Microsoft Visual Studio 2008. Księga eksperta

Autor: Lars Powers, Mike Snell
Tłumaczenie: Tomasz Walczak
ISBN: 978-83-246-2016-6
Tytuł oryginału: MS Visual Studio 2008 Unleashed
Format: 172x245, stron: 1312

Odkryj wspaniałe możliwości środowiska Visual Studio 2008 i twórz niezawodne aplikacje

- Jak optymalnie wykorzystać środowisko Visual Studio?
- Jak tworzyć wydajne aplikacje dla przedsiębiorstw?
- Jak wzbogacić środowisko programistyczne przy użyciu własnych dodatków?

Microsoft Visual Studio 2008 to nowoczesne środowisko programistyczne, które umożliwia bardziej precyzyjne i szybsze tworzenie niezawodnych aplikacji. W najnowszej wersji wprowadzono wiele poprawek w językach (takich jak zapytania w LINQ) oraz liczne nowości na platformie .NET. Usprawniono także dostępne narzędzia – na przykład w programie Visual Studio Team System, pozwalającym na skuteczną pracę zespołową. Wprowadzono w nim możliwości profilowania wydajności i udoskonalono system kompilacji grupowej. Wśród nowych funkcji pojawiły się kreatory do wiązania danych, inteligentne znaczniki oraz narzędzia do zarządzania projektami.

„Microsoft Visual Studio 2008. Księga eksperta" stanowi kompletne i szczegółowe omówienie tego wyjątkowego środowiska programistycznego. Z podręcznika dowiesz się, jak tworzyć dodatki, makra i kreatory oraz jak zbudować aplikacje oparte na formularzach Windows i platformie WPF. Poznasz produkty z rodziny Visual Studio Team System, które umożliwiają wydajną pracę w zespole. Nauczysz się korzystać z narzędzi programistycznych platformy .NET, pracować z bazami danych i zarządzać zmianami w kodzie źródłowym.

- Środowisko Visual Studio 2008-12-07 – rozwiązania i projekty
- Przeglądarki i eksploratory
- Tworzenie i wykorzystanie współużytkowanego kodu
- Korzystanie z narzędzi zwiększających produktywność
- Obiektowy model automatyzacji
- Tworzenie makr, kreatorów i dodatków
- Arkusze stylów
- Tworzenie aplikacji opartych na formularzach Windows
- Praca z bazami danych
- Dodawanie procesów do aplikacji
- Aplikacje biznesowe oparte na pakiecie Office
- Praca zespołowa i Visual Studio Team System
- Kontrolka kodu źródłowego

Wykorzystaj wiedzę ekspercką i zostań jeszcze bardziej profesjonalnym programistą!

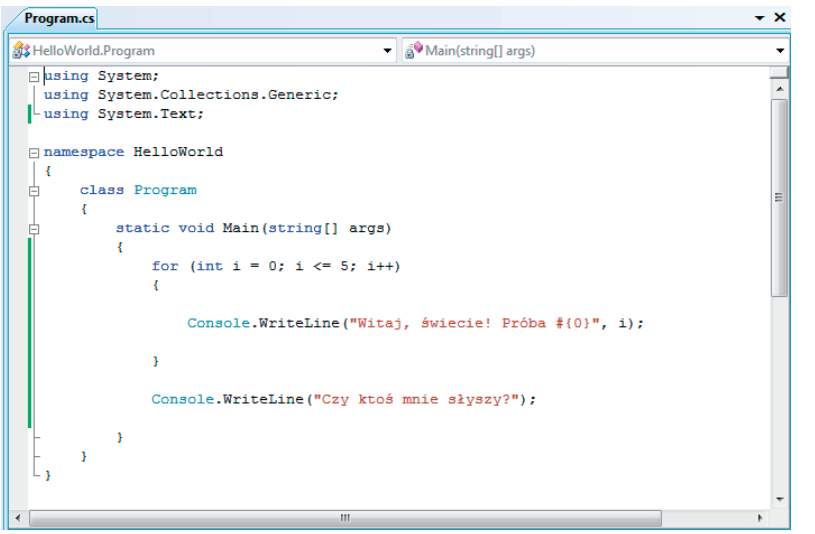
Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 8.
Korzystanie z narzędzi zwiększających produktywność

W rozdziale 6., „Wprowadzenie do edytorów i okien projektowych", opisaliśmy podstawowe funkcje okien projektowych i edytorów środowiska Visual Studio 2008. W tym rozdziale przedstawimy ich nieco bardziej zaawansowane możliwości, analizując liczne narzędzia zwiększające produktywność dostępne w IDE. Wiele z tych narzędzi jest zagnieżdżonych w edytorach tekstu. Inne są bardziej uniwersalne. Wszystkie służą jednemu celowi — mają pomagać programistom pisać kod szybko i poprawnie.

W rozdziale 6., „Wprowadzenie do edytorów i okien projektowych", przy opisie edytorów użyliśmy bardzo prostego programu — aplikacji konsolowej wyświetlającej w oknie konsoli napis `"Witaj, świecie"`. Na rysunku 8.1 widać, jak ostateczna wersja kodu wygląda w oknie edytora.

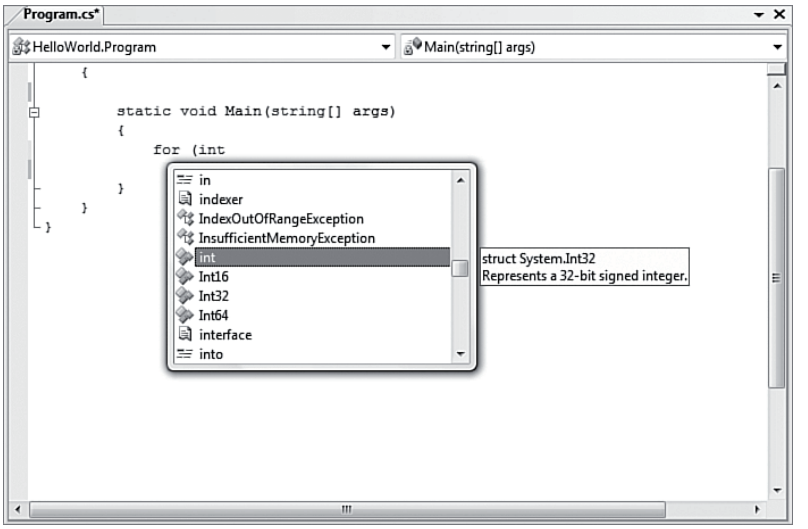


Rysunek 8.1. Program HelloWorld w edytorze kodu

Jeśli Czytelnik utworzył ten projekt i wpisał jego kod w Visual Studio, mógł zauważyć, że już w tym prostym programie narzędzia zwiększające produktywność znajdują zastosowanie. Po pierwsze, po rozpoczęciu wpisywania kodu w szablonowym pliku edytor dodaje tabulację i umieszcza kursor w nowej pozycji, dzięki czemu kod ma eleganckie wcięcia.

Po drugie, w czasie wpisywania wiersza kodu Visual Studio reaguje na każdy nowy znak, zgadując, jaką instrukcję programista chce napisać i udostępniając pomoc w różnej postaci (rysunek 8.2). Środowisko wyświetla wskazówki dotyczące uzupełniania pisanego kodu, informacje o wybieranych składowych i o parametrach potrzebnych do uzupełnienia danej metody. Te właściwości są wspólnie nazywane mechanizmem *IntelliSense*, a jego postaci i funkcje opisujemy szczegółowo w dalszej części rozdziału.

W czasie wpisywania kodu IDE nieustannie sprawdza za pomocą kompilatora, jaki tekst został już wpisany. Jeśli kompilator wykryje błąd, dynamicznie wyświetla informacje o tym w oknie *Output*.



Rysunek 8.2. Działanie mechanizmu IntelliSense

Dlatego już w przypadku jednego prostego wiersza kodu Visual Studio wykonuje wiele operacji zwiększających produktywność programisty:

- Inteligentnie formatuje kod.
- Sugeruje składnię kodu.
- Wyświetla opisy składowych, co pomaga użyć poprawnej składni.
- Graficznie wyróżnia odpowiadające sobie ograniczniki.
- Wskazuje błędy, nieustannie kompilując w tle bieżącą wersję kodu źródłowego.

Te właściwości dyskretnie pomagają programiście i prowadzą go przez proces pisania kodu, zwiększając szybkość wykonywania tego zadania.

Podstawowe narzędzia pomocnicze edytorów kodu

Już sam interfejs użytkownika edytora kodu udostępnia graficzne elementy, które pomagają radzić sobie z problemami często występującymi w czasie pisania kodu. Te mechanizmy pozwalają wykryć, co zmieniło się w dokumencie, a także jakie problemy dotyczące kompilacji w nim występują. Ponadto różne elementy składni poszczególnych języków są graficznie wyróżnione przy użyciu kolorowego tekstu.

Śledzenie zmian

W czasie modyfikowania pliku z kodem źródłowym niezwykle przydatna jest wiedza o tym, które wiersze zostały już zatwierdzone (czyli zapisane na dysk), a które nie. Śledzenie zmian pozwala to wykryć. Na marginesie wyboru edytora widoczna jest żółta pionowa linia ciągnąca się wzdłuż wszystkich wierszy kodu, które zostały zmodyfikowane, ale nie są jeszcze zapisane. Jeśli zawartość pliku zmieniła się, a następnie programista ją zapisał, środowisko oznacza ją zieloną pionową linią na marginesie wyboru.

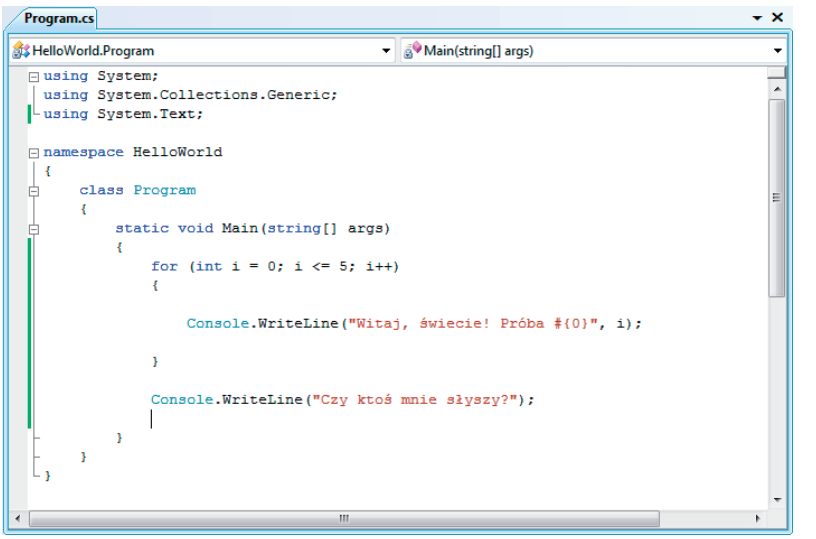
Dzięki tym żółtym i zielonym liniom można szybko wyróżnić:

- Kod, który nie zmienił się od czasu wczytania pliku (brak linii).
- Kod, który został zmieniony i zapisany po wczytaniu pliku (zielona linia).
- Kod, który został zmodyfikowany, ale nie jest jeszcze zapisany (żółta linia).

Śledzenie zmian funkcjonuje do czasu zamknięcia okna edytora. Inaczej mówiąc, śledzenie zmian obejmuje jedynie daną „sesję" modyfikowania bieżącego dokumentu. Po zamknięciu i ponownym otwarciu okna linie znikną, ponieważ dany dokument działa wtedy w nowej sesji.

Rysunek 8.3 przedstawia fragment pliku kodu zawierający różne linie związane ze śledzeniem zmian.

Rysunek 8.3. Śledzenie zmian



Wskazówki dotyczące problemów

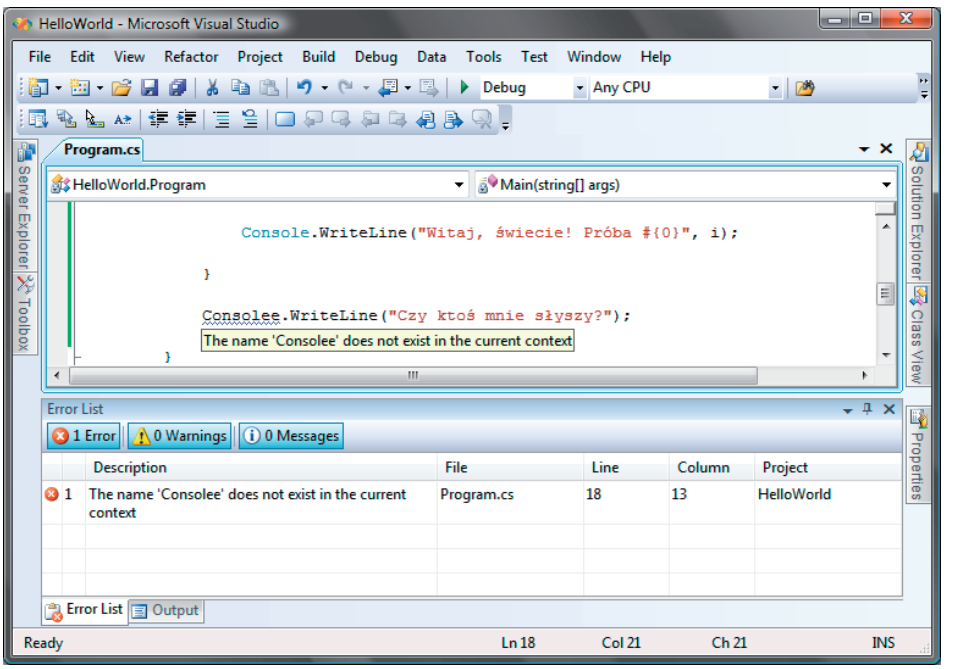
Kompilator Visual Studio współpracuje z edytorem kodu w celu wyróżnienia wszelkich problemów wykrytych w kodzie źródłowym. Kompilator potrafi działać w tle, co umożliwia edytorowi wskazywanie problemów w czasie wpisywania kodu przez programistę, dzięki czemu nie trzeba czekać na etap kompilacji projektu.

Problemy dotyczące kodu są oznaczane „wężykami", czyli falistymi, kolorowymi liniami umieszczanymi pod niepoprawnymi fragmentami. Są to te same wężyki, które w programie Microsoft Word służą do oznaczania błędów związanych z pisownią i gramatyką. Kolor wężyka oznacza klasę problemu. Tabela 8.1 opisuje, jakie typy problemów odpowiadają poszczególnym kolorom.

Tabela 8.1. Kolory informujące o problemach

Kolor	Problem
Czerwony	Błąd składni. Kod nie skompiluje się z powodu naruszenia składniowych wymagań i zasad języka.
Niebieski	Błąd semantyczny. Informuje on o tym, że kompilator nie potrafi w bieżącym kontekście znaleźć użytego typu lub rozpoznać konstrukcji programowej. Niebieskim wężykiem zostanie oznaczona na przykład nazwa typu, który nie jest dostępny w kontekście kompilacji. Najczęściej wskazuje to na literówkę (na przykład niepoprawnie napisaną nazwę klasy).
Purpurowy	Ostrzeżenie. Purpurowy wężyk oznacza, że dany fragment kodu wywołał ostrzeżenie.

Umieszczenie kursora myszy nad wskazówką dotyczącą problemu pozwala wyświetlić komunikat o błędzie lub o ostrzeżeniu, co widać na rysunku 8.4.



Rysunek 8.4. Wskazówki dotyczące problemów w kodzie

Aktywne odnośniki

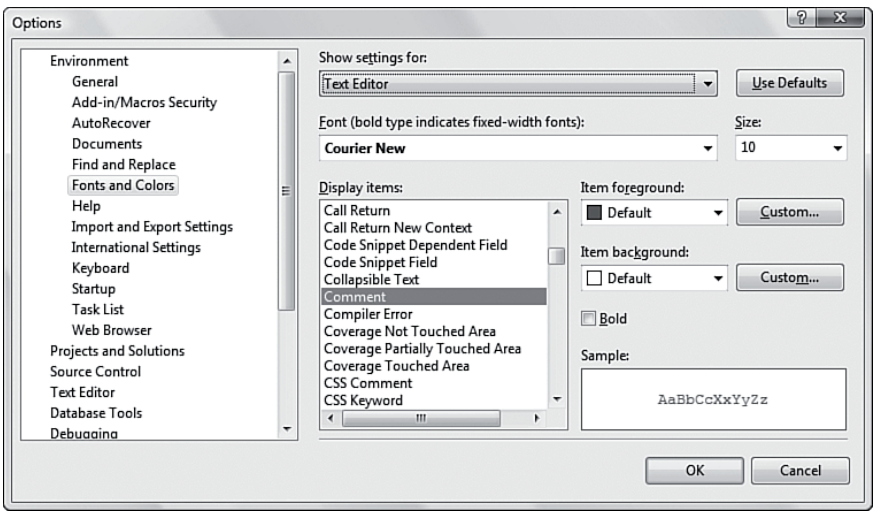
Edytory tekstu obsługują w dokumentach aktywne odnośniki. Kliknięcie odnośnika powoduje otwarcie danego adresu URL w przeglądarce. Jednym z zastosowań tej właściwości jest zagnieżdżanie w komentarzach adresów URL prowadzących do dokumentacji lub innych pomocnych informacji.

Kolorowanie składni

Edytor tekstu potrafi wykrywać różne konstrukcje kodu i oznaczać je, dzięki czemu można je łatwiej zidentyfikować. Na przykład okno edytora kodu domyślnie wyświetla komentarze na zielono. Identyfikatory w kodzie są czarne, słowa kluczowe niebieskie, łańcuchy znaków czerwone i tak dalej.

Liczba różnych elementów, jakie edytor tekstu potrafi wykryć i nadać im odmienny kolor, jest bardzo duża. Edytor rozpoznaje ponad 100 różnych elementów. Programista może dostosować sposób kolorowania każdego z tych typów do własnych preferencji za pomocą węzła *Environments* w oknie dialogowym *Options*. Może lubi używać większej czcionki? Może wyższy kontrast bardziej odpowiada wykonywanym przez programistę zadaniom? A co z umieszczaniem większej ilości kodu na widocznym obszarze ekranu? Jest to tylko kilka przyczyn, które mogą skłaniać programistę do modyfikacji ustawień domyślnych za pomocą wspomnianego okna dialogowego.

Rysunek 8.5 przedstawia okno dialogowe z czcionkami i kolorami. Umożliwia ono określenie koloru tekstu oraz koloru tła zwykłego kodu, kodu HTML, kodu CSS i innych elementów. Można wybrać element z ich listy, a następnie zmienić sposób jego kolorowania przy użyciu list rozwijanych z kolorami tekstu i tła.



Rysunek 8.5. Ustawianie czcionki i koloru

Uwaga

Możliwość kontroli sposobu kolorowania elementów edytora kodu to nie wszystko, co oferuje okno dialogowe widoczne na rysunku 8.5. Za jego pomocą można zmienić schematy kolorów wszystkich okien środowiska Visual Studio. Obiekt wybrany z listy rozwijanej *Show Settings For* pozwala określić modyfikowany fragment środowiska IDE, co pociąga za sobą zmianę elementów widocznych na liście *Display Items*.

Zawsze można użyć przycisku *Use Defaults* (w prawym górnym rogu okna dialogowego) i przywrócić domyślne ustawienia kolorów.

Schematy i nawigacja

Niektóre dokumenty, takie jak pliki z kodem źródłowym i znacznikami, zawierają naturalne relacje nadrzędny-podrzędny między elementami w zakresie organizacji i składni kodu. Na przykład węzły danych XML mogą zawierać inne węzły. Także funkcje i inne konstrukcje języków programowania, takie jak pętle czy bloki `try-catch`, działają jako kontenery dla innych wierszy kodu. Generowanie schematów (ang. *outlining*) umożliwia utworzenie graficznej reprezentacji relacji nadrzędny-podrzędny.

Schematy kodu

Schematy kodu są tworzone w edytorach kodu. Umożliwiają zwijanie lub rozwijanie obszarów kodu w granicach okna edytora. Na marginesie wyboru widoczne są linie grupujące oraz pola służące do rozwijania i zwijania tekstu. Te pola można kliknąć w celu ukrycia lub wyświetlenia wierszy kodu pogrupowanych w logiczny sposób.

Tworzenie schematów kodu najłatwiej jest zrozumieć na prostym przykładzie. Wróćmy do rysunku 8.1. Widać na nim wyjściowy kod aplikacji konsolowej. Zawiera ona procedurę `Main`, deklarację klasy, deklarację przestrzeni nazw i kilka instrukcji `using`. Grupy schematu kodu widoczne na marginesie wyboru reprezentują graficznie obszary kodu, które mogą zostać zwinięte (ukryte).

Ponieważ deklaracja klasy to logiczny kontener, na marginesie wyboru przy tym wierszu kodu znajduje się pole umożliwiające zwinięcie kodu (pole ze znakiem minus). Między tym polem a końcem kontenera ciągnie się linia (w tym przypadku, ponieważ jest to kod w języku C#, deklaracja klasy kończy się nawiasem klamrowym). Kliknięcie pola zwijania przy deklaracji klasy spowoduje, że Visual Studio ukryje cały jej kod.

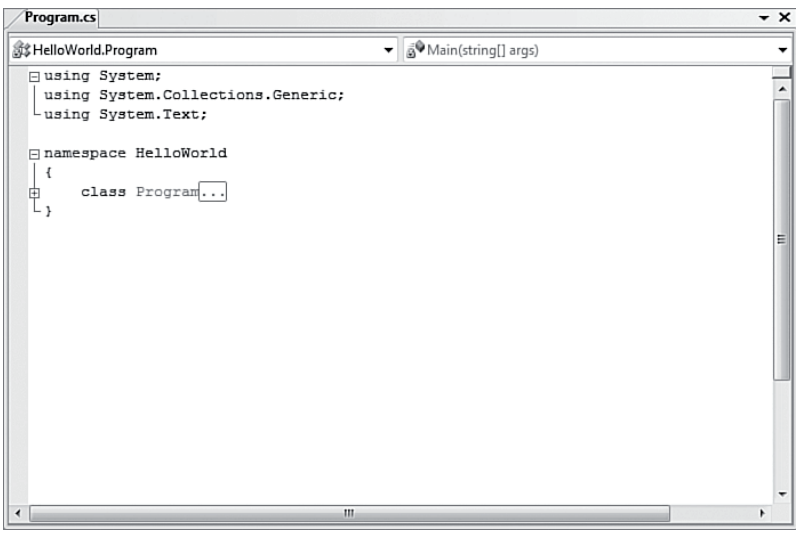
Wskazówka

Języki Visual Basic i C# umożliwiają samodzielne tworzenie nazwanych obszarów kodu. Służy do tego specjalne słowo kluczowe. Należy użyć pary `#region` i `#endregion` (`#Region` i `#End Region` w Visual Basic), aby utworzyć własny kontener na kod, który będzie odpowiednio wyświetlany przez mechanizm obsługi schematów kodu. Ponieważ każdy obszar ma nazwę, jest to wygodna technika organizowania i segregowania logicznych fragmentów kodu. Na przykład kod wygenerowany przez okno projektowe formularzy Windows jest automatycznie umieszczany w obszarze o nazwie `Windows Forms Designer generated code`.

Szybki sposób na utworzenie regionu polega na użyciu opcji *Surround With*. W edytorze należy zaznaczyć kod, który ma znaleźć się w nowym obszarze, kliknąć ten fragment prawym przyciskiem myszy, wybrać opcję *Surround With* z menu kontekstowego, a następnie użyć polecenia *#region* (*#Region* w Visual Basic).

Rysunek 8.6 przedstawia okno edytora, w którym kod deklaracji klasy jest ukryty. Warto zauważyć, że w polu zwijania widoczny jest teraz znak plus, co informuje, że kliknięcie go spowoduje ponowne wyświetlenie ukrytego obecnie kodu. Ponadto pierwszy wiersz kodu deklaracji klasy został zmodyfikowany i kończy się polem z trzema kropkami.

Rysunek 8.6. Zwinięty obszar schematu

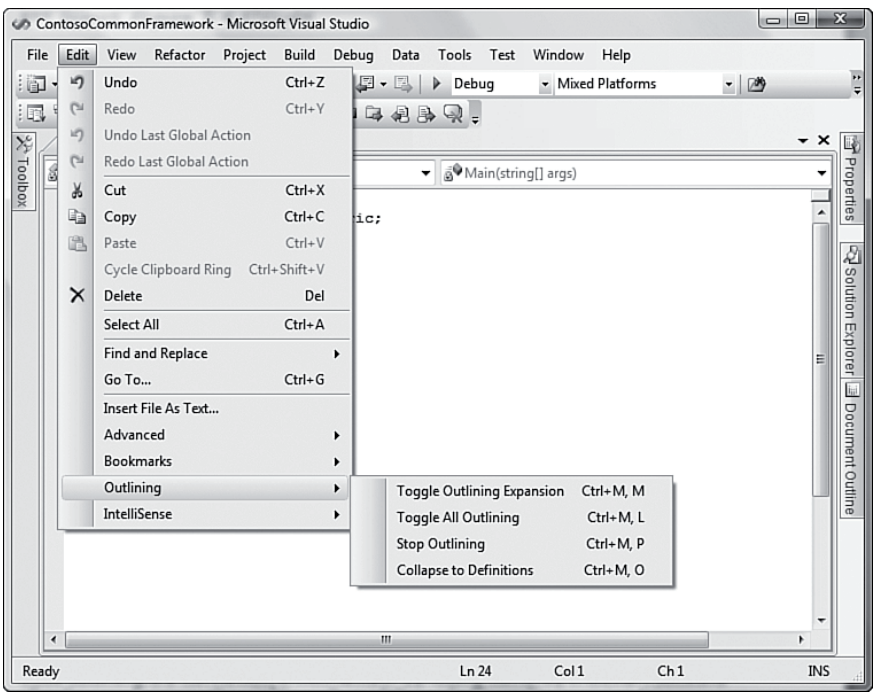


Edytor HTML także obsługuje tworzenie podobnych schematów. Elementy HTML można rozwijać i zwijać, aby wyświetlić lub ukryć zawarte w nich obiekty.

Menu Outlining

W menu *Edit/Outlining* dostępnych jest kilka poleceń związanych ze schematami kodu (rysunek 8.7).

- **Toggle Outlining Expansion** — ukrywa lub wyświetla obszar schematu na podstawie bieżącej pozycji kursora w oknie edytora.
- **Toggle All Outlining** — ukrywa lub wyświetla wszystkie obszary schematu w edytorze.
- **Stop Outlining** — wyłącza automatyczne ukrywanie schematów kodu (wszystkie ukryte obszary zostaną rozwinięte). To polecenie jest dostępne tylko po wcześniejszym włączeniu automatycznego generowania schematu.
- **Stop Hiding Current** — usuwa schemat bieżącego obszaru. To polecenie jest dostępne tylko po wcześniejszym wyłączeniu automatycznego generowania schematu.
- **Collapse to Definitions** — ukrywa wszystkie obszary procedur. To polecenie jest przydatne do wyświetlania pojedynczych wierszy kodu dla wszystkich składowych danego typu.
- **Start Automatic Outlining** — włącza generowanie schematu kodu. To polecenie jest dostępne tylko po wcześniejszym wyłączeniu generowania schematu.



Rysunek 8.7. Menu Edit/Outlining

Generowanie schematu kodu to udogodnienie, które umożliwia zmniejszenie widocznej ilości kodu i zwiększenie jego czytelności poprzez ukrycie nieistotnych fragmentów. Dzięki temu można wyświetlać jedynie specyficzne obszary dotyczące wykonywanego zadania.

Wskazówka

Umieszczenie wskaźnika myszy nad polem z trzema kropkami w ukrytym obszarze kodu pozwala wyświetlić zawartość danego obszaru w polu przypominającym okno podpowiedzi. Dzięki temu nie trzeba rozwijać danego obszaru, aby go ponownie wyświetlić.

Nawigowanie po kodzie HTML

Jednym z problemów związanych z dużymi lub złożonymi stronami internetowymi jest nawigowanie po nich. Analiza kodu HTML może być skomplikowana z powodu wielu poziomów i warstw zagnieżdżonych znaczników. Programista może utworzyć stronę zawierającą tabelę w tabeli, która znajduje się w następnej tabeli. Jak jednak ma określić w czasie modyfikowania kodu HTML (w oknie projektowym lub edytorze kodu), nad którym fragmentem tabeli pracuje? Mówiąc inaczej, skąd wiadomo, która część hierarchii znaczników jest aktywna?

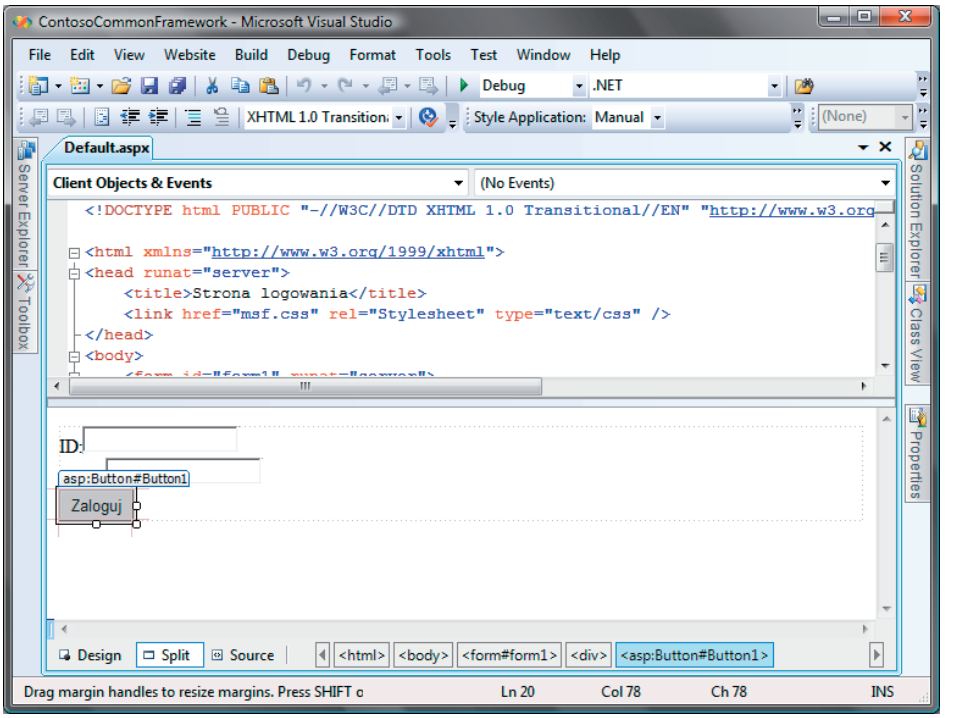
Mechanizm nawigowania po znacznikach

Powyższy problem można rozwiązać za pomocą mechanizmu nawigowania po znacznikach dostępnego w Visual Studio. Do jego obsługi służy szereg przycisków widocznych na dole okna edytora stron internetowych. Te przyciski znajdują się na prawo od zakładek okna projektowego i edytora. Środowisko wyświetla znaczniki (jako przyciski) reprezentujące znaczniki od bieżącego do zewnętrznego. Jeśli ścieżka jest zbyt długa, aby można ją wyświetlić w oknie edytora w całości, jest skracana po stronie znacznika zewnętrznego. Dostępny jest wtedy przycisk, który pozwala przejść w kierunku tego znacznika.

Rysunek 8.8 przedstawia program z rozdziału 6. — stronę HTML służącą do logowania. W czasie modyfikowania przez programistę przycisku *OK* tej przykładowej strony mechanizm nawigowania wyświetla ścieżkę prowadzącą do zewnętrznego znacznika `<html>`.

Dowolnego przycisku widocznego na pasku nawigacji można użyć do bezpośredniego zaznaczenia zawartości danego znacznika z uwzględnieniem jego samego lub bez niego. Lista rozwijana, wyświetlana w wyniku kliknięcia przycisku znacznika, zawiera opcje umożliwiające wybranie całego znacznika lub tylko jego zawartości. Ta pierwsza opcja powoduje zaznaczenie znacznika wraz z całą jego zawartością. Druga możliwość wiąże się z pominięciem początku i końca znacznika, przez co wybierana jest jedynie jego zawartość.

Jest to doskonały mechanizm do szybkiego poruszania się w górę i w dół dużych drzew znaczników dokumentów HTML.



Rysunek 8.8. Mechanizm nawigowania po znacznikach

Okno Document Outline

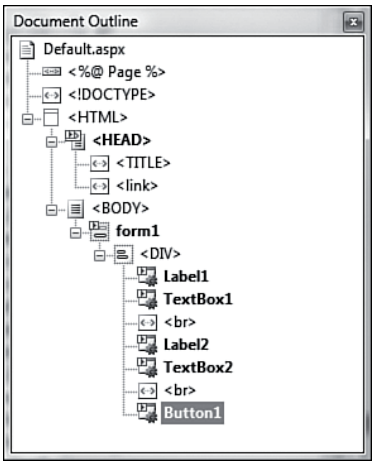
Okno *Document Outline* wyświetla drzewo reprezentujące elementy HTML strony. Ten hierarchiczny widok to także doskonałe narzędzie do nawigowania, ponieważ umożliwia szybkie objęcie wzrokiem całej struktury strony internetowej i natychmiastowe przejście do dowolnego elementu.

Aby otworzyć okno *Document Outline*, należy wybrać opcję *Document Outline* z menu *View*. Rysunek 8.9 przedstawia przykładowe okno schematu kodu. Widać w nim elementy nagłówka, strony i ciała oraz jednostki ze skryptów i kodu.

Kliknięcie dowolnego elementu powoduje przejście do niego (i zaznaczenie go) w oknie projektowym oraz oczywiście rozwinięcie lub zwinięcie odpowiednich węzłów drzewa, jeśli to konieczne.

Inteligentne znaczniki i operacje

Inteligentne znaczniki i **inteligentne operacje** (tych pojęć można w zasadzie używać zamiennie) to właściwości związane z menu lub mechanizmem IntelliSense służące do automatycznego konfigurowania często używanych kontrolek oraz operacji często wykonywanych w IDE. Okna



Rysunek 8.9. Okno Document Outline

projektowe i edytory obsługują inteligentne znaczniki w wielu różnych sytuacjach. W poniższych punktach opisujemy kilka sposobów ułatwiania sobie pracy za pomocą inteligentnych znaczników. Zacznijmy od okna projektowego HTML.

Okno projektowe HTML

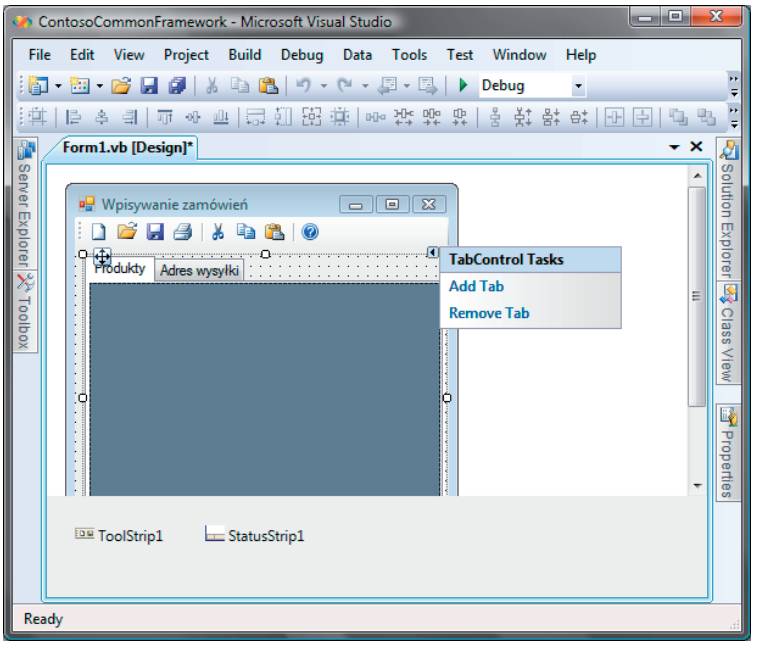
Po umieszczeniu kontrolki w oknie projektowym HTML pojawia się wyskakująca lista często wykonywanych operacji. Te operacje, nazywane wspólnie **inteligentnymi operacjami**, umożliwiają „ustawienie" danej kontrolki w celu szybkiego skonfigurowania jej do wykonywania odpowiednich zadań.

Po przeciągnięciu nowej kontrolki na powierzchnię okna projektowego automatycznie pojawia się wyskakująca lista często wykonywanych operacji. Można użyć jej do szybkiego skonfigurowania właściwości kontrolki, a także operacji często wykonywanych za jej pomocą. Na przykład po dodaniu do strony kontrolki `GridView` pojawia się lista operacji umożliwiająca szybkie włączenie sortowania, stronicowania i możliwości modyfikowania danych tej kontrolki. W przypadku kontrolki `TextBox` widoczna jest lista, która pozwala szybko powiązać z nią kontrolkę służącą do sprawdzania poprawności danych.

Także w oknie projektowym formularzy Windows wyświetlane są inteligentne znaczniki.

Okno projektowe formularzy Windows

W oknie projektowym formularzy Windows funkcjonalność inteligentnych znaczników jest taka sama, jednak wyglądają one nieco inaczej. Na krawędzi kontrolek formularza udostępniających takie znaczniki wyświetlane są odpowiednie ikony (zwykle po prawej stronie górnej krawędzi). Kliknięcie takiej ikony powoduje wyświetlenie małej listy rozwijanej z dostępnymi operacjami. Rysunek 8.10 przedstawia działanie inteligentnego znacznika kontrolki `TabControl`.



Rysunek 8.10. Inteligentny znacznik kontrolki TabControl

Edytor kodu

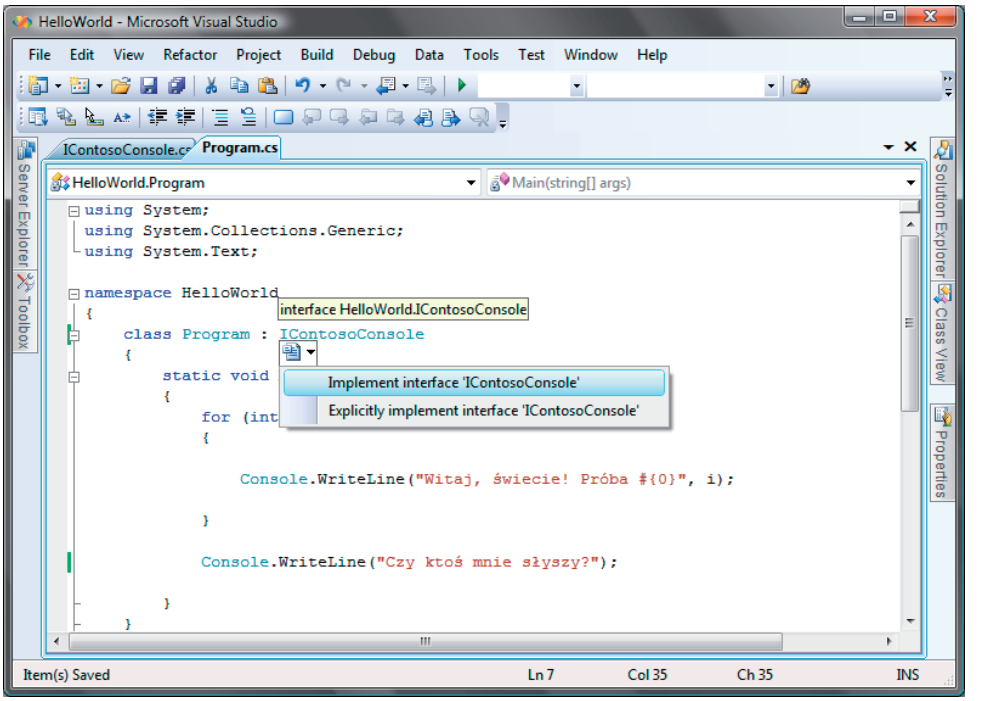
Inteligentne znaczniki pojawiają się także w kodzie, na przykład przy pisaniu interfejsów. Zwykle implementacja interfejsu wymaga napisania dużej ilości kodu, ponieważ trzeba utworzyć składowe odpowiadające wszystkim składowym zdefiniowanym w interfejsie. Inteligentny znacznik pozwala wygenerować je automatycznie w jednym z dwóch trybów nazywania elementów:

- **Explicit naming** — składowym nadawane są nazwy utworzone na podstawie używanego interfejsu.
- **Implicit naming** — nazwy składowych nie są związane z nazwami używanego interfejsu.

Działanie tego inteligentnego znacznika przedstawione jest na rysunku 7.11.

Mechanizm IntelliSense

IntelliSense to nazwa nadana grupie różnych narzędzi pomagających w pisaniu kodu i działających w oknie edytora tekstu. Jedynym zadaniem tego mechanizmu jest pomoc programiście w **szybkim** pisaniu poprawnego składniowo kodu. Ponadto IntelliSense ma udostępniać wystarczająco wiele wskazówek, aby programista mógł pisać kod poprawny w danym **kontekście**, czyli taki, który ma sens przy uwzględnieniu otaczających go wierszy.



Rysunek 8.11. Inteligentny znacznik do implementacji interfejsu

W czasie wpisywania tekstu w edytorze przez programistę IntelliSense działa na zapleczu jako agent odpowiedzialny za: udostępnianie fragmentów kodu pasujących do już wpisanych znaków, wyróżnianie lub wybieranie tych, które są najbardziej prawdopodobne w danym kontekście, a także, jeśli programista wybierze taką opcję, za automatyczne wstawianie fragmentów kodu. Pozwala to zaoszczędzić czas, ponieważ nie trzeba wyszukiwać typów i składowych w dokumentacji. Dalsze oszczędności wynikają z automatycznego wstawiania tekstu, ponieważ programista nie musi samodzielnie wpisywać znaków.

W tym punkcie poświęcamy wiele miejsca na analizę działania IntelliSense w kontekście pisania zwykłego kodu, jednak warto pamiętać, że mechanizm ten obsługuje także dokumenty innych typów, na przykład pliki XML, HTML czy XSLT.

Wskazówka

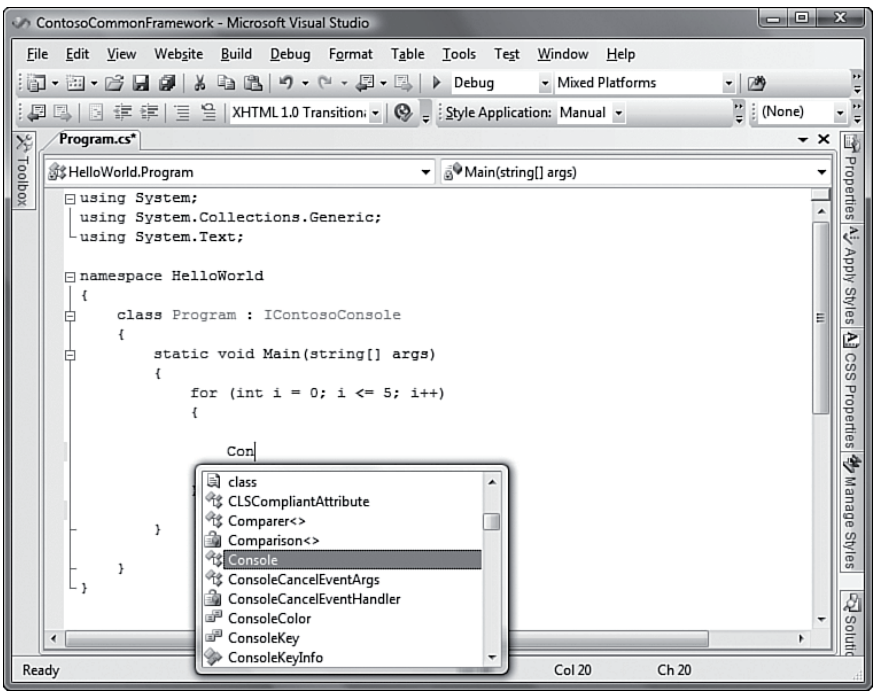
Dołączanie szablonów do dokumentu XML jest przydatne, ponieważ IntelliSense potrafi wykorzystać takie szablony i udostępnić więcej możliwości poprzez funkcję *List Members* (więcej na ten temat w dalszej części rozdziału).

Na działanie IntelliSense składa się wiele odrębnych funkcji, które bezkonfliktowo współpracują ze sobą **w czasie pisania kodu przez programistę**. Można je uruchomić także bezpośrednio z menu *Edit/IntelliSense* lub przy użyciu kombinacji *Ctrl*+spacja. Wiele funkcji można znaleźć także w menu kontekstowym edytorów kodu lub po kliknięciu prawym przyciskiem myszy dowolnego miejsca w oknie edytora. Przyjrzyjmy się tym funkcjom.

Uzupełnianie słów (Complete Word)

Uzupełnianie słów to podstawowa funkcja IntelliSense zapewniająca oszczędność czasu. Kiedy programista wpisze wystarczającą liczbę znaków, aby mechanizm mógł rozpoznać wpisywane słowo, proponowane jest uzupełnienie wpisywanego tekstu. Ta propozycja jest widoczna na liście alternatyw (nazywanej **listą uzupełniania**) i można ją wstawić do kodu za pomocą wciśnięcia pojedynczego klawisza. Stanowi to kontrast z koniecznością samodzielnego uzupełniania słów poprzez wpisywanie wszystkich znaków.

Rysunek 8.12 ilustruje ten proces. Na podstawie kontekstu oraz znaków wpisanych w edytorze IntelliSense wyświetla listę możliwych słów. Najbardziej prawdopodobne uzupełnienie jest wyróżnione, a programista może wybrać dowolną pozycję z listy, posługując się klawiszami strzałek lub myszą. Wciśnięcie klawisza *Tab* powoduje automatyczne wstawienie słowa w edytorze.



Rysunek 8.12. IntelliSense — uzupełnianie słów

Uwaga

Mechanizm uzupełniania słów uwzględnia kontekst kodu w wielu specyficznych sytuacjach. Na przykład w czasie wpisywania przez programistę typu wyjątku w bloku `try-catch` IntelliSense wyświetla na liście jedynie typy wyjątków. Podobnie w czasie wpisywania atrybutów dostępne są wyłącznie atrybuty, a przy implementacji interfejsu widoczne są tylko typy interfejsów. Ta właściwość mechanizmu IntelliSense obsługuje różne rodzaje dokumentów. Oprócz kodu języków C# i Visual Basic uzupełnianie działa także dla innych elementów, takich jak znaczniki HTML, atrybuty stylów CSS, pliki *.config* czy bloki skryptów HTML. Uzupełnianie słów w Visual Basic obejmuje funkcje niedostępne w języku C#. Dostępna jest lista uzupełniania z zakładkami, w której jednak zakładka zawiera najczęściej używane konstrukcje składniowe, a druga — wszystkie możliwe słowa.

W każdym momencie można samodzielnie wywołać mechanizm uzupełniania słów, używając kombinacji klawiszy *Ctrl*+spacja lub *Alt*+strzałka w prawo.

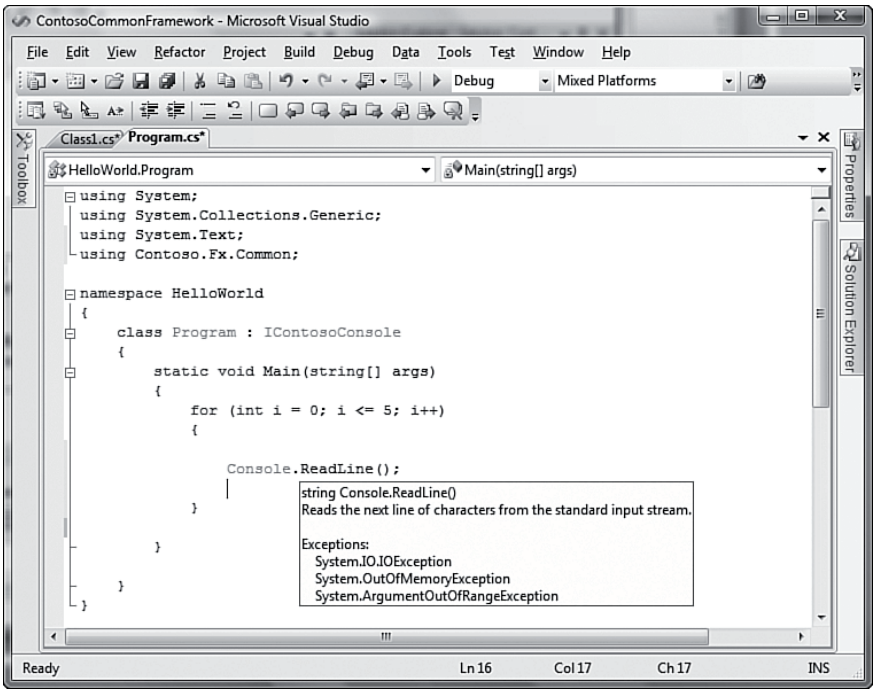
Wskazówka

Przytrzymanie klawisza *Ctrl* w czasie wyświetlania listy uzupełniania sprawia, że staje się ona częściowo przezroczysta. Jest to przydatne, jeśli w czasie wybierania elementów z listy programista chce zobaczyć zakryte przez nią wiersze kodu.

Okno z informacjami podręcznymi (Quick Info)

Okno z informacjami podręcznymi wyświetla kompletną deklarację danego fragmentu kodu oraz informacje pomocnicze z nim związane . Aby wyświetlić to okno, należy umieścić wskaźnik myszy nad danym identyfikatorem. Pojawi się wtedy okno wyskakujące z informacjami dostępnymi na jego temat.

Rysunek 8.13 przedstawia okno z informacjami podręcznymi z danymi dotyczącymi funkcji `Console.ReadLine`. Dostępna jest składnia deklaracji tej składowej, jej krótki opis oraz lista klas używanych w niej wyjątków. Opis widoczny w oknie z informacjami podręcznymi środowisko wyświetla także dla kodu napisanego przez programistę. Jeśli ze składową powiązane są komentarze, mechanizm IntelliSense przetworzy je i wyświetli na ich podstawie opisowe informacje.



Rysunek 8.13. IntelliSense — okno Quick Info

Okno z listą składowych (List Members)

Funkcja List Members działa identycznie jak uzupełnianie słów. W przypadku dowolnego typu lub przestrzeni nazw wyświetla ona przewijaną listę zawierającą wszystkie zmienne i funkcje składowe danego typu. Aby zobaczyć działanie tej funkcji, należy wykonać kilka operacji w otwartym oknie edytora kodu:

1. Wpisać nazwę klasy (wciśnięcie kombinacji *Ctrl*+spacja pozwala wyświetlić okno IntelliSense z dostępnymi nazwami klas).
2. Wpisać kropkę. Jest to informacja dla mechanizmu IntelliSense, że programista zakończył wpisywanie nazwy typu i chce przejść do „zasięgu" jego składowych.
3. Wyświetlona zostanie lista poprawnych składowych. W tym momencie można ją samodzielnie przewinąć i wybrać szukaną składową. Jeśli jednak programista wie, jakiej składowej potrzebuje, może wpisywać jej nazwę dopóty, dopóki IntelliSense nie wyświetli jej na podstawie wpisanych znaków.
4. Użyć uzupełniania słów, wciskając klawisz *Tab*, co powoduje automatyczne wstawienie składowej w odpowiednim wierszu kodu (co pozwala zaoszczędzić programiście pisania).

Ta funkcja działa także we współpracy z oknem z informacjami podręcznymi. Po wybraniu danej składowej z ich listy wyświetlane jest okno wyskakujące ze skrótowymi informacjami na temat wybranej składowej.

Uwaga

IntelliSense przechowuje informacje na temat składowych najczęściej wybieranych w oknie z listą składowych i oknie do uzupełniania słów. Dzięki tym informacjom mechanizm nie wyświetla ani nie zaznacza rzadko (jeśli w ogóle) używanych składowych danego typu.

Okno z informacjami o parametrach (Parameter Info)

Okno z informacjami o parametrach, jak sama nazwa wskazuje, służy do interaktywnego wyświetlania wskazówek dotyczących parametrów potrzebnych w wywołaniu danej funkcji. Ta właściwość jest szczególnie przydatna do pisania wywołań funkcji z długimi listami parametrów lub mających wiele przeciążonych wersji.

Okno z informacjami o parametrach jest wyświetlane zawsze, kiedy programista otworzy nawias po nazwie funkcji. Aby zobaczyć działanie tego mechanizmu, należy wykonać następujące operacje:

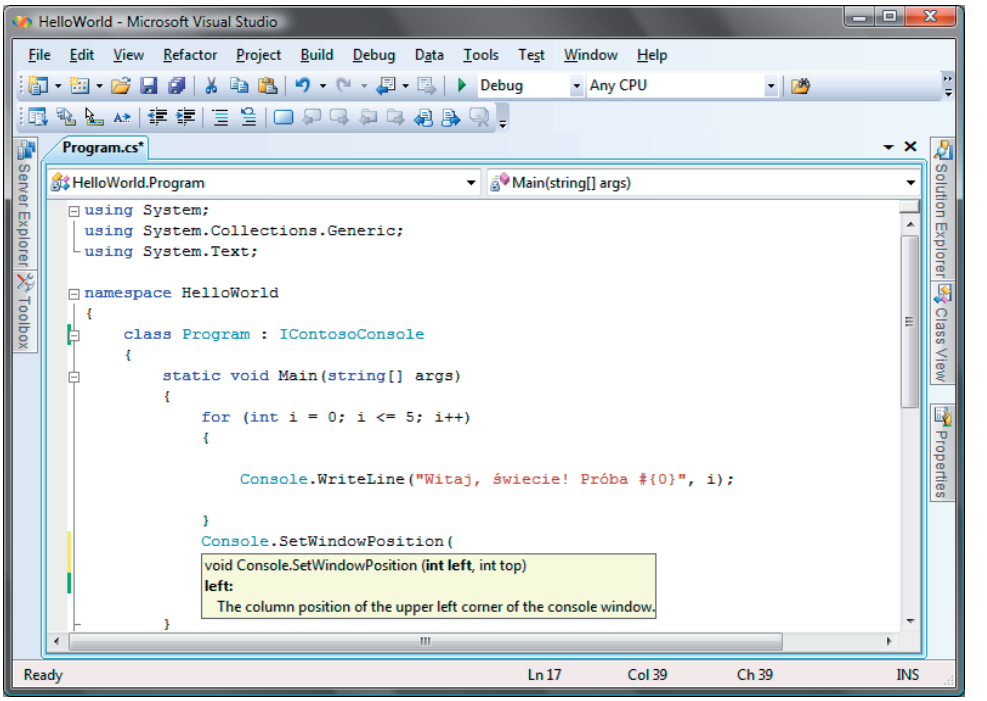
- Wpisać nazwę funkcji.
- Otworzyć nawias.
- Pojawi się okno wyskakujące z sygnaturą funkcji. Jeśli funkcja ma kilka poprawnych sygnatur (na przykład wiele przeciążonych wersji), można wyświetlać je po kolei, używając małych strzałek skierowanych w górę i w dół. Pozwala to wybrać właściwą sygnaturę.
- Po wybraniu odpowiedniej sygnatury można zacząć wpisywanie parametrów, które mają zostać przekazane do funkcji.

W czasie wpisywania parametrów widoczne jest okno wyskakujące z informacjami na ich temat. Prowadzi ono programistę przez listę parametrów, pogrubiając aktualnie wpisywany parametr. Wraz z wyróżnianiem poszczególnych parametrów mechanizm wyświetla ich definicje.

Na rysunku 8.14 programista wpisuje drugi parametr funkcji `SetWindowPosition` obiektu `Console`.

Porządkowanie instrukcji Using

Porządkowanie instrukcji `Using` to funkcja mechanizmu IntelliSense specyficzna dla języka C#. Obejmuje ona dwie odrębne opcje: *Remove Unused Usings* (usuwanie nieużywanych instrukcji `Using`) i *Sort Usings* (sortowanie instrukcji `Using`). Dostępne jest też trzecie polecenie, *Remove and Sort* (usuń i posortuj), które łączy obie wymienione funkcje. Wszystkie trzy instrukcje można wywołać przy użyciu opcji menu *Organize Usings* w menu kontekstowym edytora lub menu *Edit/IntelliSense*.



Rysunek 8.14. IntelliSense — okno z informacjami o parametrach

Funkcja *Remove Unused Usings* do doskonałe narzędzie do porządkowania kodu. Analizuje ono obecną wersję kodu i określa, które instrukcje `Using` są potrzebne do kompilacji projektu, a następnie usuwa wszystkie zbędne instrukcje tego rodzaju. Polecenie *Sort* działa w równie prosty sposób — ustawia instrukcje `Using` w porządku alfabetycznym według przestrzeni nazw.

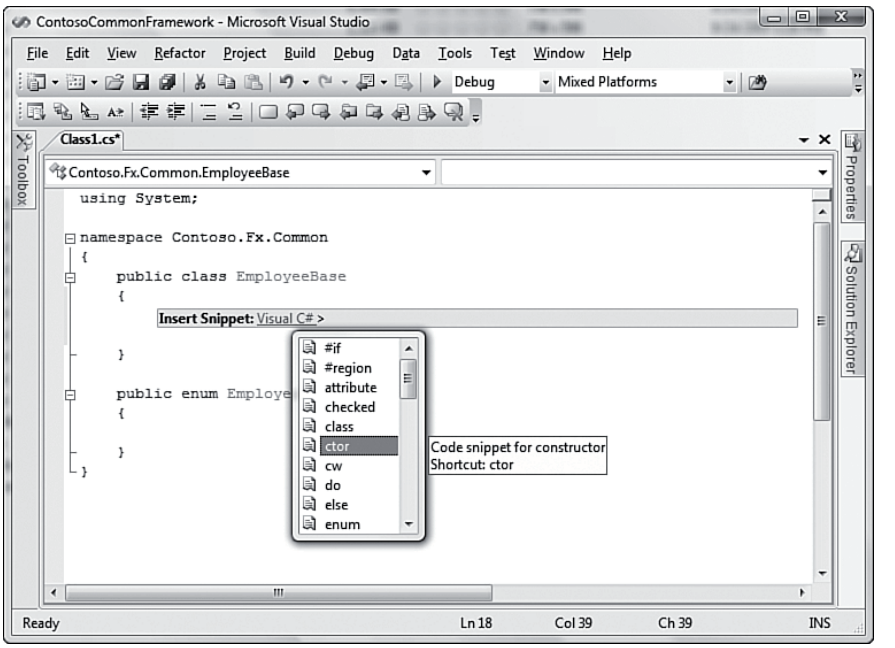
Fragmenty kodu i kod szablonowy

Fragmenty kodu (ang. *snippets*) to wbudowane wiersze kodu, które można wybrać i wstawić w edytorze kodu. Dostęp do każdego fragmentu kodu można uzyskać poprzez jego nazwę, tak zwany **alias**. Fragmenty kodu służą do automatyzacji operacji, które polegają na wpisywaniu powtarzającego się kodu. Można utworzyć własne fragmenty kodu lub używać domyślnej biblioteki standardowych elementów kodu udostępnianej przez Visual Studio.

Mechanizm wstawiania fragmentów kodu

Fragment kodu można wstawić klikając prawym przyciskiem myszy miejsce przeznaczone na ten kod, a następnie wybierając opcję *Insert Snippet* z menu podręcznego. W wyniku tej operacji środowisko otworzy okno do wstawiania fragmentów kodu, będące listą rozwijaną (lub grupą takich list) działającą podobnie do okna służącego do uzupełniania słów udostępnianego przez IntelliSense. Każdy element tej listy odpowiada fragmentowi kodu reprezentowanemu przez alias. Wybranie aliasu pozwala dodać fragment kodu do aktywnego dokumentu.

Fragmenty kodu są pogrupowane, co ułatwia wyszukiwanie potrzebnych jednostek. Na przykład aby wstawić konstruktor w klasie języka C#, należy kliknąć prawym przyciskiem myszy definicję klasy, wybrać opcję *Visual C#* z listy kategorii, a następnie użyć opcji *ctor*. Rysunek 8.15 przedstawia jeden z etapów tego procesu. Warto zauważyć, że kiedy programista wybierze kategorię fragmentów kodu, środowisko wyświetla w edytorze kodu miejsce na instrukcję, co pomaga tworzyć łańcuchy poleceń.



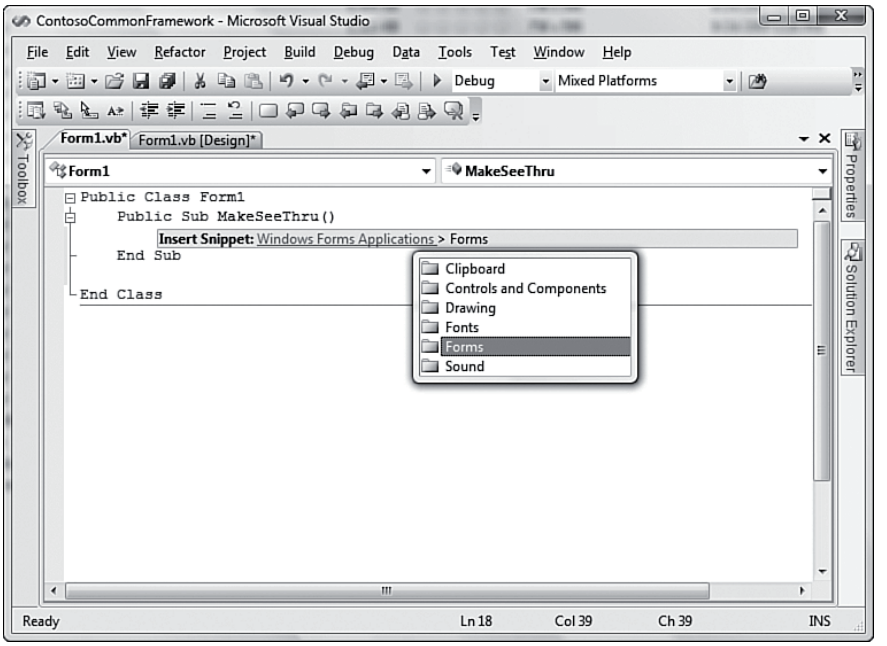
Rysunek 8.15. Fragment kodu ctor w języku C#

Wciąż trzeba napisać sensowny kod ciała konstruktora. Jednak dzięki tym fragmentom można wyeliminować konieczność żmudnego pisania kodu, którego wymyślenie nie wymaga zbyt wielu intelektualnych koni mechanicznych.

Rysunek 8.16 przedstawia ten sam proces w oknie z kodem Visual Basic. Operacja ta przebiega w niemal identyczny sposób jak w języku C#, a jedyna różnica polega na tym, że w Visual Basic podział fragmentów kodu na kategorie jest bogatszy.

Wskazówka

Szybki, alternatywny sposób wyświetlania okna do wstawiania fragmentów kodu polega na wpisaniu znaku zapytania i wciśnięciu klawisza *Tab*. Ta technika działa jedynie w plikach z kodem Visual Basic.



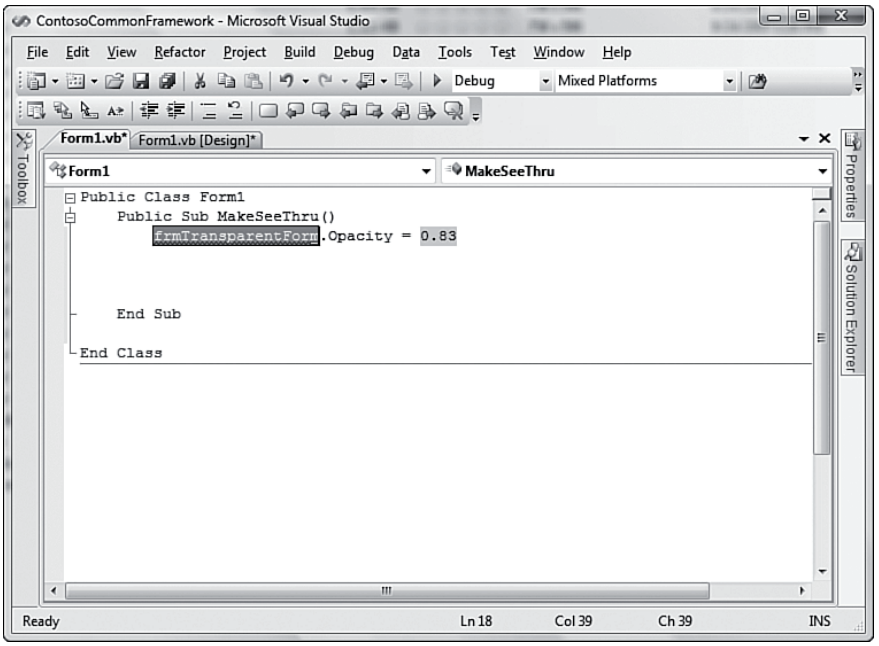
Rysunek 8.16. Fragmenty kodu w języku Visual Basic

Edytor języka Visual Basic działa nieco inaczej od edytora języka C# po rozwinięciu fragmentu kodu w oknie. Rysunek 8.17 przedstawia efekt przejścia przez kilka kategorii i wybrania fragmentu *Create Transparent Windows Form*. Warto zauważyć, że mechanizm wstawiania umieścił szablonowy fragment w kodzie Visual Basic, jednak nie był wystarczająco inteligentny (przynajmniej w tym przypadku), aby domyślić się nazwy przezroczystego formularza.

Jako nazwa formularza we fragmencie kodu podawany jest domyślny, fikcyjny tekst, który zostaje automatycznie wyróżniony. Wystarczy zacząć wpisywać odpowiednią nazwę, a domyślna zostanie zastąpiona. Także wartość właściwości określającej przezroczystość jest fikcyjna i można ją szybko zastąpić poprawną liczbą.

Wskazówka

Fragmenty kodu mogą mieć kilka miejsc na wpisanie danych, które programista może chcieć zmienić (i prawdopodobnie powinien to zrobić). Można przechodzić między tymi miejscami za pomocą klawisza *Tab*. Kiedy miejsce na dane jest wyróżnione (na niebiesko), można zacząć wpisywanie danych, aby zastąpić domyślną składnię wartościami mającymi sens w danym kontekście.



Rysunek 8.17. Fragment kodu do obsługi przezroczystości formularza

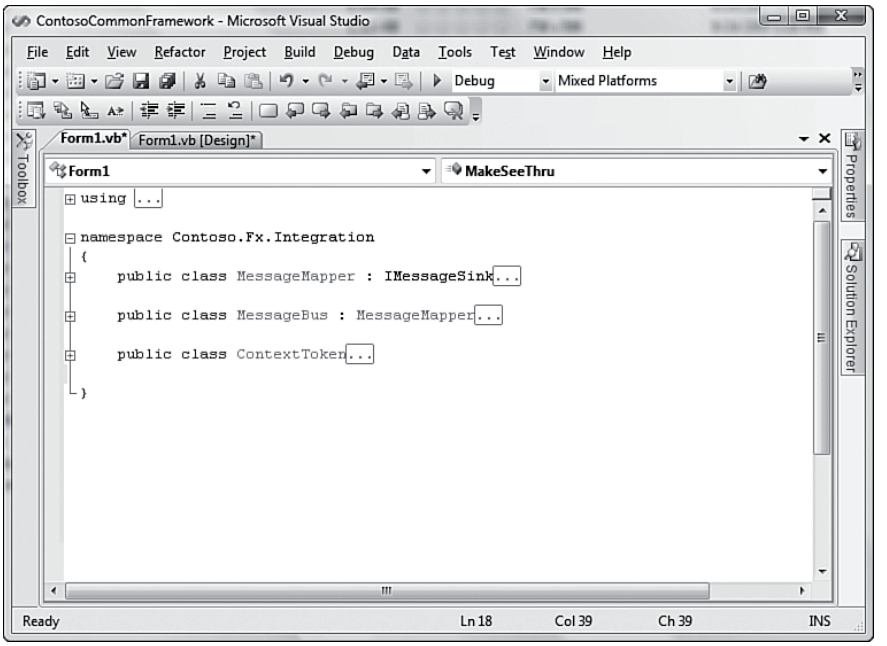
Otaczanie kodu fragmentami

Dokumenty języków C# i XML obsługują dodatkowy styl fragmentów kodu, o którym warto wspomnieć. Są to otaczające fragmenty kodu (opcja *Surround With*). Te fragmenty działają podobnie jak pozostałe (są to po prostu wbudowane wiersze kodu), jednak różnią się sposobem wstawiania.

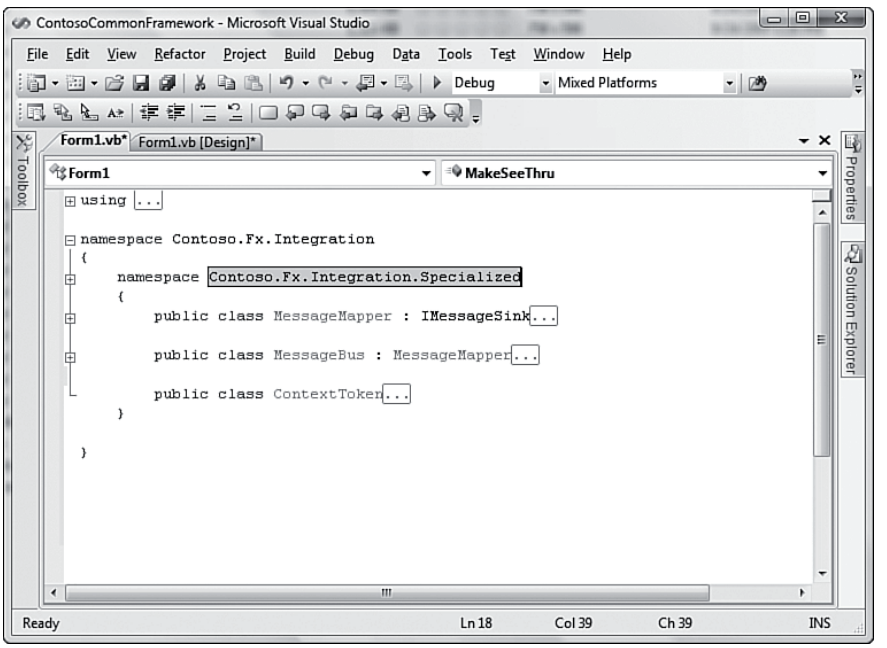
Używając otaczających fragmentów, można otoczyć kod zaznaczony w edytorze tekstu wybranym fragmentem. Na przykład programista może chcieć umieścić kilka deklaracji klas w jednej przestrzeni nazw. Dzięki otaczającym fragmentom wymaga to tylko dwóch operacji: wyróżnienia definicji klas i uruchomienia mechanizmu wstawiania. Tym razem zamiast wybierać opcję *Insert Snippet* z menu podręcznego, należy wybrać opcję *Surround With*. Wstawianie działa tak samo, jednak tym razem fragment kodu (tu jest to przestrzeń nazw) jest dodawany w odmienny sposób. Warto porównać kod przed i po wstawieniu tekstu (rysunki 8.18 i 8.19). Programista może umieścić definicje klas w nowej przestrzeni nazw zagnieżdżonej w innej przestrzeni za pomocą kilku kliknięć myszą.

Tworzenie własnych fragmentów kodu

Ponieważ fragmenty kodu są przechowywane w plikach XML, można łatwo utworzyć własne wersje. Kluczem do tego jest zrozumienie szablonu XML definiującego fragment kodu, a najlepiej zrobić to, analizując źródłowe dane XML kilku wbudowanych fragmentów dostępnych w IDE.



Rysunek 8.18. Przed wstawieniem otaczającego fragmentu kodu



Rysunek 8.19. Po wstawieniu otaczającego fragmentu kodu

Fragmenty kodu są przechowywane w plikach w specyficznych dla języka podkatalogach katalogu, w którym zainstalowane jest środowisko Visual Studio. Na przykład fragmenty kodu języka Visual Studio znajdują się domyślnie w podkatalogach katalogu *C:\Program Files\Microsoft Visual Studio 9.0\Vb\Snippets*. Choć pliki z fragmentami kodu mają format XML, ich rozszerzenie to *.Snippet*.

Format XML fragmentów kodu

Listing 8.1 przedstawia kod XML fragmentu kodu konstruktora języka C#.

Listing 8.1. Fragment kodu generujący konstruktor języka C#

```
<?xml version="1.0" encoding="utf-8" ?>

<CodeSnippets
xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">

   <CodeSnippet Format="1.0.0">
      <Header>
         <Title>ctor</Title>
         <Shortcut>ctor</Shortcut>
         <Description>Code snippet for constructor</Description>
         <Author>Microsoft Corporation</Author>
         <SnippetTypes>
            <SnippetType>Expansion</SnippetType>
         </SnippetTypes>
      </Header>
      <Snippet>
         <Declarations>
            <Literal Editable="false">
               <ID>classname</ID>
               <ToolTip>Class name</ToolTip>
               <Function>ClassName()</Function>
               <Default>ClassNamePlaceholder</Default>
            </Literal>
         </Declarations>
         <Code Language="csharp"><![CDATA[public $classname$ ()
   {
      $end$
   }]]>
         </Code>
      </Snippet>
   </CodeSnippet>
</CodeSnippets>
```

Podstawowa struktura deklaracji powyższego fragmentu kodu jest opisana w tabeli 8.2. Bardziej wyczerpujący opis szablonów można znaleźć w systemie pomocy MSDN Visual Studio, w węźle *Integrated Development Environment for Visual Studio/Reference/XML Schema References/Code Snippets Schema Reference*.

Najważniejsze w pisaniu fragmentów kodu jest to, aby zrozumieć, jak działa zastępowanie literałów i zmiennych. Załóżmy, że programista chce utworzyć fragment kodu języka C# wyświetlający

Tabela 8.2. Opisy węzłów XML używanych w plikach z fragmentami kodu

Węzeł XML	Opis
`<CodeSnippets>`	Element nadrzędny dla wszystkich informacji o fragmentach kodu. Określona jest w nim przestrzeń nazw języka XML służąca do definiowania fragmentów kodu w Visual Studio 2008.
`<CodeSnippet>`	Główny element danego fragmentu kodu. Ten znacznik określa informacje o formacie fragmentu (w przypadku pierwszego wydania Visual Studio 2008 powinna to być wersja 1.0.0). Choć w jednym elemencie `<CodeSnippets>` można umieścić wiele elementów `CodeSnippet`, zwykle w każdym pliku znajduje się tylko jeden fragment kodu.
`<Header>`	Kontener na metadane opisujące dany fragment kodu.
`<Title>`	Tytuł fragmentu kodu.
`<Shortcut>`	Zwykle ta sama wartość, co w tytule. Tekst tego elementu pojawia się na liście rozwijanej z fragmentami kodu.
`<Description>`	Opis fragmentu kodu.
`<Author>`	Autor fragmentu kodu.
`<SnippetTypes>`	Element nadrzędny przechowujący elementy opisujące typ fragmentu kodu.
`<SnippetType>`	Typ fragmentu kodu: `Expansion`, `Refactoring` lub `Surrounds With`. Nie można tworzyć niestandardowych fragmentów. Ta właściwość informuje Visual Studio o tym, w którym miejscu okna edytora należy wstawić dany fragment kodu. Fragmenty typu `Expansion` są wstawiane tam, gdzie znajduje się kursor. Fragmenty `Surrounds With` środowisko wstawia przed i po kodzie określonym przez bieżącą pozycję kursora lub zaznaczenia.
`<Snippet>`	Główny element zawierający kod fragmentu.
`<Declarations>`	Główny element literałów i obiektów używanych przez fragment kodu.
`<Literal>`	Łańcuch znaków, którego wartość można określić w procesie dodawania fragmentu kodu. Atrybut `Editable` tego znacznika określa, czy literał jest statyczny, czy też można go modyfikować. Fragment kodu `ctor` nie zawiera literałów, które można zmieniać. Warto porównać go z fragmentem służącym do ustawiania przezroczystości, który w czasie wstawiania umożliwia ustawienie nazwy formularza.
`<ID>`	Niepowtarzalny identyfikator literału.
`<ToolTip>`	Podpowiedź wyświetlana w momencie umieszczenia kursora nad literałem.
`<Function>`	Nazwa funkcji (opis w tabeli 8.3) wywoływanej w momencie, kiedy literał stanie się aktywny. Funkcji można używać tylko we fragmentach kodu języka C#.
`<Default>`	Domyślny literał wstawiany w edytorze.
`<Code>`	Element zawierający wstawiany kod.

prosty komentarz. Ten komentarz ma informować, że klasa została oceniona i zaakceptowana w procesie analizy kodu. Inaczej mówiąc, fragment ma umożliwiać dodawanie komentarzy podobnych do poniższego:

```
// Analiza kodu ContextToken.
//     Recenzent: Lars Powers

//     Data: 1/1/2006
//     Stan: Zatwierdzono
```

W tym fragmencie kodu cztery literały należy traktować jako zmienne. Mogą się one zmieniać za każdym razem, kiedy programista użyje tego fragmentu kodu. Są to: nazwa klasy, imię i nazwisko recenzenta, data oraz informacje o stanie. Można zadeklarować te elementy w sekcji `Declarations`:

```
<Declarations>
    <Literal Editable="False">
        <ID>classname</ID>
        <ToolTip>Nazwa sprawdzanej klasy/typu</ToolTip>
        <Function>ClassName()</Function>
        <Default>TuNazwaKlasy</Default>
    </Literal>
    <Literal Editable="True">
        <ID>reviewer</ID>
        <ToolTip>Imie i nazwisko recenzenta</ToolTip>
        <Default>ImieINazwiskoRecenzenta</Default>
    </Literal>
    <Literal Editable="True">
        <ID>currdate</ID>
        <ToolTip>Data sprawdzania</ToolTip>
        <Default>DataSprawadzania</Default>
    </Literal>
    <Literal Editable="True">
        <ID>approval</ID>
        <ToolTip>Zamien na "Zatwierdzono" lub "Odrzucono"</ToolTip>
        <Default>Zatwierdzono</Default>
    </Literal>
</Declarations>
```

Warto zwrócić uwagę, że powyższy kod wywołuje funkcję, która określa nazwę klasy we fragmencie kodu. Takie funkcje są dostępne jedynie w języku C# (a ich podzbiór — także w J#). Ich dokumentacja znajduje się w tabeli 8.3. W przypadku pozostałych literałów to programista musi podać w odpowiednich miejscach poprawne wartości.

Należy także udostępnić pewne podstawowe informacje nagłówkowe:

```
<Header>
    <Title>recenzja</Title>
    <Shortcut>recenzja</Shortcut>
    <Description>Komentarze dotyczace recenzji kodu</Description>
    <Author>L. Powers</Author>
    <SnippetTypes>
        <SnippetType>Expansion</SnippetType>
    </SnippetTypes>
</Header>
<Snippet>
```

Tabela 8.3. Funkcje dostępne we fragmentach kodu

Funkcja	Opis
`GenerateSwitchCases(`*`literalwyliczenia`*`)`	Tworzy składnię instrukcji `switch` obejmującą instrukcje `case` dla wszystkich wartości zdefiniowanych w wyliczeniu *`literalwyliczenia`* (C# i J#).
`ClassName()`	Wstawia nazwę klasy zawierającej dany fragment kodu (C# i J#).
`SimpleTypeName(`*`nazwatypu`*`)`	Pobiera nazwę typu określonego za pomocą literału *`nazwatypu`* i zwraca najkrótszą możliwą nazwę, uwzględniając instrukcje `using` obowiązujące w bieżącym bloku kodu. Na przykład wywołanie `SimpleTypeName ↪(System.Exception)` zwróci `Exception`, jeśli w pliku znajduje się instrukcja `using System` (C#).
`CallBase(`*`parametr`*`)`	Przydatna przy tworzeniu szkieletów składowych, które używają typu bazowego lub zwracają go. Jeśli jako parametr podana jest wartość `get`, `set` lub `method`, wywołany zostanie dany akcesor lub metoda klasy bazowej (C#).

Teraz fragment kodu jest składniowo kompletny. Choć powyższy fragment dodaje komentarze, taki sam proces prowadzi do tworzenia fragmentów generujących kod, które mają taką samą strukturę. Jeśli programista chciałby napisać otaczający fragment kodu, powinien zmienić wartość elementu `<SnippetType>` na `Surrounds With`.

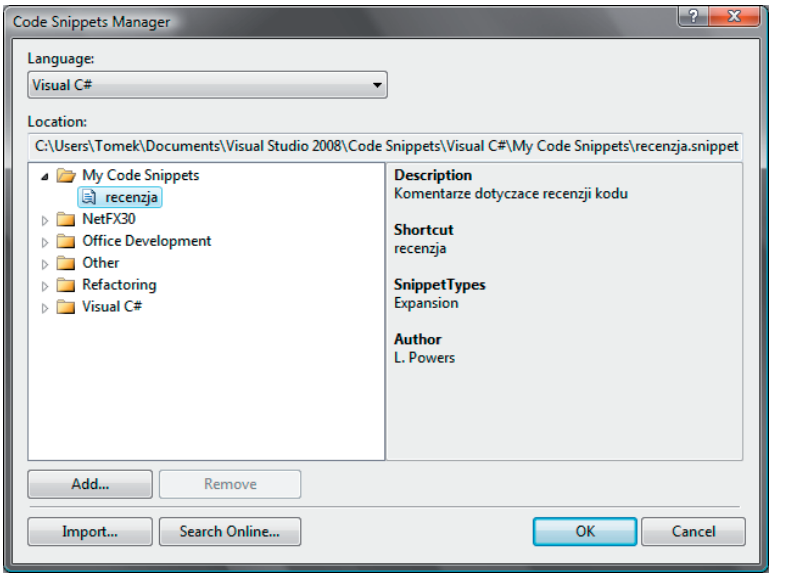
Teraz trzeba poinformować o nowym fragmencie środowisko Visual Studio.

Dodawanie fragmentów kodu do środowiska Visual Studio

Można użyć edytora XML środowiska Visual Studio w celu utworzenia dokumentu XML i zapisania go w katalogu. Dużą zaletą takiego podejścia jest możliwość wykorzystania mechanizmu IntelliSense wraz z szablonem XML fragmentów kodu. Pomaga to wpisywać nazwy elementów i określać relacje między nimi. Instalator środowiska Visual Studio tworzy domyślny katalog przeznaczony na niestandardowe fragmenty kodu. Znajduje się on w katalogu *Moje dokumenty*: *\Moje dokumenty\Visual Studio 2008\Code Snippets\VC#\My Code Snippets*. Visual Studio automatycznie udostępnia fragmenty kodu opisane w szablonach XML umieszczonych w tym katalogu.

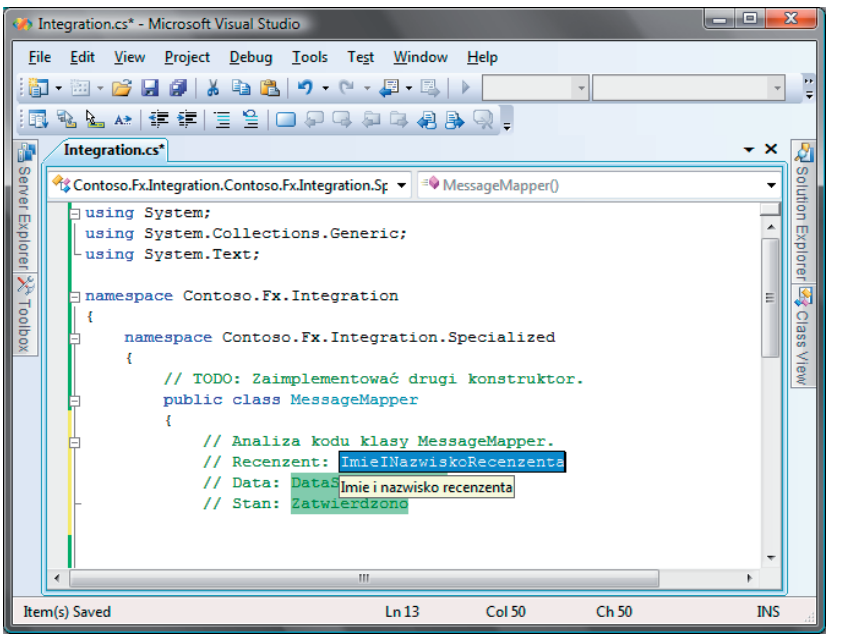
Okno dialogowe *Code Snippets Manager* to główne narzędzie do przeglądania dostępnych fragmentów kodu, dodawania nowych i usuwania istniejących (rysunek 8.20). Jak widać na rysunku, fragment kodu dodający informacje o recenzjach jest widoczny w katalogu *My Code Snippets*.

Można także dodać inne okna oprócz tych standardowych. W tym celu należy kliknąć przycisk *Add*, co pozwala dodać nowe katalogi, w których Visual Studio będzie szukać wyświetlanych fragmentów kodu.



Rysunek 8.20. Okno Code Snippets Manager

Rysunek 8.21 przedstawia wynik dodania nowego fragmentu kodu.



Rysunek 8.21. Wynik dodania nowego fragmentu kodu

Wskazówka

Fragmenty kodu można przeglądać i udostępniać także przez sieć (informacje o społeczności internetowej znajdują się w rozdziale 7., „Społeczność .NET: wykorzystanie i tworzenie współdzielonego kodu"). Doskonałym sposobem na lepsze zrozumienie struktury i działania fragmentów kodu jest zapoznanie się z plikami fragmentów dostępnych w Visual Studio, a także plikami utworzonymi przez społeczność programistów.

Fragmenty kodu w oknie narzędzi

Choć ta możliwość technicznie nie jest częścią oficjalnej technologii obsługi fragmentów kodu środowiska Visual Studio, można dodawać takie fragmenty także do okna narzędzi. Najpierw należy zaznaczyć tekst w edytorze, a potem upuścić go w oknie narzędzi. Następnie można wykorzystać dany fragment w dowolnym momencie, przeciągając go z okna narzędzi na otwarte okno edytora.

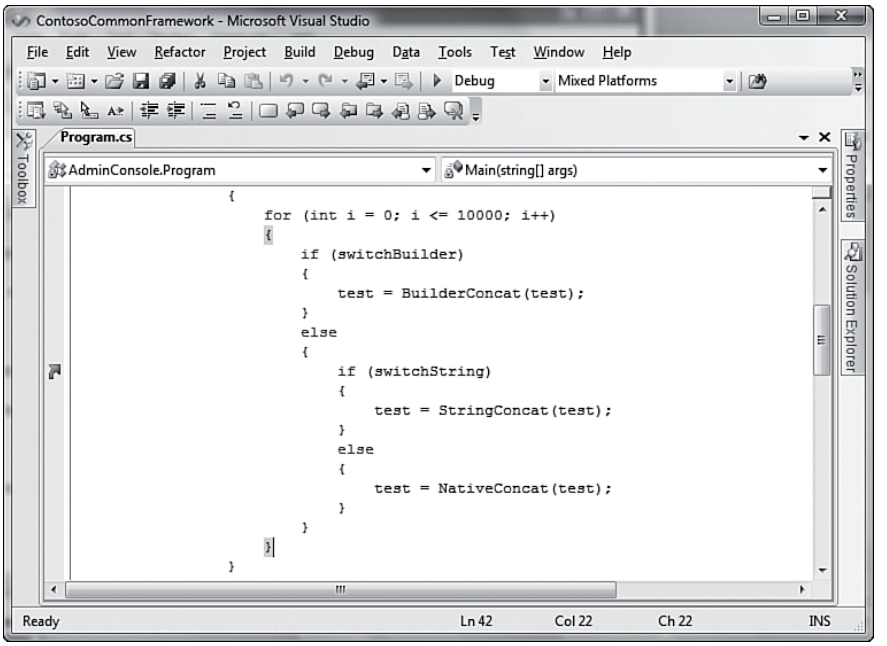
Dopasowywanie nawiasów

W językach programowania stosowane są nawiasy, nawiasy klamrowe, nawiasy ostre i inne ograniczniki, które obejmują argumenty funkcji, wyznaczają kolejność wykonywania funkcji matematycznych czy zawierają kod ciała funkcji. Wykrycie braku pasującego ogranicznika (czyli sytuacja, w której w kodzie jest więcej otwierających nawiasów niż zamykających) może być trudne, zwłaszcza w przypadku kodu z wieloma poziomami zagnieżdżenia.

Dopasowywanie nawiasów dotyczy graficznych wskazówek, za pomocą których edytor kodu informuje programistę o pasujących ogranicznikach. Gdy programista pisze kod w edytorze, za każdym razem, kiedy doda zamykający ogranicznik, środowisko na krótko wyróżnia oba nawiasy. Na rysunku 8.22 mechanizm dopasowywania nawiasów pomaga określić pasujące do siebie ograniczniki wewnętrznej pętli `for`.

Wskazówka

Można także uruchomić dopasowywanie nawiasów, umieszczając kursor bezpośrednio po lewej stronie otwierającego ogranicznika lub po prawej stronie zamykającego. Jeśli programista przegląda procedurę pełną nawiasów i nawiasów klamrowych, może szybko wykryć pasujące pary, umieszczając kursor obok różnych ograniczników.



Rysunek 8.22. Dopasowywanie nawiasów

Choć ta właściwość nazywana jest dopasowywaniem nawiasów, obsługuje różne rodzaje ograniczników:

- Nawiasy — `()`.
- Nawiasy kwadratowe i ostre — `[]`, `<>`.
- Cudzysłowy — `""`.
- Nawiasy klamrowe — `{}`.

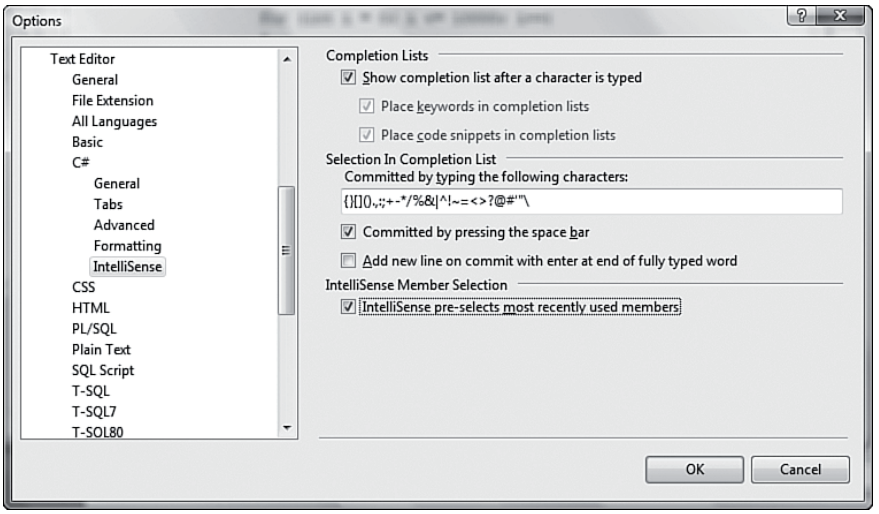
W języku C# dopasowywanie nawiasów obsługuje także następujące pary słów kluczowych (służą one do ograniczania bloków kodu za pomocą słów kluczowych):

- `#region`, `#endregion`.
- `#if`, `#else`, `#endif`.
- `case`, `break`.
- `default`, `break`.
- `for`, `break`, `continue`.
- `if`, `else`.
- `while`, `break`, `continue`.

Dostosowywanie mechanizmu IntelliSense do własnych potrzeb

Niektóre właściwości IntelliSense można dostosować do własnych potrzeb za pomocą okna dialogowego *Options* środowiska Visual Studio. Te modyfikacje dotyczą poszczególnych języków. Po otwarciu okna dialogowego *Options* (z menu *Tools*) i przejściu do węzła *Text Editor* widoczne są opcje mechanizmu IntelliSense w mylący sposób rozproszone między strony *General* i *IntelliSense*.

Rysunek 8.23 przedstawia stronę *IntelliSense* okna dialogowego *Options* z opcjami dla języka C#.



Rysunek 8.23. Opcje mechanizmu IntelliSense

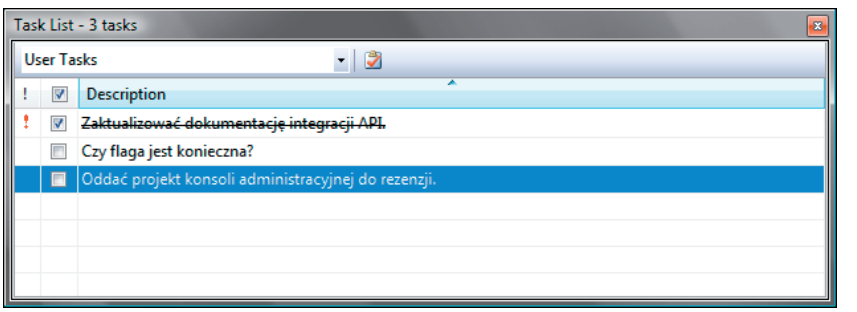
Sekcja *Completion Lists* w tym oknie dotyczy wszelkich właściwości mechanizmu IntelliSense, które obsługują automatyczne uzupełnianie kodu (na przykład wybierania składowych i listy do uzupełniania słów). W tabeli 8.4 opisane są opcje dostępne w tym oknie dialogowym.

Okno Task List

Lista w oknie *Task List* to zintegrowana lista zadań do wykonania. Obejmuje ona wszystkie elementy, które z jakiegoś powodu wymagają uwagi lub kontroli. Okno *Task List* wyświetla tę listę i pozwala wchodzić z nią w interakcje. Aby wyświetlić to okno, należy otworzyć menu *View* i wybrać opcję *Task List*. Rysunek 8.24 przedstawia okno *Task List* wyświetlające kilka zadań. Te zadania mogą należeć do jednej z trzech kategorii: zadania związane z komentarzami, zadania związane ze skrótami i zadania użytkownika. Jednocześnie można wyświetlać zadania z tylko jednej kategorii.

Tabela 8.4. Opcje mechanizmu IntelliSense

Opcja	Działanie
Show Completion List After a Character Is Typed	Zaznaczenie tego pola powoduje, że funkcja uzupełniania słów jest automatycznie uruchamiana po wpisaniu znaku w oknie edytora.
Place Keywords in Completion Lists	Jeśli to pole jest zaznaczone, na liście uzupełniania widoczne będą słowa kluczowe danego języka. Na przykład w języku C# na tej liście znajdą się słowa kluczowe `class` czy `string`.
Place Code Snippets in Completion Lists	Zaznaczenie tego pola powoduje dodanie aliasów fragmentów kodu do wyświetlanych list uzupełniania.
Committed by Typing the Following Characters	To pole zawiera znaki, które powodują uzupełnienie tekstu przez mechanizm IntelliSense. Inaczej mówiąc, wpisanie dowolnego ze znaków z tego pola tekstowego w czasie, kiedy widoczna jest lista uzupełniania, powoduje wstawienie bieżącego wyboru w oknie edytora.
Committed by Pressing the Space Bar	Jeśli to pole jest zaznaczone, wciśnięcie spacji powoduje wstawienie bieżącego wyboru.
Add New Line on Commit with Enter at End of Fully Typed Word	Zaznaczenie tego pola powoduje przesunięcie kursora do następnego wiersza po wpisaniu całego słowa w polu listy IntelliSense. Jest to przydatne wtedy, jeśli po danych słowach kluczowych zwykle nie występuje w wierszu dalszy kod.
IntelliSense Pre-selects Most Recently Used Members	Jeśli to pole jest zaznaczone, IntelliSense będzie przechowywać listę najczęściej używanych składowych poszczególnych typów i korzystać z niej. Ta lista służy do wybierania domyślnych składowych na liście uzupełniania.



Rysunek 8.24. Okno Task List

Lista rozwijana widoczna u góry okna *Task List* pozwala określić kategorię wyświetlanych zadań. Każda kategoria wiąże się z nieco innymi kolumnami, choć kolumny *Priority* i *Description* są widoczne w każdym przypadku. Na przykład zadania użytkownika i związane ze skrótami udostępniają pole wyboru, które służy do zaznaczania zakończonych zadań, a zadaniom związanym ze skrótami i komentarzami towarzyszą nazwa pliku i numer wiersza.

Można posortować zadania według dowolnej kolumny listy. Kliknięcie prawym przyciskiem myszy nagłówka kolumny powoduje wyświetlenie menu podręcznego umożliwiającego zmianę sposobu sortowania, a także wybór wyświetlanych kolumn z listy wszystkich obsługiwanych.

Zadania związane z komentarzami

Zadania związane z komentarzami są tworzone w kodzie źródłowym. Umieszczenie komentarza ze specjalnym literałem (znacznikiem) powoduje dodanie go przez Visual Studio do listy zadań związanych z komentarzami. W Visual Studio dostępne są trzy takie znaczniki: `HACK`, `TODO` i `UNDONE`. Aby wyświetlić je w oknie zadań, należy zaznaczyć opcję *Comments* na liście rozwijanej w górnej części listy zadań.

Na przykład poniższy kod języka C# powoduje dodanie do listy czterech różnych zadań związanych z komentarzami:

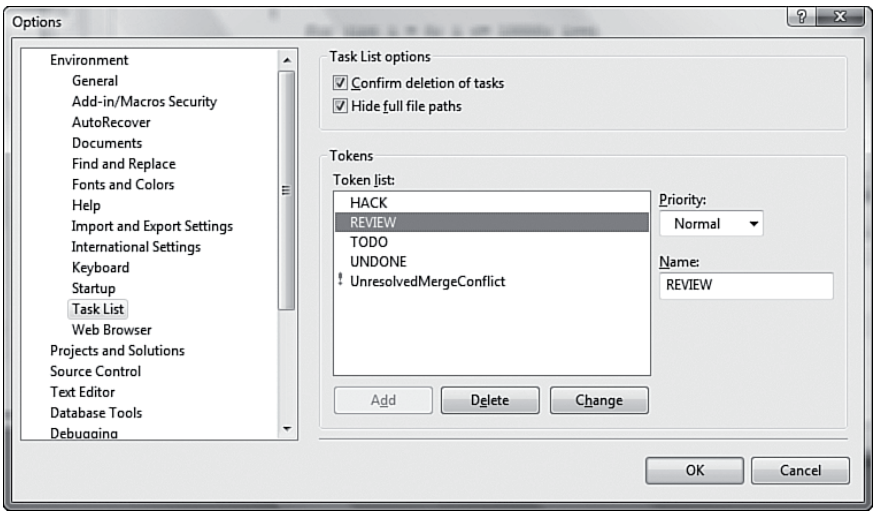
```
namespace Contoso.Fx.Integration.Specialized
{
   //TODO: Zaimplementować drugi konstruktor
   public class MessageMapper : IMessageSink
   {
      public MessageMapper()
      {
      }
   }

   //TODO: Sprawdzić implementację interfejsu IMap
   public class MessageBus : MessageMapper
   {
      public MessageBus()
      {
         //UNDONE: Konstruktor MessageBus
      }
   }

   //HACK: Napisać od nowa TokenStruct
   public class ContextToken
   {
      public ContextToken()
      {
      }
      public ContextToken(string guid)
      {
      }
   }
}
```

Dwukrotne kliknięcie zadania związanego z komentarzem pozwala przejść bezpośrednio do wiersza danego komentarza w oknie edytora.

Niestandardowe znaczniki w komentarzach

W razie potrzeby można dodać własny zestaw znaczników rozpoznawanych jako zadania związane z komentarzami. W oknie dialogowym *Tools/Options* należy otworzyć stronę *Task List* znajdującą się w węźle *Environment*. W tym miejscu można dodawać, modyfikować i usuwać znaczniki komentarzy obsługiwane na liście zadań.

Na rysunku 8.25 widoczny jest znacznik `REVIEW` dodany do standardowej listy. Warto zauważyć, że można także ustawić priorytety dla tych znaczników lub doprecyzować sposób ich wyświetlania, używając pól wyboru na stronie *Task List* okna *Options*. Te pola pozwalają określić, czy usuwanie zadań wymaga potwierdzenia, czy nie, a także czy na liście wyświetlane są same nazwy plików, czy pełne ścieżki.



Rysunek 8.25. Dodawanie niestandardowego znacznika zadań związanych z komentarzami

Zadania związane ze skrótami

Zadania związane ze skrótami to odnośniki do wierszy kodu. Te odnośniki można dodać za pomocą menu *Bookmarks*. W tym celu należy umieścić kursor w odpowiednim wierszu i wybrać opcję *Edit/Bookmarks/Add Task List Shortcut*. Zawartość tego wiersza kodu będzie widoczna jako opis zadania.

Dwukrotne kliknięcie zadania powoduje bezpośrednie przejście do odpowiedniego wiersza edytora kodu.

Zadania użytkownika

Zadania użytkownika są wpisywane bezpośrednio w oknie zadań. Obok listy rozwijanej z kategoriami zadań okna *Task List* widoczny jest przycisk *Create User Task* (rysunek 8.24). Ten przycisk pozwala dodać nowy wpis do listy. Następnie można bezpośrednio wpisać tekst w kolumnie opisu w celu określenia tytułu zadania.

W przeciwieństwie do zadań związanych ze skrótami i komentarzami zadania użytkownika nie prowadzą do specyficznych wierszy kodu.

Uwaga

Model automatyzacji środowiska Visual Studio umożliwia pełną kontrolę nad listą zadań. Używając dostępnych obiektów automatyzacji, takich jak `TaskList` czy `TaskListEvents`, można na przykład programowo dodawać i usuwać zadania z listy, reagować na dodanie, modyfikację, a nawet wybór zadania, czy kontrolować powiązania między zadaniem a edytorem.

Podsumowanie

Visual Studio 2008 udostępnia niezwykłą liczbę funkcji zwiększających produktywność programistów. Ten rozdział opisuje wiele aspektów technologii IntelliSense: od uzupełniania instrukcji, po nową technologię wstawiania fragmentów kodu. Pokazaliśmy także, jak korzystać z różnych elementów IntelliSense w celu szybszego pisania kodu oraz poprawy jego jakości.

Opisaliśmy tu techniki nawigowania po skomplikowanych i zagmatwanych plikach z kodem oraz przeglądania ich.

Przedstawiliśmy także fragmenty kodu oraz opisaliśmy różne ich typy oraz zastosowania.

Na zakończenie pokazaliśmy, jak używać okna *Task List*, aby wykorzystać wszystkie jego możliwości, a także jak organizować i śledzić różne zadania do wykonania obecne w każdym projekcie programistycznym.

Z punktu widzenia produktywności Visual Studio 2008 to coś więcej niż suma jego części. Dzięki synergii między poszczególnymi elementami środowiska każda z właściwości znosi bariery i łagodzi bolączki programistów niezależnie od ich wcześniejszych doświadczeń, poziomu biegłości czy preferowanego języka.