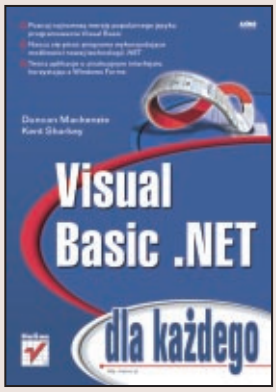
# Visual Basic. NET dla każdego




Autorzy: Duncan Mackenzie, Kent Sharkey
Tłumaczenie: Tomasz Miszkiel
ISBN: 83-7197-700-X
Tytuł oryginału: Teach Yourself Visual Basic .NET in 21 Days
Format: B5, stron: 624
Przykłady na ftp: 1889 kB

O znaczeniu języka Visual Basic nie trzeba nikogo przekonywać: jest to najpopularniejszy język programowania na świecie, umożliwiający nawet początkującym rozpoczęcie przygody z programowaniem w środowisku Windows. Teraz jednak jego rola jest jeszcze większa: stał się on bowiem, w nowej wersji nazwanej VisualBasic.NET, składnikiem platformy .NET Microsoftu. Dzięki zintegrowaniu z .NET VisualBasic przestał być językiem „drugiej kategorii": jego możliwości są obecnie prawie porównywalne z C++.

21 napisanych przystępnym językiem rozdziałów opisuje między innymi:

- ewolucję VisualBasica i budowę platformy .NET
- składnię języka
- sposoby eliminowania błędów w aplikacjach
- programowanie zorientowane obiektowo
- tworzenie interfejsu aplikacji za pomocą Windows Forms
- pisanie aplikacji działających w Internecie
- korzystanie z baz danych, plików tekstowych i graficznych

Książka, napisana przed dwóch doświadczonych programistów, będących pracownikami Microsoftu, opisuje więc nie tylko sam VisualBasic.NET, ale także platformę .NET oraz narzędzia przeznaczone do tworzenia aplikacji. Na końcu każdego rozdziału znajdują się ćwiczenia, pozwalające czytelnikowi sprawdzić swoją wiedzę. Jest to zarówno książka dla tych, którzy programowali w VisualBasicu wcześniej, a teraz chcą poznać jego nowe możliwości, jak i dla zupełnych nowicjuszy.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

# Spis treści

Rozdział .

# Praca z Visual Basic .NET

Poprzedni rozdział zawierał podstawowe informacje dotyczące programowania, języka Visual Basic i platformy .NET. W niniejszym rozdziale stworzymy właściwą aplikację działającą w systemie Windows. Omówimy następujące zagadnienia:

- Środowisko Visual Studio
- Praca z plikami, projektami i rozwiązaniami
- Przykładowy projekt dla Windows

Najpierw poznamy zintegrowane środowisko programistyczne Visual Studio (VS).

## Środowisko Visual Studio

Zintegrowane środowisko programistyczne (IDE — Integrated Development Environment) jest przeznaczone dla programistów tworzących aplikacje. Można, oczywiście, programować bez korzystania ze środowiska programistycznego i wpisywać kod źródłowy w edytorze tekstowym (np. w Notatniku), a następnie powstałe w edytorze pliki kompilować z wiersza poleceń. Programowanie w taki sposób jest jednak kłopotliwe i czasochłonne — niewielu programistów zdecydowałoby się na tworzenie oprogramowania bez wykorzystania środowiska programistycznego, które zawiera wiele funkcjonalnych narzędzi wspomagających tworzenie programów. Środowisko programistyczne pozwala na tworzenie rozbudowanych i funkcjonalnych aplikacji w bardzo krótkim czasie.

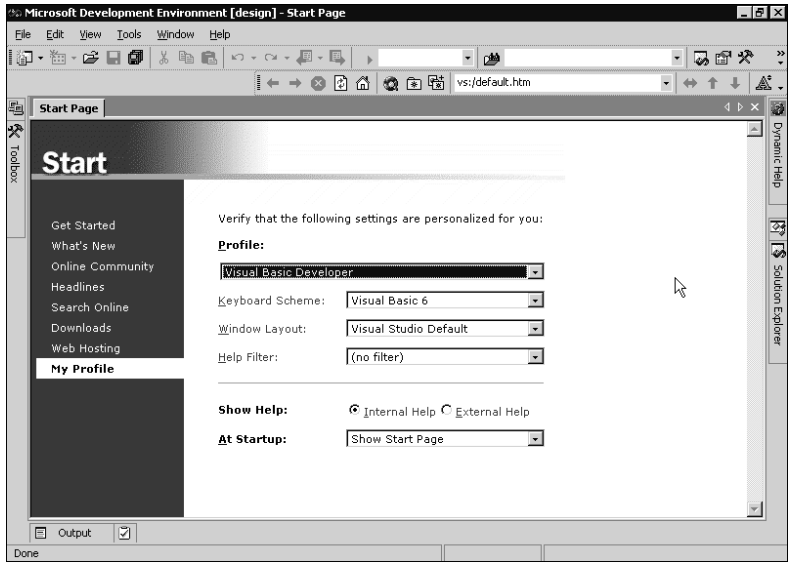
### Zanim zaczniemy

Zanim rozpoczniemy pracę w Visual Basicu, musimy się upewnić, że nasze środowisko jest zainstalowane. Jeśli mamy jakiekolwiek wątpliwości związane z procesem instalacji, wróćmy do poprzedniego rozdziału.

## Ustawienia parametrów środowiska

Podczas pierwszego uruchomienia Visual Studio .NET pojawia się okno przypominające swym wyglądem okno przeglądarki internetowej. Jest to *Start Page* — strona startowa zintegrowanego środowiska programowego Visual Studio .NET. W tym środowisku znajdą się wszystkie nasze projekty, począwszy od prostych aplikacji, po strony WWW.

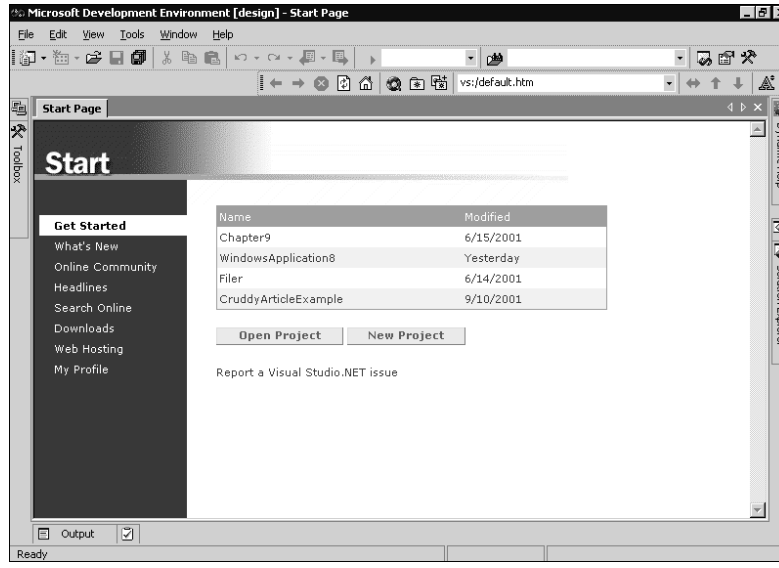
W lewej części ekranu znajdują się odnośniki do kilku interesujących stron. Domyślnie wybrana jest strona służąca do określenia ustawień profilu użytkownika *My Profile*. Na tej stronie należy wybrać odpowiednie ustawienia parametrów środowiska wedle własnego uznania. Każdy użytkownik ma do wyboru kilka konfiguracji środowiska Visual Studio. Aby wybrać odpowiednią dla siebie konfigurację, należy trochę „poeksperymentować" z różnymi ustawieniami. Na razie pozostawmy ustawienia domyślne, takie jak na rysunku 2.1.




**Rysunek 2.1.** *Środowisko Visual Studio .NET można ustawić według własnego uznania tak, aby łatwo przełączać się pomiędzy poszczególnymi narzędziami programistycznymi, łącznie ze starszą wersją Visual Basica*

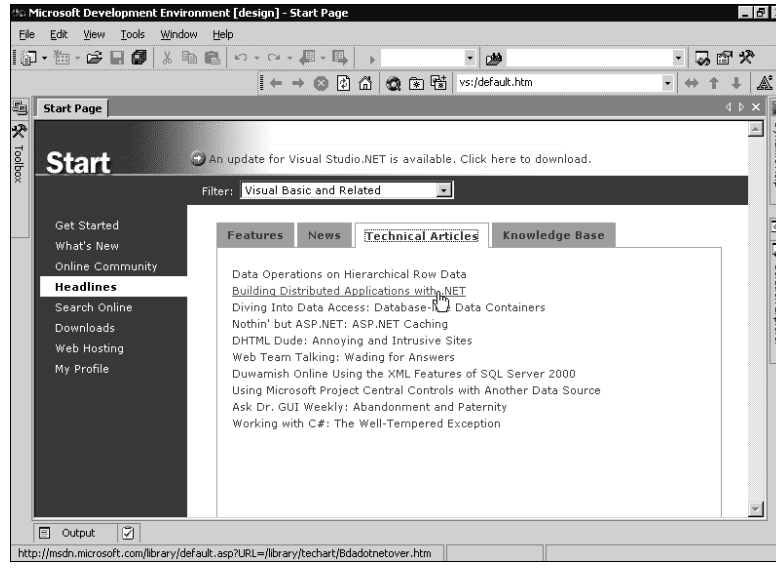
Na razie należy wybrać profil *Visual Basic Developer*. Aby opuścić stronę *My Profile*, należy kliknąć odnośnik do strony *Get Started*. To, co widzimy, jest środowiskiem programistycznym, w którym możemy korzystać z wielu zawartych w nim narzędzi, służących do tworzenia oprogramowania. W środowisku umieszczono przeglądarkę internetową, która pozwala na wyświetlanie witryny startowej środowiska (rysunek 2.2).

Strona *Get Started* zawiera odnośniki do istniejących, wcześniej utworzonych, projektów oraz dwa przyciski. Jeden z nich służy do utworzenia nowego projektu *New Project,* a drugi — do otwarcia jednego z istniejących projektów *Open Project*. Ze strony startowej można, za pomocą odnośników umieszczonych w lewej części ekranu, wybrać jedną z ośmiu stron. Oprócz omówionej strony *Get Started* można wejść do:




**Rysunek 2.2.** *Strona startowa Visual Studio zawiera informacje dotyczące wcześniej utworzonych projektów*

- *Online Community*. Tu znajdują się odsyłacze do stron i grup dyskusyjnych poświęconych Visual Studio .NET.
- *Headlines*. Na tej stronie można znaleźć wiadomości dotyczące środowiska Visual Studio i związanych z nim narzędzi. Aby strona otwierała się poprawnie, musimy mieć połączenie z Internetem (rysunek 2.3).
- *Search Online*, czyli wyszukiwarka internetowa.
- *My Profile*, strona ustawienia parametrów środowiska.




**Rysunek 2.3.** *Najnowsze informacje dotyczące Visual Basic .NET można znaleźć na stronie Headlines, gdzie są odsyłacze do stron umieszczonych w witrynie msdn.Microsoft.com*

Strona startowa Visual Studio .NET została zaprojektowana w taki sposób, by sprostać oczekiwaniom większości użytkowników. W razie potrzeby można zmienić zawartość strony startowej — kod źródłowy jest dostępny w folderze *\Program Files\Microsoft Visual Studio .NET\HTML*. Należy pamiętać, że może zdarzyć się sytuacja gdy modyfikacja strony startowej doprowadzi do jej uszkodzenia, bez możliwości odtworzenia. Najlepiej przed przystąpieniem do wprowadzania jakichkolwiek zmian w folderze HTML utworzyć jego kopię bezpieczeństwa.

# Główne okna środowiska Visual Studio

Visual Studio zawiera wiele użytecznych okien. Po wybraniu profilu Visual Basic Developer, na ekranie komputera powinny znajdować się okna: omówionej wbudowanej przeglądarki, okno *Solution Explorer* (w którym pokazane są tworzone projekty i pliki wygenerowane podczas ich tworzenia), okno *Properties* (właściwości) i okno narzędzi *Toolbox*. Istnieją jeszcze inne okna, niewidoczne po pierwszym uruchomieniu. Są to między innymi:

- *Object Browser*
- *Command/Immediate*
- *Task list*
- *Class View*
- *Server Explorer*

Te okna zostaną przedstawione w tym rozdziale, a o pozostałych będziemy dowiadywać się w kolejnych rozdziałach.

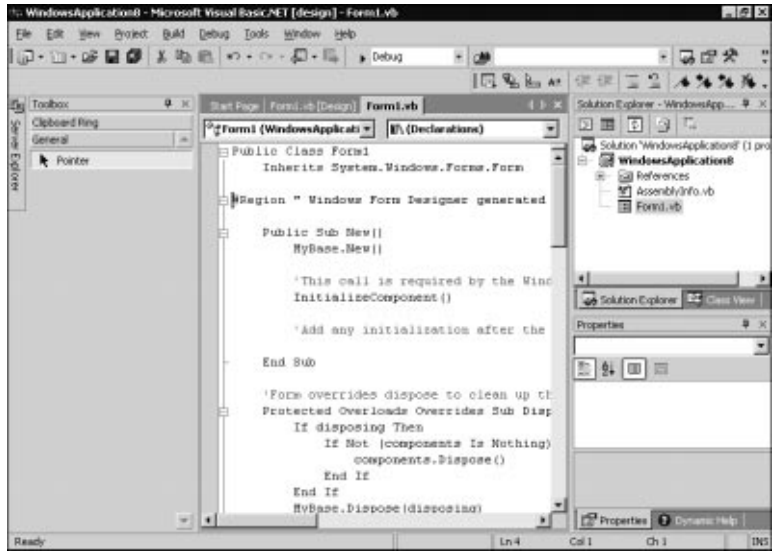
## Właściwości okien środowiska

Okna w środowisku Visual Studio są przystosowane do efektywnego wykorzystania powierzchni ekranu środowiska. Visual Studio .NET zapewnia wiele narzędzi i opcji służących do efektywnego tworzenia aplikacji przez programistów. Głównym celem twórców środowiska programistycznego było zapewnienie wielu narzędzi, do których jest łatwy dostęp. Umieszczenie odnośników do tych narzędzi w jednym oknie jest niemożliwe (chyba że programista dysponuje monitorem o przekątnej 21'). Aby umożliwić dostęp do narzędzi i ułatwić ich wybór, projektanci środowiska programistycznego podzielili narzędzia na grupy i umieścili każdą grupę w jednym oknie. W ten sposób powstała pokaźna liczba okien zawierających poszczególne narzędzia. Te okna nazwano *oknami narzędziowymi*. Wszystkie okna można wyrównywać do krawędzi ekranu i łączyć, czyli *dokować*, a także dowolnie zmieniać ich rozmiar.

### Dokowanie okien

Po uruchomieniu Visual Studio .NET z wybranym ustawieniem środowiska *Visual Basic Developer* pojawia się okno pokazane na rysunku 2.4. Okno to przypomina trochę środowisko programistyczne Visual Basica 6.0. W lewej części ekranu znajduje się okno narzędziowe *Toolbox*, a po prawej stronie jest okno *Solution Explorer* oraz

okno właściwości *Properties*. W prawym górnym rogu omawianych okien znajdują się dwa przyciski. Jeden z nich służy do zamknięcia okna (przycisk z „krzyżykiem" po prawej stronie), a drugi — do umieszczenia okna na pasku (jest to przycisk z „pinezką", którą „przypinamy" okno do ekranu). Każde okno narzędziowe można dokować w dowolnym miejscu ekranu.




**Rysunek 2.4.** *Środowisko Visual Studio .NET, uruchomione w trybie Visual Basic przypomina wyglądem środowisko znane z Visual Basica 6.0*

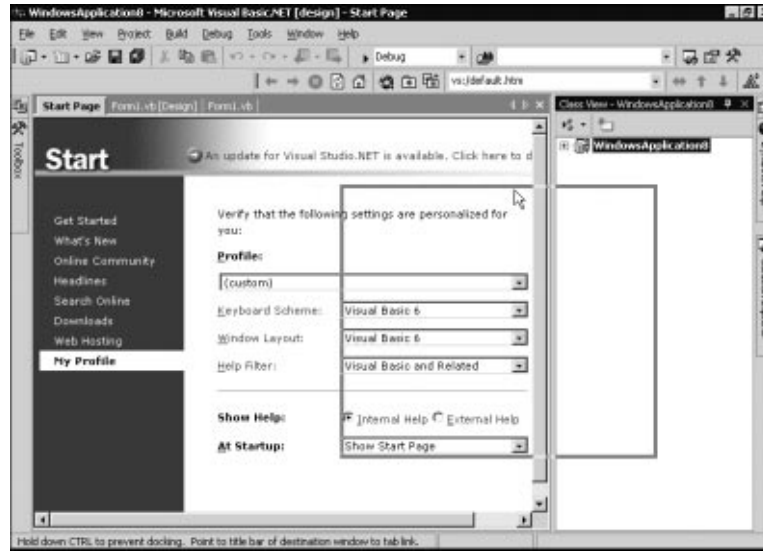
Aby zmienić położenie zadokowanego okna, wystarczy kliknąć i przytrzymać lewy przycisk myszy na pasku tytułu, a następnie przeciągnąć okno w pożądane miejsce. Podczas przesuwania na ekranie widoczny jest zarys przemieszczanego okna, który oznacza położenie okna po zwolnieniu lewego przycisku myszy. Aby zadokować okno, wystarczy nacisnąć i przytrzymać lewy przycisk myszy na pasku tytułu, a następnie przeciągnąć okno do brzegu ekranu i zwolnić przycisk myszy. Na rysunku 2.5 pokazano przykładowy zarys okna, widoczny podczas przemieszczania.

Gdy podczas przemieszczania zarys nie przylgnie do brzegu ekranu, okno stanie się obiektem samodzielnym, nie będzie zadokowane (rysunek 2.6).

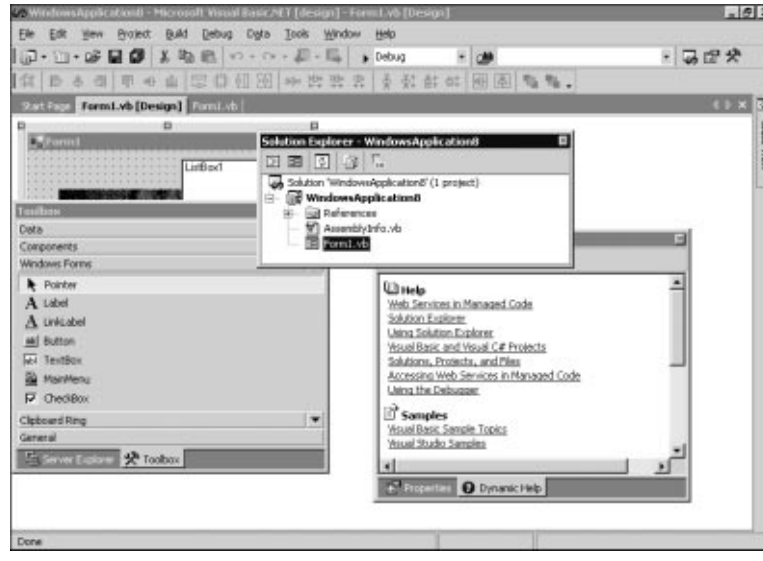
Dokowanie okna może na początku sprawiać kłopoty. Aby szybko zadokować okno, wystarczy dwa razy kliknąć myszą w obszarze paska tytułu, co spowoduje zadokowanie okna do brzegu, z którego okno zostało zdjęte.

## Ukrywanie okna

Niektóre okna, zadokowane lub samodzielne, mogą okazać się w danej chwili nieprzydatne i nie ma potrzeby, by zajmowały miejsce na ekranie. Aby zamknąć zbędne okno, wystarczy kliknąć myszą przycisk „X" znajdujący się w prawym górnym rogu, na pasku tytułu (rysunek 2.7).

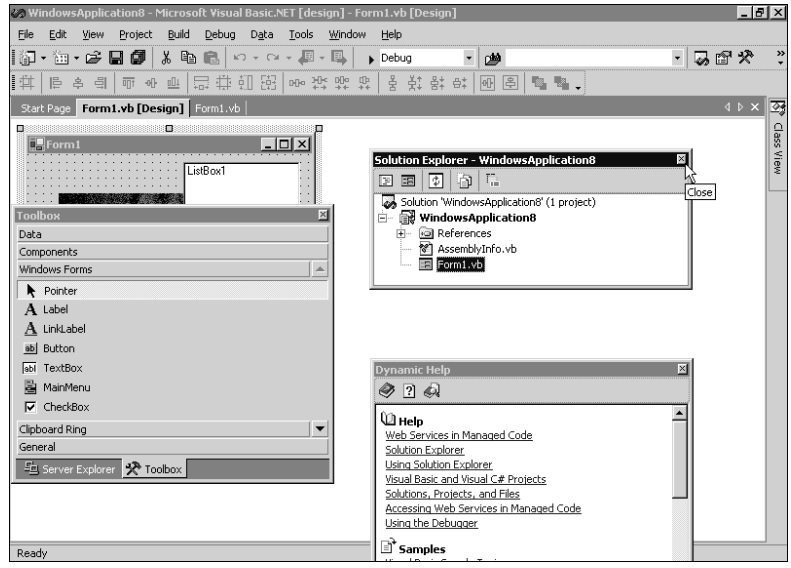



**Rysunek 2.5.** *Zarys okna, widoczny podczas jego przemieszczania, wskazuje na nowe miejsce, w którym okno znajdzie się po zwolnieniu lewego przycisku myszy*

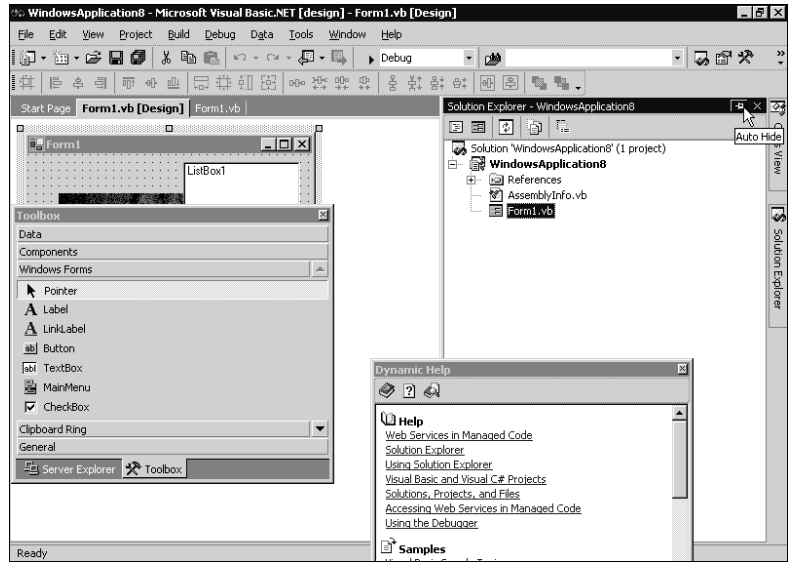



**Rysunek 2.6.** *Okna można umieścić w dowolnym miejscu ekranu jako obiekty samodzielne*

Aby odzyskać wcześniej zamknięte okno, należy wybrać je z głównego menu *View*. Jeśli potrzebny jest częsty dostęp do narzędzi zawartych w oknie, które chcemy ukryć, wystarczy włączyć opcję automatycznego ukrywania okna *Auto-Hide*. Aby włączyć tę opcję, należy nacisnąć przycisk w kształcie pinezki znajdujący się na pasku tytułu okna narzędziowego (rysunek 2.8). Po włączeniu opcji *Auto-Hide* okno narzędziowe zostaje ukryte, ale na jednym z brzegów ekranu jest widoczna zakładka ukrytego okna. Za każdym razem, kiedy kursor myszy znajdzie się na obszarze zakładki, okno staje się widoczne. Okno zostanie ponownie ukryte, gdy przesuniemy kursor myszy poza jego obszar. W menu Windows znajduje się opcja *Auto Hide All*, która powoduje, że wszystkie okna na ekranie zostają automatycznie ukryte (rysunek 2.9). Dzięki tej opcji możemy zwiększyć powierzchnię obszaru edycji kodu źródłowego i okna dialogowego (formy).



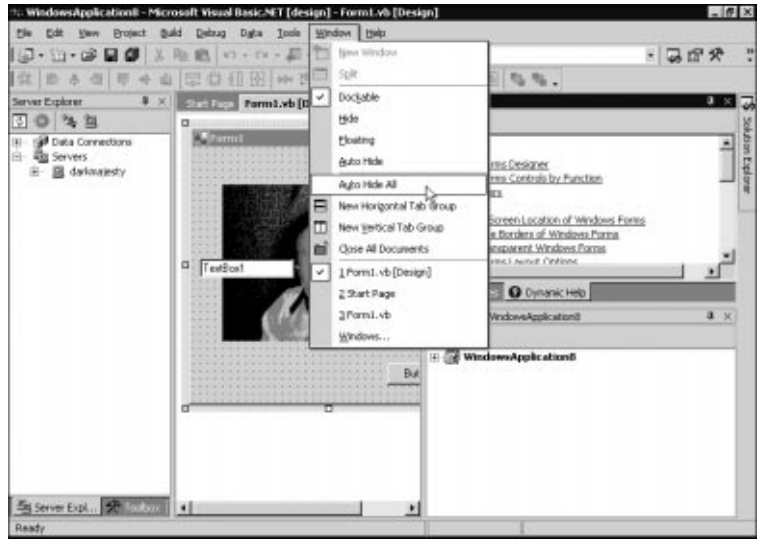

**Rysunek 2.7.** *Okna można zamknąć poprzez naciśnięcie lewym przyciskiem myszy przycisku „X”. Zamknięte okna można otwierać (za pomocą poleceń menu View)*




**Rysunek 2.8.** *Każde okno narzędziowe ma przycisk w kształcie pinezki, który pozwala na pozostawienie okna we właściwym miejscu ekranu*

## Zakładki

Aby umieścić kilka okien na pewnym fragmencie ekranu, należy skorzystać z zakładek (rysunek 2.10). Kilka zakładek jest widocznych po uruchomieniu środowiska Visual Basic (np. okno *Solutions Explorer* jest wyświetlane w tym samym obszarze, co okno *Class View*). Okna narzędziowe możemy łączyć dowolnie, oszczędzając powierzchnię ekranu.




**Rysunek 2.9.** *Można użyć opcji Auto Hide All, aby zwiększyć obszar edycji*


**Rysunek 2.10.** *Korzystając z zakładek możemy mieć otwartych wiele okien narzędziowych i przełączać się pomiędzy nimi*

Aby stworzyć kombinację okien, wystarczy przeciągnąć jedno okno i umieścić na drugim. W ten sposób będzie widoczne tylko jedno okno, a w dolnej części ekranu znajdą się dwie zakładki. Kliknięcie drugiej zakładki spowoduje przełączenie okien. W ten sposób można tworzyć kombinację wielu okien. Aby rozdzielić okna, wystarczy przycisnąć i przytrzymać lewy przycisk myszy na jednym z okien i przeciągnąć w inne miejsce ekranu.

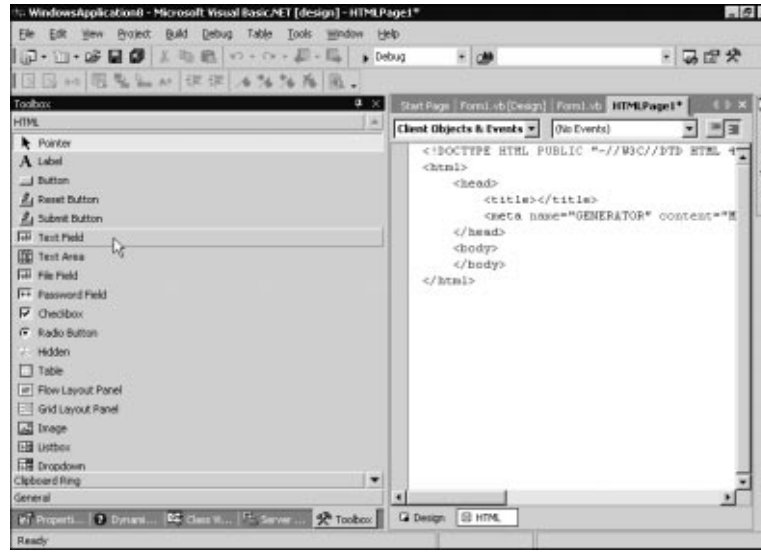
## Zmiana rozmiarów okna

W środowisku Visual Studio można dowolnie zmieniać rozmiary okna. Wyjątek stanowią okna zadokowane, w których można zmienić tylko szerokość. Aby zmienić rozmiary okna, należy ustawić wskaźnik myszy na krawędzi okna tak, by zmienił się

on na wskaźnik zmiany rozmiaru. Następnie trzeba nacisnąć i przytrzymać lewy przycisk myszy i zwiększyć (lub zmniejszyć) rozmiary okna. Gdy przy jednej krawędzi ekranu są zadokowane dwa okna, próba zmiany rozmiaru jednego z nich wpływa na zmianę rozmiaru drugiego.

## Okno narzędziowe Toolbox

Jednym z najczęściej używanych okien w trakcie projektowania aplikacji jest okno narzędziowe *Toolbox*. To okno zawiera wiele obiektów kontrolnych, które można dodawać do tworzonego projektu. Wszystkie obiekty kontrolne zostały podzielone na listy. Aby wybrać obiekt z listy, należy kliknąć jej zakładkę i rozwinąć listę. W zależności od realizowanego projektu dostępne są różne zestawy narzędzi. Na przykład, gdy jedynym aktywnym oknem jest przeglądarka, w oknie narzędziowym jest dostępny tylko Wskaźnik (*Pointer*). Wskaźnik jest zawsze widocznym obiektem w oknie narzędziowym. Zestaw dostępnych narzędzi zależy od aktywnego okna, na przykład, gdy tworzona jest strona internetowa w języku HTML, w oknie narzędziowym znajduje się zakładka HTML (rysunek 2.11) z wieloma różnymi elementami przydatnymi podczas projektowania strony.




**Rysunek 2.11.** *Podczas tworzenia strony internetowej wszystkie potrzebne elementy znajdują się w oknie narzędziowym*

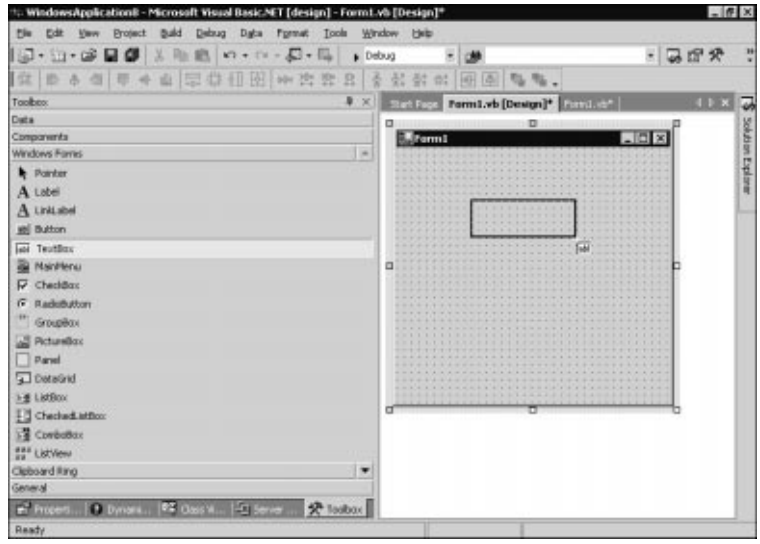
Aby dodać element do okna edycji, można:

- Kliknąć i przeciągnąć element w odpowiednie miejsce.
- Dwukrotnie kliknąć element okna narzędziowego, co spowoduje wygenerowanie odpowiedniej części kodu w obszarze edycji, w miejscu wskazanym przez znak kursora.

Aby elementy z okna narzędziowego umieścić w oknie dialogowym (formie), można:

- Kliknąć odpowiedni element i przeciągnąć w obszar okna graficznego. Podczas przeciągania widoczny jest zarys elementu.

- Kliknąć element okna narzędziowego dwa razy — element pojawi się w lewym górnym rogu okna graficznego.
- Wybrać pożądany element z listy, a następnie kliknąć w obszarze okna graficznego. Nowy element pojawia się w oknie graficznym w pożądanym miejscu (rysunek 2.12).

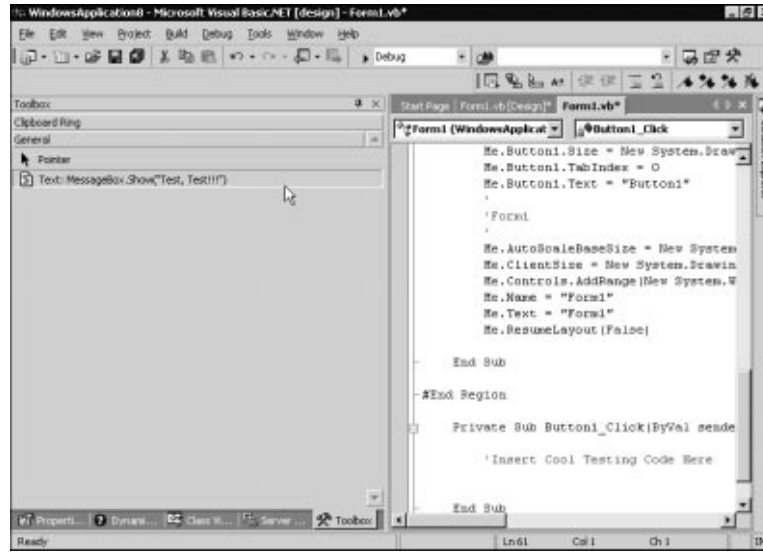



**Rysunek 2.12.** *Elementy można wrysować w formę po zaznaczeniu określonego elementu w oknie narzędziowym*

Metody umieszczania elementów okna narzędziowego na ekranie zostały opisane w sposób bardzo ogólny, co może sugerować, że użycie tych elementów jest kłopotliwe. Taki ogólny opis jest konieczny ze względu na naturę środowiska Visual Studio, które jest przeznaczone do projektowania aplikacji w wielu językach programowania. Różnorodność środowisk programowania w Visual Studio sprawia, że wyjaśnienie zasad zachowania się środowiska (np. umieszczania elementów z okna narzędziowego) jest trudne do opisania. Przedstawione metody umieszczania elementów okna narzędziowego można stosować w innych środowiskach programowania (np. Visual C++ .NET).

Konkretnym przykładem wykorzystania obiektów okna narzędziowego zajmiemy się w dalszej części rozdziału, podczas projektowania pierwszej aplikacji dla systemu Windows. Okno *Toolbox* pozwala na zdefiniowanie określonego fragmentu tekstu, który można wstawić w dowolne miejsce okna edycji. Jest to bardzo przydatna funkcja (przypominająca Schowek w programie Word), która pozwala na stworzenie bloków kodu źródłowego (np. rozbudowanych elementów witryny internetowej), gotowych do wykorzystania w polu edycji.

Aby utworzyć własny blok kodu źródłowego, wystarczy zaznaczyć odpowiedni fragment programu w polu edycji i przeciągnąć do okna narzędziowego. W oknie pojawi się nowy element gotowy do użycia. Aby umieścić ten fragment kodu w oknie edycji, należy nacisnąć nad nim lewy przycisk myszy i przytrzymać, następnie przeciągnąć fragment tekstu w pożądane miejsce i zwolnić przycisk. Przykład takiego elementu pokazano na rysunku 2.13.




**Rysunek 2.13.** *Kod źródłowy, HTML albo inny wycinek tekstowy można umieścić w oknie narzędziowym i korzystać z niego w taki sam sposób, w jaki korzysta się z innych obiektów kontrolnych*

Okno narzędziowe posiada wiele dodatkowych opcji, o których możesz się dowiedzieć korzystając z plików pomocy środowiska programistycznego. Te opcje to między innymi tworzenie dodatkowych zakładek wewnątrz okna narzędziowego, zmiana nazwy zakładek, czy zmiana zawartości list elementów. Dodatkowe opcje są udostępnione poprzez menu kontekstowe, otwierane poprzez kliknięcie prawym przyciskiem myszy w obszarze okna narzędziowego.

## Okno Command/Immediate

Dla wielu użytkowników skorzystanie z odpowiedniej komendy wiersza poleceń jest sposobem na szybkie wykonanie określonego zadania. Często występują takie sytuacje, w których korzystanie z okien dialogowych jest mniej wydajne niż korzystanie z wiersza poleceń, dlatego też w Visual Studio .NET wbudowano okno przypominające konsolę, które zawiera wiersz poleceń. W poprzednich wersjach Visual Basica zadanie konsoli spełniało okno *Immediate*, a w pakiecie Fox Pro — okno *Command*.

Okna *Immediate* i *Command* dodano do środowiska Visual Studio .NET pod jedną nazwą, *Command*. Aby otworzyć okno *Command*, należy wybrać polecenie *Command Window* z menu *View*, *Other Windows.* Powinno pojawić się na ekranie nowe okno, przypominające swym wyglądem konsolę (znaną ze starszych wersji systemów operacyjnych, np. DOS-u). W oknie znajduje się kursor poprzedzony znakiem zachęty. Znak zachęty sygnalizuje pracę okna w trybie *Command*. Za znakiem zachęty można wpisywać różne polecenia, które zostaną wykonane po naciśnięciu klawisza *Enter*. Aby przełączyć się do trybu *Immediate*, należy wpisać komendę `immed` i wcisnąć *Enter*. Na pasku tytułu okna pojawia się wówczas napis *Command Window — Immediate*, co oznacza, że zmieniono tryb pracy konsoli z *Command* na *Immediate*. Aby powrócić do trybu *Command*, wystarczy wpisać `>cmd` i wcisnąć Enter (znak zachęty > należy wprowadzić samodzielnie, gdyż w trybie *Immediate* nie pojawia się on automatycznie).

Konsola pracująca w trybie *Command* pozwala na wykonywanie różnych poleceń dotyczących środowiska programistycznego. Na przykład, gdy w oknie *Command* wpiszemy polecenie `File.NewProject`, to utworzymy nowy projekt. W środowisku programistycznym to samo zdarzenie można wykonać poprzez wybranie polecenia *Project* z menu *File*, *New*. Wiele zadań można wykonać szybciej za pomocą instrukcji wpisywanych w konsoli niż poprzez wybieranie odpowiednich poleceń interfejsu graficznego. Przygotowano wiele komend dostępnych z konsoli, większość z nich jest repliką zdarzeń generowanych za pomocą poleceń menu środowiska graficznego. Nazwy instrukcji wprowadzanych w konsoli są związane z nazwami ich odpowiedników środowiska graficznego. Na przykład, aby wybrać jedno z poleceń menu *File*, wprowadzamy za znakiem zachęty instrukcję `File`, a następnie polecenie tego menu, poprzedzone kropką (`.`). Oto lista kilku bardzo użytecznych instrukcji:

```
File.NewProject
File.SaveAll
Window.AutoHideAll
```

Tryb *Immediate* pozwala na natychmiastowe wykonywanie instrukcji języka programowania (w naszym przypadku — instrukcji Visual Basica). Można wprowadzić w tym oknie fragment kodu odpowiedzialny za wykonanie określonego zadania i w ten sposób szybko sprawdzić, czy rezultat jest zgodny z naszymi oczekiwaniami. Właściwości okna *Immediate* są wykorzystywane podczas pracy z wykorzystaniem punktów przerwań programu (punkty przerwań ustawiamy podczas wykrywania błędów — to zagadnienie opisano dokładniej w rozdziale szóstym, „Wykrywanie błędów w programach i zapobieganie ich występowaniu"). Spróbujmy napisać prosty program, w którym wykorzystamy punkt przerwania.

Aby bliżej poznać sposób korzystania z konsoli, wykonamy nasz prosty przykład w oknie *Command*. Należy upewnić się, że okno konsoli jest widoczne i ustawione w trybie *Command* (widoczny jest znak zachęty `>`). Wpiszmy instrukcję `File.NewProject` i wciśnijmy klawisz `Enter`. Podczas wprowadzania instrukcji pojawia się lista z dostępnymi instrukcjami, z której, po wprowadzeniu kilku liter, możemy wybrać odpowiednią komendę. Pojawi się okno dialogowe, w którym wybieramy folder *Visual Basic Projects*, a następnie typ tworzonej aplikacji — *Windows Application*. Aby zamknąć okno dialogowe, naciskamy przycisk *OK*, po czym pojawia się okno nowego projektu.

Nowy projekt stanowi pusta forma aplikacji, którą należy teraz oprogramować. Niewielka ilość kodu została wygenerowana przez kreatora projektu, resztę należy wpisać samodzielnie. Aby zobaczyć kod źródłowy aplikacji, należy wybrać polecenie *View Code* z menu kontekstowego formy (menu kontekstowe otwiera się poprzez kliknięcie prawym przyciskiem myszy w obszarze formy), co powoduje wyświetlenie nowego okna z kodem źródłowym.

Znajdźmy linię `Me.Text = "Form1"`. Teraz ustawimy punkt przerwania, który oznacza miejsce przerwania wykonywania programu. Są trzy sposoby na ustawienie punktu przerwania. Jednym z nich jest kliknięcie w obszarze szarego marginesu z lewej strony okna zawierającego kod źródłowy. Innym sposobem jest kliknięcie prawym przyciskiem myszy odpowiedniej linii i wybranie opcji *Insert Breakpoint* z menu kontekstowego. Punkt przerwania można wstawić również poprzez ustawienie kursora na

odpowiedniej linii i wciśnięcie klawisza F9. Ustawienie punktu przerwania powoduje zakreślenie danej linii i umieszczenie charakterystycznego punktu na marginesie okna z kodem źródłowym.

Uruchomienie programu zostanie przerwane, gdy tylko kompilator natrafi na linię oznaczoną punktem przerwania. Aby rozpocząć wykonywanie programu, należy wcisnąć klawisz *F5*, wybrać opcję *Start* z menu *Debug* albo wcisnąć przycisk *Start* (jego kształt jest podobny do przycisku „*Play*" w magnetowidzie), znajdujący się na pasku narzędzi (rysunek 2.14). Najszybciej wybiera się funkcje za pomocą klawiatury, ale sposób wybierania opcji zależy tylko od programisty — każdy sposób jest dobry, jeśli prowadzi do celu.


**Rysunek 2.14.** *Pasek narzędzi Debug zawiera przyciski pozwalające na uruchomienie i zatrzymanie wykonywania programu (pasek ten jest stylizowany na panel magnetowidu)*

Po wciśnięciu klawisza *F5* wykonywanie programu zostaje przerwane w miejscu oznaczonym punktem przerwania. Aplikacja jest w trybie przerwania, co jest oznaczone pojawieniem się napisu `[break]`na pasku tytułu środowiska programistycznego. W punkcie oznaczającym przerwanie pojawia się żółta strzałka, a instrukcja, która została przerwana, jest zakreślona żółtym kolorem. Teraz należy przejść w konsoli do trybu *Immediate* i wypróbować jego możliwości.

Jeśli okno *Command* jest niewidoczne, należy je otworzyć. Następnie należy wpisać komendę `immed`, aby przełączyć się w tryb *Immediate*. Teraz można wprowadzać dowolne instrukcje języka Visual Basic, które natychmiast zostaną wykonane. Spróbujmy wykonać następujące działania:

```
?Me.Width
300
?3+5
8
?3=5
False
```

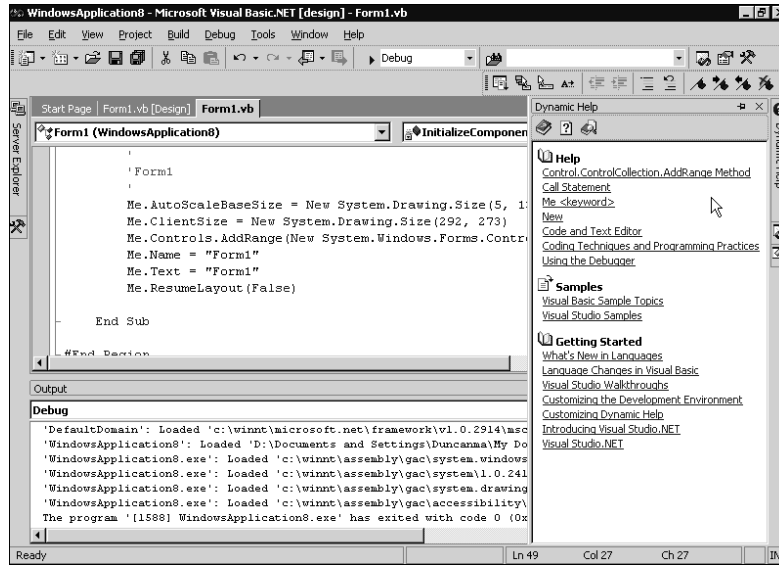
W trybie Immediate nie można poruszać się pomiędzy poszczególnymi liniami za pomocą klawiszy strzałek. Strzałki (góra/dół) służą do przewijania wcześniej wpisanych instrukcji. Aby skorzystać z tekstu wygenerowanego w oknie Immediate, należy zaznaczyć odpowiedni blok, skopiować (CTRL + C) i przenieść w odpowiednie miejsce (np. do dokumentu w edytorze Word). Można ustawić kursor (za pomocą myszy) na odpowiedniej linii i nacisnąć Enter, wówczas ta konkretna linia zostanie wypisana jako ostatnia wprowadzona instrukcja w konsoli. Po ponownym wciśnięciu klawisza Enter ta instrukcja zostanie wykonana.

Należy zauważyć, że znak zapytania ? jest skrótem dobrze znanej z języka Basic instrukcji `Print` (wypisz na ekranie). Gdy nie napiszemy tej instrukcji przed naszym wyrażeniem (np. `3+5`), nie zostanie ono wykonane (pojawi się błąd składni — *Syntax Error*). Błąd nie wystąpi, gdy napiszemy np. `Me.Width = Me.Width*2`, bo jest to prawidłowa instrukcja Visual Basica (nie zobaczymy efektu zadziałania tej instrukcji).

Gdy teraz naciśniemy *F5*, program zostanie wykonany do końca, a na ekranie pojawi się okno dialogowe. To okno ma teraz taką szerokość, jaką zadeklarowano instrukcją `Me.Width` (w naszym przypadku `Me.Width = 112`). Łatwo więc zauważyć, że można modyfikować aplikację za pomocą okna *Command/Immediate.*

## Pomoc kontekstowa

System pomocy znajduje się w zakładce okna właściwości (*Properties*). W oknie pomocy mamy możliwość uzyskania informacji dotyczącej każdej instrukcji Visual Basica i sposobu jej wykorzystania oraz innych informacji dotyczących programowania w tym języku. Okno pomocy pojawia się po naciśnięciu klawisza *F1* albo po wybraniu tematu pomocy z menu *Help*. Okno pomocy zawiera odnośniki do tematów powiązanych z zawartością aktywnego okna środowiska programistycznego, w którym znajduje się kursor. W oknie pomocy (rysunek 2.15) znajdują się również odnośniki do dokumentacji platformy .NET. W oknie pomocy można wpisać w odpowiednim polu hasło i znaleźć informacje na temat instrukcji języka Visual Basic albo technik programowania. W dokumentacji znajduje się cały rozdział poświęcony programowaniu, są tam nawet przykładowe aplikacje.

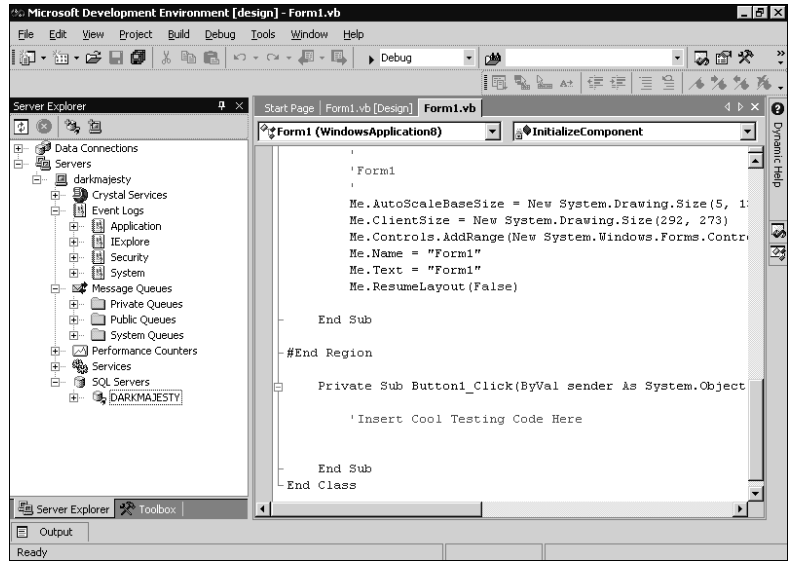



**Rysunek 2.15.** *W oknie pomocy można uzyskać informacje dotyczące aktualnie wykonywanego zadania*

## Okno Server Explorer

W oknie *Server Explorer* znajduje się lista serwerów i baz danych (rysunek 2.16). Lista baz danych zawiera zbiór wszystkich połączeń tworzonego projektu z serwerami

baz danych. Poprzez to okno można uzyskać dokładne informacje dotyczące baz danych. Można np. przeglądać informacje wpisane w tabelach, procedury zapamiętane i inne użyteczne dane dotyczące dostępnych baz danych.




**Rysunek 2.16.** *Okno Server Explorer zawiera informacje dotyczące zasobów serwerów lokalnych i innych komputerów*

Lista serwerów zawiera informacje dotyczące dostępnych komputerów, z którymi można się połączyć. Ponadto, za pomocą okna *Server Explorer* można w prosty sposób udostępnić zasoby tych komputerów tworzonej aplikacji (są to np. liczniki wydajności, rejestry, informacje o kolejkach, itp.).

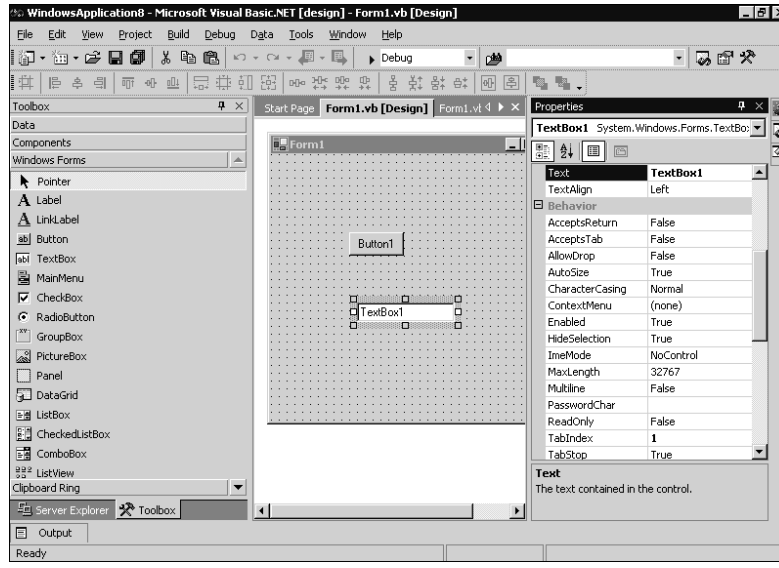
W rozdziale trzynastym „Server Explorer" szczegółowo opisano to narzędzie.

## Okno Properties

Okno właściwości (*Properties*) zawiera informacje na temat każdego obiektu w projekcie. W Visual Studio każdy element (projekt, rozwiązanie, forma, klasa, i inne) posiada zestaw definiowanych właściwości, które go opisują. Przykładem właściwości jest nazwa projektu. W trakcie projektowania aplikacji środowisko automatycznie ustawia właściwości domyślne każdego elementu. Aby przejrzeć właściwości obiektu i dokonać ewentualnych zmian (np. zmienić nazwę projektu), należy skorzystać z okna *Properties* (rysunek 2. 17). Niektóre pola w oknie właściwości nie mogą zostać zmienione. Te pola pełnią rolę zakładek, dwukrotne ich kliknięcie powoduje rozwinięcie listy właściwości znajdujących się w zakładce.

## Solution Explorer

W tym oknie znajduje się wykaz aktualnie otwartych projektów. Okno to przypomina trochę Eksplorator Windows, znajdujący się w systemie operacyjnym Windows. *Solution Explorer* służy do zarządzania projektami i plikami wchodzącymi w skład każdego z projektów. Podczas tworzenia aplikacji programista może wybrać jedną z trzech




**Rysunek 2.17.** *W oknie Właściwości są pokazane wszystkie atrybuty zaznaczonego obiektu. Na rysunku zaznaczonym obiektem jest pole tekstowe, którego atrybuty pokazuje okno Właściwości*

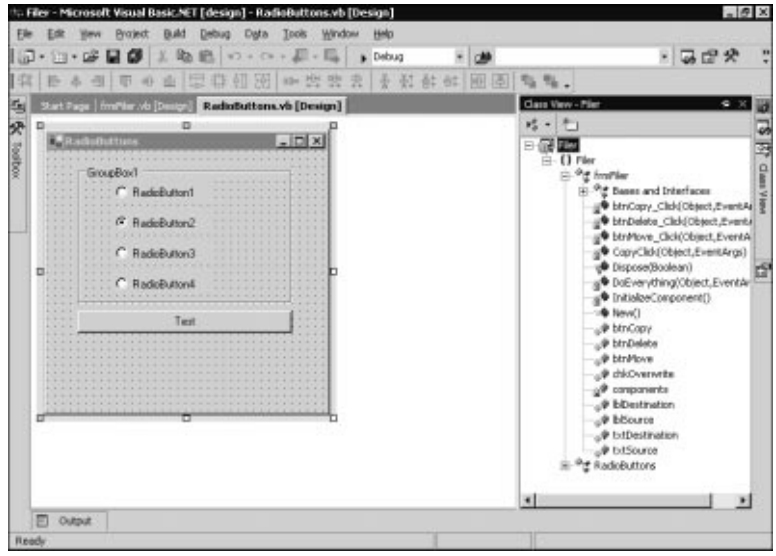
metod zarządzania plikami należącymi do projektu. W środowisku Visual Studio .NET tworzony program można umieszczać w rozwiązaniu (*Solution)*, projekcie (*Project*) lub pojedynczym pliku. Rozwiązanie jest zbiorem projektów, a w projektach umieszczone są pliki. Projekty pełnią rolę tworzonych aplikacji zawierających poszczególne pliki, klasy, komponenty. Okno Eksploratora pozwala na wyświetlenie wszystkich rozwiązań, projektów i plików pod warunkiem, że są one otwarte. Za pomocą Eksploratora można wykonać wiele użytecznych funkcji, na przykład:

- dodać nowe pliki do projektu (należy kliknąć prawym przyciskiem myszy i z menu kontekstowego wybrać polecenie *Add*);
- usunąć pliki z projektu (należy kliknąć prawym przyciskiem myszy i z menu kontekstowego wybrać polecenie *Remove*);
- dodać lub usunąć projekty zawarte w grupie rozwiązań (należy kliknąć prawym przyciskiem myszy rozwiązanie, aby dodać projekt, albo kliknąć prawym przyciskiem myszy projekt, by go usunąć z rozwiązania).

## Class View

W oknie *Class View* są umieszczone wszystkie klasy, funkcje i zmienne należące do tworzonego projektu. *Class View* (rysunek 2.18) pokazuje hierarchiczną strukturę projektu, czyli uporządkowany wykaz wszystkich klas, funkcji i zmiennych. W tym oknie łatwo sprawdzić, które funkcje i zmienne należą do określonej klasy, a także które klasy są pochodnymi danej klasy.

Aby sprawdzić, w jaki sposób zbudowany jest projekt, wystarczy po kolei klikać nazwy poszczególnych klas. Spowoduje to rozwinięcie listy z funkcjami i zmiennymi należącymi do tych klas oraz nazwami klas potomnych. Aby wyświetlić okno z kodem źródłowym któregokolwiek z obiektów znajdujących się w *Class View*, wystarczy dwukrotnie kliknąć nazwę określonego obiektu. Widok klas pozwala na szybkie




**Rysunek 2.18.** *Okno widoku klas przedstawia hierarchiczną strukturę projektu i zapewnia dostęp do kodu źródłowego każdego z obiektów*

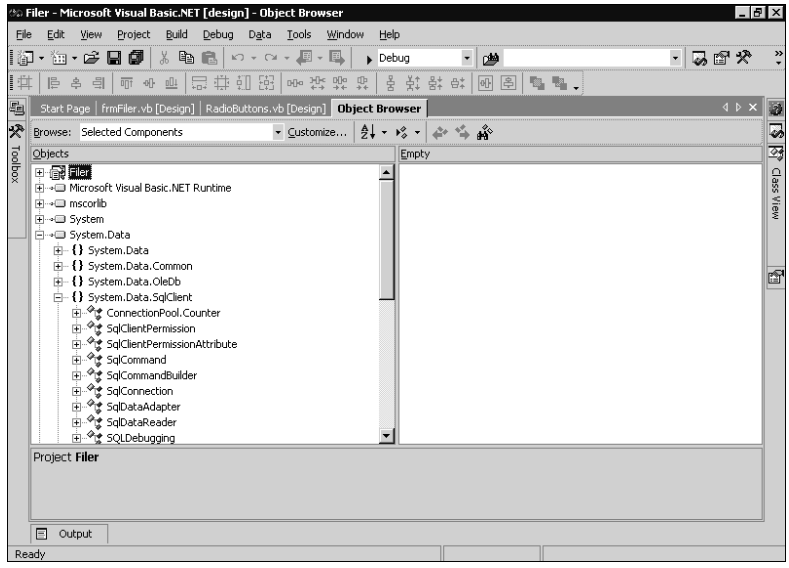
zorientowanie się w budowie aplikacji, a także na szybki dostęp do odpowiednich klas, funkcji i zmiennych. Za pomocą widoku klas można szybko sprawdzić, gdzie znajduje się definicja (lub deklaracja) określonego obiektu. Często za pomocą widoku klas programista jest przełączany do okna *Object Browser*, opisanego w następnym podpunkcie. To okno jest niezwykle pomocne podczas projektowania aplikacji w Visual Basic .NET.

## Object Browser

Programowanie w środowisku .NET opiera się na wykorzystywaniu obiektów (obiektów tworzonych przez programistę, obiektów .NET Framework, itp.). Każdy z obiektów zawiera określone metody i posiada swoje właściwości. Aby sprawdzić, jakie funkcje są dostępne w obiekcie i jakie właściwości posiada, projektanci środowiska Visual Studio .NET stworzyli *Object Browser* (przeglądarkę obiektów). Przeglądarka pokazuje nie tylko obiekty należące do określonego projektu, ale także pozwala na wyświetlenie właściwości obiektów nie należących do tworzonej aplikacji. *Object Browser* jest bardzo przydatnym narzędziem do wyświetlania informacji dotyczących obiektów .NET Framework (np. właściwości lub funkcji) lub innych bibliotek klas. Na rysunku 2.19 przedstawiono przykład wykorzystania przeglądarki obiektów, która wyświetla zawartość biblioteki `System.Data` z informacjami dotyczącymi wykorzystania jej elementów.

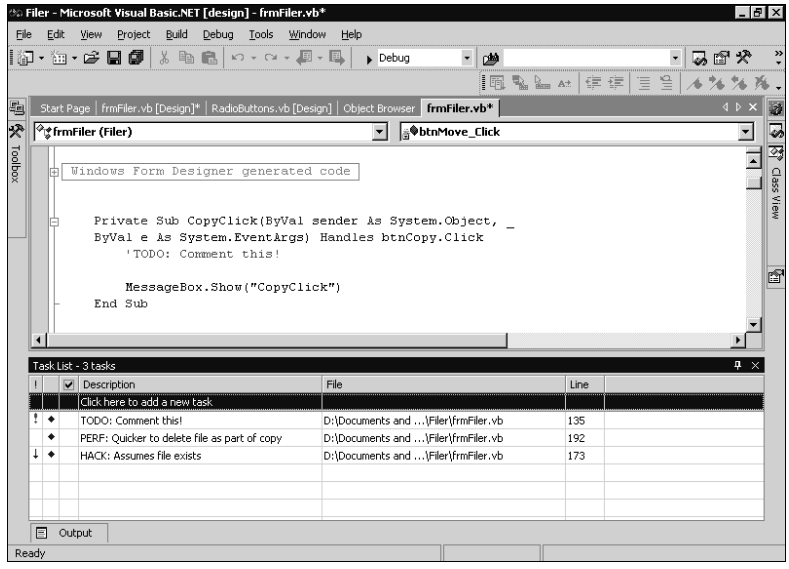
## Task List

Podczas projektowania różnego rodzaju aplikacji istnieje wiele zadań do wykonania. Poszczególne funkcje programu wykonujące określone zadania mogą korzystać z innych funkcji. Aby zachować porządek pisanego programu, programiści używają komentarzy, w których opisują zadanie wykonywane w danej części kodu źródłowego. W komentarzach często używa się charakterystycznych słów (np. `TODO` lub `BUG`), które




**Rysunek 2.19.** *Przeglądarka obiektów udostępnia informacje dotyczące klas .NET Framework lub innych bibliotek klas*

ułatwiają dojście do odpowiednich bloków kodu. W środowisku Visual Studio .NET wprowadzono standardową listę słów kluczowych zadań, które powinny znajdować się w komentarzu. Gdy dodajemy jakiś obiekt automatycznie do formy, środowisko programistyczne generuje odpowiedni kod źródłowy danego obiektu wraz z komentarzem zawierającym słowa kluczowe standardowej listy zadań. Każdy komentarz zawierający słowo kluczowe jest wyświetlany w oknie *Task List* (lista zadań). Przykład listy zadań pokazano na rysunku 2.20.




**Rysunek 2.20.** *Każdy komentarz, poprzedzony odpowiednim słowem kluczowym, zostanie rozpoznany i dołączony do listy zadań*

Oprócz standardowych, można używać własnych słów kluczowych. Aby zdefiniować własne słowa kluczowe, należy użyć z głównego menu opcji *Tools*, *Options, Task List*. W lewej części ekranu znajduje się lista z istniejącymi słowami kluczowymi, do której można dopisać własne, a następnie kliknąć przycisk *Add*. Po tej prostej operacji można pisać komentarze zawierające nowe słowa kluczowe, które zostaną rozpoznane i wpisane w oknie *Task List*.

Dwukrotne kliknięcie jednego z zadań znajdującego się na liście powoduje wyświetlenie okna z kodem źródłowym i ustawienie kursora w miejscu komentarza. Jest to dobry sposób na szybkie przemieszczanie się pomiędzy poszczególnymi funkcjami podczas sprawdzania poprawności składni kodu. Do listy zadań można dodać linie nie zawierające komentarza (tak zwane *skróty*).

Aby dodać do listy zadań skrót, należy przesunąć wskaźnik myszy na odpowiednią linię, nacisnąć prawy przycisk i wybrać polecenie *Add Task List Shortcut*. Odpowiednia linia zostanie wówczas dodana do listy zadań, a na marginesie okna z kodem źródłowym pojawi się symbol niebieskiej strzałki. Podobny symbol (ale w czarnym kolorze) pojawi się na liście zadań, przy dodanej linii. Po dodaniu linii do listy zadań, można się do niej szybko odwołać poprzez dwukrotne kliknięcie. Aby usunąć linię z listy zadań, należy kliknąć ją prawym przyciskiem myszy, a następnie z menu kontekstowego wybrać opcję *Delete*.

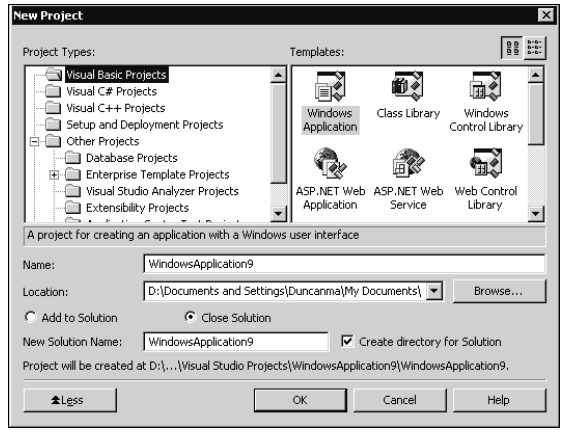
Okno listy zadań umożliwia dodawanie tak zwanych *zadań definiowanych przez użytkownika*. Są to zadania, które nie mają nic wspólnego z kodem źródłowym i stanowią swego rodzaju notatkę dla programisty. Te zadania pełnią tylko funkcję informacyjną, podobnie jak zadania definiowane w programie Microsoft Outlook. W zadaniach definiowanych przez użytkownika można wykorzystać tylko pola opisu (*Description*) i priorytetu zadania (*Priority*). Mamy do wyboru priorytet niski, normalny i wysoki (*Low, Normal, High*).

# Rozwiązania i projekty

Podczas tworzenia oprogramowania można skorzystać z kilku poziomów grupowania poszczególnych składników aplikacji. Najwyższym poziomem jest rozwiązanie (*Solution*), które zawiera składniki tworzonej aplikacji, czyli projekty. Przed rozpoczęciem programowania w środowisku Visual Studio, należy umieścić wewnątrz rozwiązania co najmniej jeden projekt. Niniejszy podpunkt poprowadzi nas przez proces tworzenia nowych projektów, wpisywania kodu źródłowego i korzystania z istniejących projektów i plików.

## Tworzenie nowego projektu

Istnieje kilka sposobów tworzenia nowego projektu. Najczęściej korzysta się z polecenia menu (*File, New Project)*. Wybranie tego polecenia powoduje wyświetlenie okna dialogowego (rysunek 2.21), w którym mamy do wyboru kilka typów projektów. Ponieważ Visual Studio umożliwia tworzenie aplikacji w wielu językach, w oknie dialogowym znajdują się opcje dotyczące wszystkich zainstalowanych języków programowania. My będziemy tworzyć aplikację korzystając z opcji zawartych w folderze *Visual Basic Projects.*




**Rysunek 2.21.** *Okno dialogowe New Project pozwala na wybór typu tworzonego projektu wraz z opcjami językowymi*

Aby utworzyć aplikację, którą będzie można uruchomić na lokalnym komputerze w systemie Windows (za pomocą graficznego interfejsu użytkownika, okien dialogowych i innych elementów), należy z dostępnych typów projektów wybrać ikonę aplikacji dla Windows (*Windows Application*). Ponadto trzeba wpisać nazwę tworzonej aplikacji i w razie potrzeby zmienić ścieżkę dostępu do folderu wykorzystywanego przez projekt. Po naciśnięciu przycisku *OK* kreator aplikacji środowiska Visual Studio utworzy nowy projekt. Dobrym nawykiem jest stosowanie poprawnego nazewnictwa tworzonych aplikacji — posiadanie w komputerze projektów o nazwach *WindowsApplication1*, *WindowsApplication2...*, może przysporzyć trudności w odnalezieniu właściwego projektu.

## Otwieranie istniejącego projektu

Podczas zamykania środowiska Visual Studio pojawia się okno dialogowe z zapytaniem, czy zapisać zmiany w projekcie, a następnie środowisko zamyka się automatycznie. Chcąc otworzyć wcześniej utworzony projekt, powinniśmy zacząć od otwarcia samego środowiska, a następnie skorzystać z jednej z kilku możliwości oferowanych przez Visual Studio. Jedną z nich jest otwarcie projektu poprzez opcję głównego menu (*File, Open, Project* albo przez opcję *Recent Files* w dolnej części menu *Files*). Innym sposobem jest otwarcie projektu ze strony *Get Started*, gdzie znajdują się odnośniki do istniejących projektów wraz z datami ich powstania. Otwarcie nowego projektu powoduje automatyczne zamknięcie wcześniej tworzonych projektów, chyba że otworzymy go za pomocą opcji *File, Add Project*, która powoduje dodanie projektu do otwartego rozwiązania (grupy projektów).

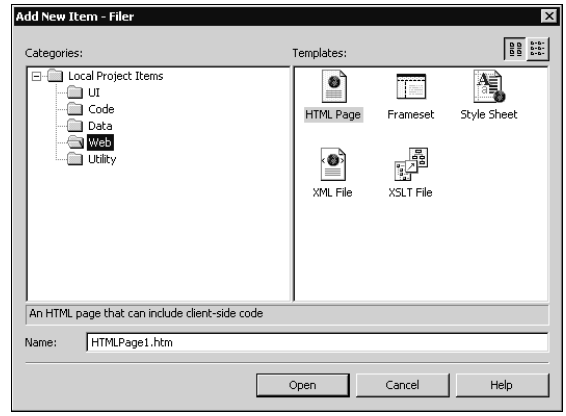
# Pliki

Koncepcję istnienia rozwiązań i projektów stworzono ze względów organizacyjnych — właściwy kod aplikacji jest rozbity pomiędzy wiele plików, którymi trzeba zarządzać. Podczas tworzenia nowego projektu niektóre pliki są tworzone automatycznie, np. plik nowej formy (`Form1.vb`), gdy jest tworzona aplikacja pod Windows, albo pliki nowego modułu (`Module1.vb`), gdy jest tworzona aplikacja uruchamiana z konsoli.

Te pliki istnieją niezależnie od projektów i mogą być współdzielone przez kilka aplikacji, gdy zaistnieje taka potrzeba.

## Dodawanie nowych plików do projektu

Oprócz plików generowanych przez środowisko Visual Studio, można dołączać do projektu własne moduły, klasy, biblioteki, formy i inne pliki z kodem źródłowym. Własne pliki można dołączać albo przez menu *Project*, albo za pomocą menu kontekstowego w oknie Eksploratora rozwiązań (należy ustawić wskaźnik myszy na nazwie projektu i kliknąć prawym przyciskiem, a następnie wybrać odpowiednią opcję menu *Add*). W zależności od typu pliku, jaki chcemy dołączyć, wybieramy odpowiednią opcję menu *Project* lub kontekstowego menu *Add*. Wybranie opcji *Add New Item* powoduje wyświetlenie okna dialogowego (rysunek 2.22) z typami plików, które można dodać do projektu.

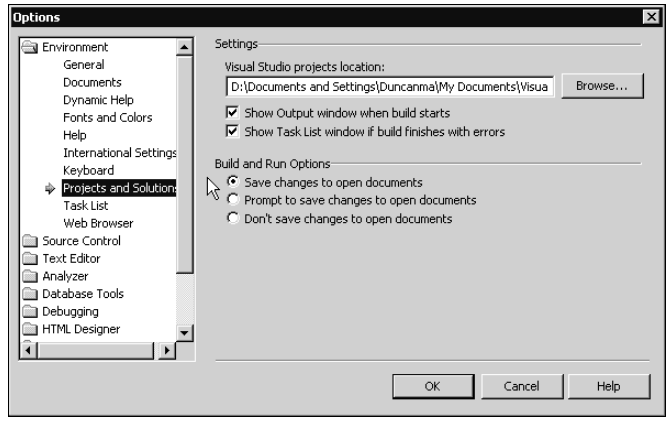



**Rysunek 2.22.** *Okno Add New Item posiada kilka typów plików, które można umieścić w projekcie. Wyglądem przypomina okno New Project*

## Zapisywanie zmian

Istnienie wielu grup elementów aplikacji (rozwiązań, projektów, plików) może stanowić pewien problem podczas zamykania środowiska. Należy wiedzieć, w jaki sposób zapisywać zmiany projektów, które mogą zawierać wiele plików. Visual Studio posiada dwie opcje zapisu zmian: *Zapisz* (*Save*) i *Zapisz wszystko* (*Save All*). Te opcje znajdują się w głównym menu *File*. Opcja *Zapisz* umożliwia zapisanie zmian pliku zaznaczonego w Eksploratorze rozwiązań, a opcja *Zapisz wszystko* powoduje zapisanie zmian we wszystkich plikach należących do aplikacji.

Jeśli chcemy mieć pewność, że wszystkie zmiany w tworzonych projektach będą zapisane, możemy w oknie *Options* wybrać folder *Environment* i zakładkę *Project and Solutions* (rysunek 2.23), a następnie wybrać jedną z trzech opcji *Build and Run*. Wybranie opcji *Save changes to open documents* spowoduje zapisywanie wszystkich zmian przed uruchomieniem aplikacji. Ustawienie tej opcji jest bardzo ważne, bo w razie zawieszenia się środowiska podczas kompilacji lub testowania programu nie stracimy zmian dokonanych przed próbą uruchomienia projektu.




**Rysunek 2.23.** *Należy zawsze sprawdzić ustawienie opcji Save, aby zapewnić zachowanie wszystkich dokonanych w projekcie zmian na wypadek zawieszenia się systemu*

# Tworzenie pierwszej aplikacji dla systemu Windows

Po wprowadzeniu do środowiska programistycznego Visual Basic, czas na stworzenie pierwszej aplikacji. Ten przykład ma na celu pokazanie możliwości środowiska programistycznego, dlatego utworzymy bardzo prostą aplikację. Nasza aplikacja będzie uruchamiana w systemie Windows i umożliwi wprowadzenie przez użytkownika kilku liczb, które zostaną do siebie dodane, a na ekranie pojawi się wynik tej operacji.

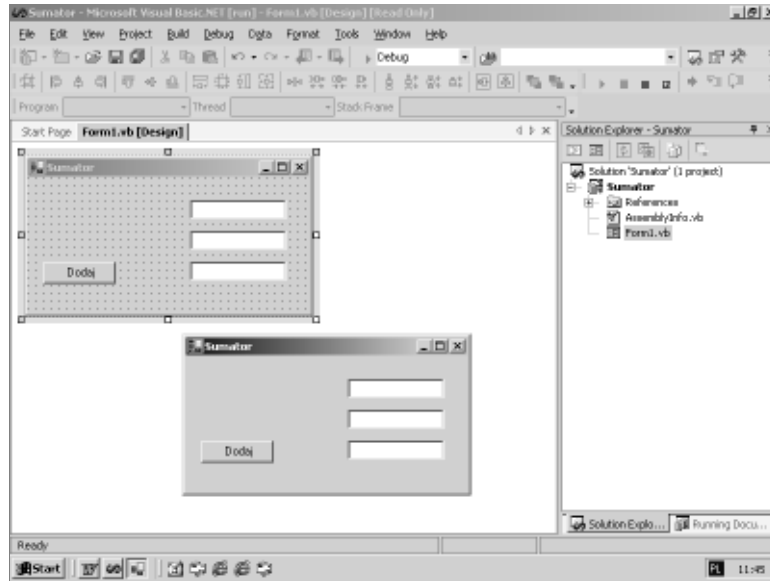
## Tworzenie projektu

Najpierw należy utworzyć nowy projekt za pomocą opcji *Project* z menu *File*, *New*. W oknie dialogowym *New Project* zaznaczamy ikonę *Windows Application* i wpisujemy nazwę projektu `Adder`. Nowy projekt zawiera formę, a Visual Studio automatycznie utworzy folder *Solution* o nazwie *Adder*. Po wprowadzeniu odpowiednich danych należy nacisnąć *OK*.

## Interfejs użytkownika

W projekcie należy utworzyć graficzny interfejs użytkownika w postaci okna dialogowego, w którym będą trzy pola tekstowe i jeden przycisk (rysunek 2.24). Aby zobaczyć obszar tworzonego okna dialogowego (czyli formy), należy dwa razy kliknąć ikonę *Form1.vb* w oknie *Solution Explorer*. Z okna narzędziowego (*Toolbox*) będziemy wybierać potrzebne elementy i umieszczać w formie. Potrzebne elementy to pole tekstowe *TextBox* i przycisk *Button*. Aby je umieścić w formie, należy ustawić wskaźnik myszy na pożądanym obiekcie w oknie narzędziowym, przeciągnąć obiekt w obszar formy i zwolnić przycisk. W ten sposób umieścimy wszystkie elementy w oknie dialogowym (jak pokazano na rysunku 2.24). Po umieszczeniu elementów w obszarze

formy, należy odpowiednio zmienić ich rozmiary. Aby nabrać wprawy w ustawianiu elementów i doborze ich rozmiarów, można poświęcić chwilę na ustawianie elementów w różnej konfiguracji. Gdy wszystkie potrzebne elementy są już w obszarze formy, można ustawić ich właściwości.




**Rysunek 2.24.** *Widok gotowego interfejsu użytkownika, który składa się z trzech pól tekstowych i jednego przycisku*

Zaznaczmy pole tekstowe (pierwsze od góry) i otwórzmy okno właściwości (*Properties*) za pomocą klawisza F4 lub przez opcję menu (*View*, *Properties Window*). Okno zawiera listę ustawień zaznaczonego pola tekstowego, ale nas interesują tylko dwie właściwości:

- *Text* (w zakładce *Appearance*) reprezentuje zawartość pola tekstowego, która pojawia się w trakcie uruchomienia aplikacji. Zawartość tego pola należy usunąć, aby po uruchomieniu aplikacji pole tekstowe było puste.
- *(Name)* (w zakładce *Design*). W tym polu wpisujemy nazwę pola tekstowego. Ta nazwa jest identyfikatorem pola. Za każdym razem, kiedy chcemy odwołać się do pola tekstowego w kodzie źródłowym (np. pobrać liczbę wpisywaną w pole przez użytkownika), posługujemy się właśnie tą nazwą. Wpiszmy jakąś konkretną nazwę, np. `txtFirstValue`.

Powyższe ustawienia należy zmienić w pozostałych dwóch polach tekstowych poprzez usunięcie tekstu z właściwości *Text* i zmianę nazwy pól odpowiednio na `txtSecondValue` i `txtResult`.

Teraz należy zaznaczyć przycisk i zmienić jego właściwości. W polu *Text* należy wpisać `Dodaj` (ten napis pojawi się na przycisku i będzie widoczny podczas pracy aplikacji), a w polu *(Name)* — `btnAdd`. Tak jak w przypadku pól tekstowych, nazwa przycisku wpisana w pole *(Name)*okna właściwości stanowi identyfikator tego obiektu.

Ostatnią zmianą, jakiej dokonamy w naszym pierwszym projekcie, jest zmiana nazwy formy. Należy kliknąć formę tak, aby nie zaznaczyć żadnego z jej obiektów — wówczas pojawi się ramka wokół projektowanego okna dialogowego. Następnie należy znaleźć w oknie właściwości pole *Text*, które znajduje się w zakładce *Appearance*. Tekst wpisany w to pole będzie widoczny na pasku tytułowym tworzonego okna dialogowego. W naszej aplikacji wpiszemy nazwę *Sumator*.

# Uruchomienie aplikacji

Do tej pory nie wpisaliśmy ani jednej linijki kodu, ale możemy uruchomić naszą aplikację. Środowisko Visual Studio umożliwia uruchomienie aplikacji wewnątrz środowiska, bez tworzenia plików wykonywalnych (np. plików z rozszerzeniem *.exe*). Można w szybki sposób uruchamiać tworzony projekt i kontrolować jego zachowanie za pomocą wielu narzędzi diagnostycznych, które zostaną szczegółowo omówione w rozdziale szóstym. Należy podkreślić różnicę pomiędzy uruchamianiem aplikacji wewnątrz środowiska a tworzeniem plików wykonywalnych, które mogą być uruchamiane w systemie Windows. W następnym rozdziale nauczymy się tworzyć aplikacje uruchamiane bezpośrednio w systemie operacyjnym.

Aby uruchomić aplikację, należy wcisnąć klawisz *F5* albo wybrać polecenie *Debug*, *Start* z głównego menu. Można również wcisnąć przycisk umieszczony w pasku narzędzi, przypominający swym kształtem klawisz „Start" w magnetowidzie. W oknie środowiska pojawi się okno dialogowe naszej aplikacji, które można przemieszczać, zmieniać jego rozmiar, wpisywać tekst w pola tekstowe. Pomimo to, że nie wpisaliśmy ani jednej linijki kodu, nasza aplikacja może być uruchamiana, a w dodatku możemy wykonać kilka czynności związanych z oknem dialogowym. Tę funkcjonalność umożliwia .NET Framework i środowisko programowe, które pozwala na tworzenie interfejsu użytkownika w sposób graficzny i generuje samoczynnie odpowiedni kod obsługujący okno dialogowe wraz z umieszczonymi w nim elementami. Aby umożliwić uruchomienie aplikacji bez udziału środowiska programistycznego, należy „zbudować" (*Build*) aplikację, czyli stworzyć plik wykonywalny.

# Tworzenie pliku wykonywalnego

Operacja tworzenia pliku wykonywalnego dla środowiska Windows sprowadza się do skompilowania projektu, co spowoduje wygenerowanie pliku wykonywalnego (z rozszerzeniem *.exe*). Plik wykonywalny tworzymy po to, aby można było uruchomić naszą aplikację na jakimkolwiek komputerze z zainstalowanym systemem operacyjnym Windows.

## Ustawienia w projekcie

Aby utworzyć plik wykonywalny naszego projektu, należy wybrać polecenie *Build* z menu *Build*. Wykonanie tego polecenia spowoduje utworzenie pliku wykonywalnego (w naszym przypadku będzie to plik o nazwie *Sumator.exe*), który zostanie umieszczony w katalogu *My Documents\Visual Studio Projects\Sumator\bin\Adder.exe*, podczas gdy projekt znajduje się w katalogu *My Documents\Visual Studio Projects\Adder*.

Polecenie *Build* ma pewne właściwości, które można zmienić w zależności od potrzeb. Aby zobaczyć domyślne ustawienia tej komendy, należy kliknąć prawym przyciskiem myszy nazwę projektu w oknie Eksploratora rozwiązań i wybrać z menu kontekstowego opcję *Properties*. Otworzy się okno z właściwościami naszego projektu, podzielonymi na kilka grup. W grupie *Common Properties/General* znajdują się pola:

- *Assembly name*. W tym polu umieszczona jest nazwa, jaką będzie miał utworzony plik wykonywalny po kompilacji. W naszym przypadku to pole zawiera nazwę *Adder*. Jeśli zmienimy tę nazwę na np. *MyAdder*, to po kompilacji zostanie utworzony plik o takiej nazwie.
- *Output type*. Jest to lista typów plików możliwych do otrzymania w trakcie kompilacji. Gdy wybierzemy *Windows Application* lub *Console Application*, otrzymamy plik wykonywalny (z rozszerzeniem *.exe*). Jeśli zostanie wybrana opcja *Library*, to w wyniku kompilacji uzyskamy plik biblioteczny (z rozszerzeniem *.dll*).
- *Startup object.* Jest to lista wyboru zawierająca wszystkie części projektu, umożliwiająca okreslenie, która z nich ma być uruchomiona w pierwszej kolejności po uruchomieniu programu. W naszym przypadku jest to `Sumator.Form1`. Należy zauważyć, że jeśli zmienimy opcję *Windows Application* na liście *Output type* na inną, to może to spowodować zmianę na liście *Startup object.* Ponowne ustawienie opcji *Windows Application* na liście *Output type* nie przywróci poprzedniej opcji na liście *Startup object*; trzeba ją ustawić samodzielnie.

Jeśli w oknie *Properties* naszego projektu dokonamy zmian, które mogą być nieprawidłowe, możemy anulować je za pomocą przycisku *Cancel*.

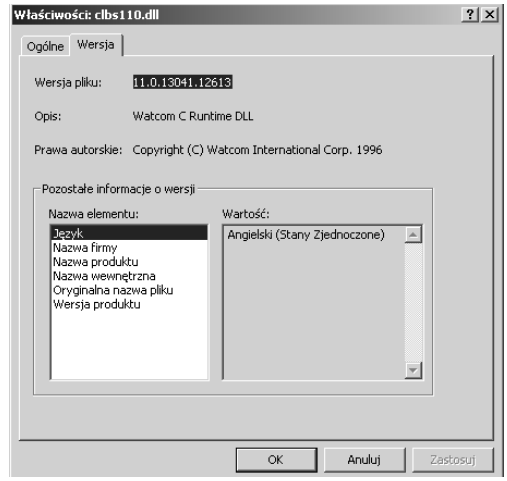
Wszystkie ustawienia w zakładce *Common Properties/Version* są przechowywane w plikach tworzonych podczas kompilacji. Te informacje są widoczne, gdy w systemie Windows klikniemy prawym przyciskiem myszy plik i wybierzemy z menu kontekstowego zakładkę *Właściwości/Wersja* (rysunek 2.25). Informacje, które pojawią się w tej zakładce mogą być ważne dla użytkowników naszych aplikacji. Szczególnie ważne jest wpisanie szczegółowych informacji dotyczących tworzonych plików bibliotecznych.

W zakładce *Common Properties/Build* można wybrać ikonę tworzonego pliku wykonywalnego. Z listy rozwijanej *Application Icon* możemy wybrać ikonę naszego pliku wykonywalnego. Zamiast domyślnej (*Default icon*), możemy wybrać dowolną ikonę umieszczoną w pliku z rozszerzeniem *.ico*.

W pozostałych zakładkach omawianego okna znajdują się inne informacje i ustawienia dotyczące tworzonej aplikacji.

## Ustawienia konfiguracji kompilatora

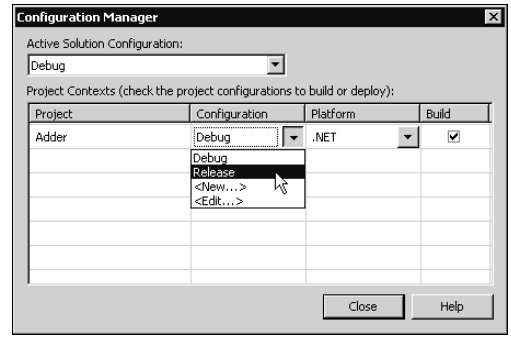
W górnej części okna właściwości projektu znajduje się lista *Configuration*. Domyślnie na liście *Configuration* mamy do wyboru dwa ustawienia: *Debug* i *Release*, które służą do testowania aplikacji (*Debug*) lub tworzenia wersji dostarczanej użytkownikom




**Rysunek 2.25.** *Informacje dotyczące wersji aplikacji zostały dodane jeszcze przed kompilacją*

(*Release*). Środowisko Visual Studio pozwala na utworzenie wielu grup ustawień konfiguracji dla jednego projektu. Konfiguracje rozwiązań (*Solution Configurations*) służą do wyboru odpowiedniego schematu tworzenia aplikacji.

Za pomocą menedżera konfiguracji (*Configuration Manager*) można stworzyć własne grupy konfiguracji, a nawet usunąć istniejące. Okno menedżera konfiguracji pokazano na rysunku 2.26. Na razie wystarczy wiedzieć, że opcja *Release* służy do utworzenia plików aplikacji, a opcja *Debug* jest przeznaczona do testowania tworzonego projektu.




**Rysunek 2.26.** *Okno Configuration Manager pozwala na utworzenie różnych konfiguracji do wykonania różnych zadań (testowania, tworzenia pliku wykonywalnego, itp.)*

## Tworzenie pliku wykonywalnego

Najlepszym sposobem na poznanie środowiska programistycznego i jego możliwości jest wykonanie prostej aplikacji i utworzenie pliku wykonywalnego. Wystarczy wybrać polecenie *Build* z menu *Build*, a kompilator utworzy nam plik wykonywalny. W trakcie kompilacji w dolnej części ekranu pojawia się okno *Output*, w którym wyświetlane są informacje na temat przebiegu kompilacji i ewentualnych błędów powstałych podczas kompilacji. Jeśli kompilacja przebiegła prawidłowo, możemy zminimalizować okno środowiska Visual Studio i spróbować uruchomić utworzony plik. Nasza aplikacja powinna znajdować się w katalogu *My Documents\Visual Studio Projects\ Sumator\ bin\Adder.exe*. Jeśli jej tam nie ma, można wykorzystać Eksplorator Windows, aby ją znaleźć. Po dwukrotnym kliknięciu ikony naszej aplikacji pojawia się zaprojektowane

przez nas okno dialogowe, co oznacza, że program działa. Plik wykonywalny utworzony przez kompilator można uruchomić na każdym komputerze wyposażonym w system operacyjny Windows.

# Wpisywanie kodu źródłowego

Nasz program wprawdzie się uruchomił, ale nie wykonuje postawionego mu zadania. Na razie mogliśmy się przekonać, w jaki sposób tworzy się podstawową formę aplikacji i w jaki sposób umieszcza się elementy w oknie dialogowym. W oknie naszej aplikacji będziemy do dwóch pól tekstowych wpisywać dwie liczby, a w trzecim polu pojawi się wynik dodawania dwóch wprowadzonych składników. Aby to zadanie zostało wykonane, należy oprogramować przycisk *Dodaj* naszej aplikacji.

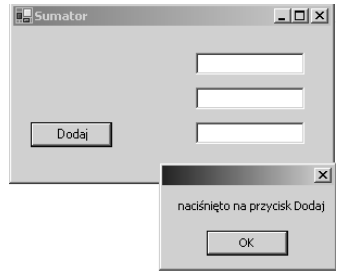
Wpisywanie kodu źródłowego odpowiedzialnego za wykonanie odpowiedniego zadania po naciśnięciu przycisku jest bardzo proste. Najpierw w oknie *Solution Explorer* należy kliknąć dwukrotnie naszą formę. Następnie trzeba dwukrotnie kliknąć przycisk *Dodaj*, pokaże się okno z kodem źródłowym, zawierające procedurę obsługującą *zdarzenie* związane z tym przyciskiem. Każdy obiekt kontrolny umieszczany w oknie dialogowym może reagować na określone zdarzenia (np. kliknięcie, dwukrotne kliknięcie, zaznaczenie, usunięcie zaznaczenia, itp.) generowane przez użytkownika programu. Aby umożliwić reagowanie na zdarzenia, środowisko generuje procedury odpowiedzialne za wykonanie funkcji w nich zawartych. Gdy dwukrotnie klikniemy przycisk *Dodaj*, który ma identyfikator `btnAdd`, środowisko utworzy nam procedurę, która będzie wykonywana za każdym razem, gdy użytkownik naszego programu naciśnie ten przycisk. Nazwa tej procedury jest domyślnie tworzona od identyfikatora obiektu, którego dotyczy i zdarzenia, które spowoduje jej wykonanie. Po dwukrotnym naciśnięciu przycisku *Dodaj* w naszej formie zostanie utworzona procedura `btnAdd_ Click`. Aby sprawdzić, jak zdarzenia wpływają na wykonanie procedury związanej ze zdarzeniem, można posłużyć się obiektem klasy `MessageBox` należącym do .NET Framework. Za pomocą klas z grupy, do której należy klasa `MessageBox` tworzone są formy, przyciski, pola tekstowe i inne elementy interfejsu użytkownika. Klasa `MessageBox` pozwala na wyświetlenie okna dialogowego z informacją, np. funkcja:

```
MessageBox.Show("naciśnięto na przycisk Dodaj")
```

spowoduje wyświetlenie okna dialogowego przedstawionego na rysunku 2.27. Spróbujmy wpisać powyższy kod do procedury związanej z naszym przyciskiem. Po uruchomieniu programu, za każdym razem, gdy naciśniemy przycisk, pojawi się okno dialogowe z odpowiednią informacją. To oznacza, że za każdym razem, gdy generujemy zdarzenie „kliknięcie przycisku", wykonywana jest procedura związana z tym zdarzeniem.

Wróćmy do procedury związanej z naszym przyciskiem i usuńmy jej zawartość. W jej miejsce wstawimy kod źródłowy, którego zadaniem będzie dodanie dwóch liczb i wyświetlenie wyniku tej operacji. Najpierw należy dokonać konwersji łańcucha znaków pól tekstowych, w których wpisywane są liczby, na wartości liczbowe, a następnie należy dodać dwie liczby do siebie i wyświetlić wynik. Wszystkie te zadania wykonamy za pomocą jednej linijki kodu:

```
txtResult.Text = (CInt(txtFirstValue.Text)_
   + CInt(txtSecondValue.Text)).ToString
```




**Rysunek 2.27.** *Funkcje klasy MessageBox pozwalają na korzystanie z okien dialogowych do wyświetlania informacji. Takie okna często wykorzystuje się podczas testowania aplikacji*

W powyższym wyrażeniu najpierw dokonywana jest konwersja łańcuchów tekstowych pierwszego i drugiego składnika na wartości całkowite, następnie te dwie liczby są dodawane do siebie, a wynik operacji jest wyświetlany w trzecim polu tekstowym, po zamianie na łańcuch tekstowy. Sposób programowania w Visual Basicu jest dokładniej opisany w rozdziale trzecim, „Wprowadzenie do programowania w Visual Basic .NET".

# Podsumowanie

Środowisko programistyczne wymyślono po to, by ułatwić pracę programistom i umożliwić pisanie, edycję, sprawdzanie poprawności i kompilację programów. Visual Studio zapewnia wiele możliwości, których nie sposób opisać w jednym rozdziale. W niniejszym rozdziale przedstawiono główne okna i narzędzia Visual Studio, a także pokazano sposób tworzenia aplikacji. W kolejnych rozdziałach nie zawsze będziemy korzystać ze środowiska graficznego, ale w trakcie samodzielnych prób tworzenia aplikacji możemy się coraz bardziej z nim „zaprzyjaźnić".

# Pytania i odpowiedzi

**P: Czy możemy tworzyć aplikacje korzystając z edytora tekstowego i konsoli, bez udziału środowiska programistycznego?**

**O:** W środowisku Visual Basic .NET można programować w dwojaki sposób. Możemy pisać kod źródłowy w edytorze tekstowym i kompilować go z wiersza poleceń konsoli albo tworzyć aplikacje w sposób wizualny, korzystając z możliwości oferowanych przez środowisko programistyczne. W środowisku programuje się jednak o wiele łatwiej, bo oferuje ono wiele narzędzi ułatwiających tworzenie aplikacji.

**P: Czy można zdefiniować własne funkcje dostępne w środowisku programistycznym?**

**O:** Oczywiście! Środowisko programistyczne umożliwia dostosowanie go do potrzeb użytkownika za pomocą kilku metod (np. można tworzyć własne makra i dodatki). W tej książce nie zostanie opisany sposób dostosowywania środowiska do własnych potrzeb, ale można przejrzeć przykłady zawarte w *Tools*, *Macros*, *Macros IDE*.

# Warsztat

W warsztacie znajduje się prosty quiz, który pozwoli na ugruntowanie wiedzy zdobytej w tym rozdziale, i ćwiczenia, które pomogą nabrać doświadczenia w wykorzystaniu tej wiedzy. Odpowiedzi na pytania i wskazówki do ćwiczeń znajdują się w dodatku A, „Odpowiedzi".

## Quiz

1. W którym oknie środowiska programistycznego można wyświetlić wszystkie pliki należące do tworzonego projektu?
2. W którym folderze domyślnie są umieszczane nowe projekty?
3. W jaki sposób można wybrać ikonę, która będzie przedstawiać plik wykonywalny tworzonej aplikacji?
4. Jeśli okno *Command* jest w trybie *Immediate*, to w jaki sposób można przejść w tryb *Command*?

## Ćwiczenie

Spróbujmy poeksperymentować z funkcjami klasy `MessageBox` i wyświetlić okno dialogowe z informacją, gdy użytkownik wykona konkretne zdarzenie. W obszarze okna edycji kodu źródłowego wybierzmy zmienną `txtResult` w prawej liście rozwijanej i zdarzenie `TextChanged` w lewej liście rozwijanej, a następnie napiszmy funkcję wyświetlającą okno dialogowe, gdy użytkownik spróbuje zmienić zawartość pola tekstowego, do którego przypisano zmienną `txtResult`.