączony komputer.

Kluczową kwestią związaną z programowaniem automatycznych strategii transakcyjnych jest zapewnienie możliwości ciągłego analizowania rynku. Jedną z największych zalet EA jest możliwość analizowania rynku ciągle, w sposób nieprzerwany. Rodzi się jednak pytanie: jak to zapewnić? Niektórzy wykorzystują swoje prywatne komputery jako maszyny docelowe, na których będą handlować. Proszę sobie wyobrazić, co się stanie, kiedy zabraknie nam prądu, nagle stracimy dostęp do sieci Internetu czy nasz komputer najzwyczajniej w świecie się zawiesi. Wszystkie z wymienionych przyczyn skutkują de facto wyłączeniem naszego automatu i nie mamy nad tym jakiejkolwiek kontroli. Jeśli takich przerwań w działaniu jest więcej, możemy uznać, że w ogóle nie panujemy nad naszym automatem, co jest bolesne w sytuacji, kiedy automat ma otwarte pozycje i pozostaje bez nadzoru: ani naszego osobistego, ani automatu.

Aby ustrzec się przed takimi sytuacjami, warto zainstalować Meta Tradera na serwerze VPS (Virtual Private Server). Firmy, które zajmują się tego typu usługami są przygotowane na awarie sprzętu oraz wpływ czynników od nich niezależnych, takich jak przerwy w dostępie do sieci Internet, poprzez równoległe połączenia. W momencie kiedy jedno łącze przestaje spełniać swoją rolę, następuje automatyczne przełączenie w taki sposób, że klienci nie są świadomi tego, że jedno z łączy uległo awarii. Logowanie się do serwera VPS może przebiegać za pomocą standardowego oprogramowania dostępnego w systemie MS Windows – Pulpit Zdalny (Remote Control) lub oprogramowania komercyjnego (zazwyczaj darmowego dla celów osobistych). Nie ma też problemu z zalogowaniem się na serwer przy pomocy urządzenia mobilnego. Dzięki temu rozwiązaniu jesteśmy pewni, że nawet gdy my nie mamy dostępu do Internetu, nasz automat stale monitoruje rynek. Komputer możemy spokojnie zamknąć lub wyjść z nim do kawiarni, by swobodnie porozmawiać z klientem o nowym kontrakcie.

Najpopularniejsze błędy logiczne w automatycznych strategiach

Zauważyłem, że niemalże każdy początkujący programista automatów bardzo szybko napotyka na bardzo trudne do wykrycia problemy, których przyczyny tkwią w samej logice programu. Jest to o tyle niebezpieczne, o ile błędy na poziomie kodu źródłowego zostają zauważone automatycznie. Mam tu na myśli po prostu brak pomyślnej kompilacji. Dużo większym zagrożeniem są błędy logiczne, które nie zostają wyłapane na poziomie kompilacji, a powinny zostać jak najszybciej wychwycone na poziomie testowania na rzeczywistych danych. Wielu początkujących programistów uznaje jednak, że „odkryli Graala", dzięki czemu szybko staną się milionerami. Napiszę więc na koniec parę słów, aby mogli oni uniknąć tego typu rozczarowań na rynku inwestycyjnym.

Kiedy programujemy system transakcyjny, powinniśmy zadbać o to, aby symulowanie handlu odbywało

się w warunkach jak najbardziej zbliżonych do rzeczywistych. Co ważne, przed uruchomieniem systemu automatycznego powinniśmy wiedzieć, czego możemy się po nim spodziewać, aby uniknąć rozczarowań oraz zaskoczenia podczas rzeczywistego tradingu. Poniżej opiszę najczęściej popełniane błędy:

- Niemożliwa logika – mam na myśli sytuacje w kodzie źródłowym, które są jak najbardziej możliwe w przypadku kiedy mamy załadowane dane historyczne, a niemożliwe na rzeczywistym rynku. Przykładem takiego błędu może być podejmowanie decyzji inwestycyjnej po podstawie (n+1) świeczki, co w przełożeniu na język potoczny jest przeniesieniem się w przyszłość.
- Ignorowanie kosztów transakcyjnych – w przypadku rynku walutowego jest to zazwyczaj różnica pomiędzy Ask a Bid, czyli spread. O ile w systemach, których celem jest łapanie dłuższych ruchów cenowych, okazuje się to mało istotne, to już w przypadku handlu na tikach lub danych o niskiej granularności, może mieć fatalne skutki. Niemniej chciałbym wyrazić się jasno – brak implementacji kosztów transakcyjnych jest błędem i zawsze powinniśmy brać koszta transakcji pod uwagę, dokładnie tak samo jak w każdej innej dziedzinie biznesu.
- Przeoptymalizowanie systemu – klasyczny błąd polegający na dopasowaniu parametrów systemu transakcyjnego do danych historycznych. Problemem są tu bardzo prymitywne testery systemów, polegające np. na tym, że okres średniej (jeśli korzystamy ze średnich) jest dopasowany tak, aby otrzymać jak najwyższy zwrot z inwestycji. W rzeczywistości nie możemy zrobić takiej operacji, stąd mając system, dla którego stopa zwrotu na danych historycznych jest dużo wyższa niż w rzeczywistości, z dużym prawdopodobieństwem nie mamy szans utrzymać się na rynku.
- Niekompletne dane historyczne – może to być zaskoczeniem dla wielu Czytelników, jednak nieraz spotkałem się z sytuacją, kiedy testy zrobione na danych dostarczonych przez brokera na platformie Meta Trader bardzo różniły się od testów przeprowadzonych na własnym testerze. Problem tkwi w jakości danych historycznych. Nie polecam danych, które są standardowo dostępne i udostępniane na platformach, bo zazwyczaj są one bardzo wybrakowane, posiadają wiele luk, a im starsze, tym bardziej odbiegają od rzeczywistych cen. Dlatego proponuję wykupić za stosunkowo niewielką kwotę dane u firm, które zajmują się archiwizacją notowań.
- Zbyt mały okres testowania – należy mieć świadomość, że rynek stale ulega zmianie. Największą bolączką w programowaniu automatycznych systemów transakcyjnych jest fakt, że musimy te zmiany przewidzieć i zaprogramować system tak, aby móc dostosować się do nich. Nie należy zakładać, że system zarabiający przez ostatni kwartał będzie zarabiać już zawsze, dlatego bardzo ważne jest testowanie na możliwe najdłuższym okresie danych, aby system mógł się „wykazać" w jak najdłuższym okresie czasu.
- Niezrozumienie skali osi OY podczas czytania raportu z testera.

Odnosząc się do ostatniej kwestii opiszę następujący przypadek. Jakiś czas temu pewien klient podesłał mi automat, według którego na podstawie testów uznał, że będzie zarabiać ok. 70% rocznie. Kiedy wczytałem się w kod systemu, wydawał się on bardzo rozsądny, na tyle, że postanowiłem przetestować go na własnym testerze i w oparciu o własne dane. Jaki byłem zdziwiony, kiedy system okazał się być programem najzwyczajniej tracącym pieniądze. Raz jeszcze poprosiłem o wcześniej przeprowadzone testy i „jedyną" różnicą były dwie rzeczy:

- **wstępny depozyt**
- **wolumen poszczególnej transakcji**

W przypadku gdy testy były wykonywane przy kwocie wstępnej w wysokości 1 000 000 euro system był zyskowny, były widoczne jedynie z pozoru niewielkie obsunięcia na kapitale. Kiedy zmieniłem skalę osi OY i przyjrzałem się jej, obsunięcia te wynosiły momentami 60% kapitału, jednak skala była na tyle duża, że wydawały się dużo mniejsze. Przy kwocie 10000 euro system okazał się być stratny – niestety na żywym rachunku.

Często używane skróty klawiszowe

F4 – uruchomienie MetaEditor z poziomu Meta Trader

F4 – szybkie przejście do Meta Trader z poziomu Meta Editor

F7 – kompilacja kodu źródłowego

ALT + M – lista funkcji z poziomy Meta Editor

CTRL + SPACJA – podpowiedź kontekstowa w Meta Editor

Literatura

1. Michael Lewis, Flash Boys.
2. Haim Bodek, The Problem of HFT –Collected Writings on High Frequency Trading and Stock Market, http://haimbodek.com
3. ALGLIB, Numerical Analysis Library dla Meta Trader 5
4. Strona internetowa: http://docs.mql4.com
5. Strona internetowa: https://www.mql5.com/en/articles/81

Redakcja i korekta: Joanna Tyka

Autor dołożył wszelkich starań, by zawarte w niej treści były prawdziwe i rzetelne, nie ponosi jednak odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w publikacji. Rynki OTC charakteryzują się dużym ryzykiem straty, która może przekroczyć początkowy depozyt.

Treści zawarte w niniejszej pracy nie stanowią rekomendacji w rozumieniu przepisów Rozporządzenia Ministra Finansów z dnia 19 października 2005 r., w sprawie informacji stanowiących rekomendacje dotyczące instrumentów finansowych, lub ich emitentów (Dz.U. z 2005 r. Nr 206, poz. 1715).

Partner publikacji:

TMS Brokers
DOM MAKLERSKI