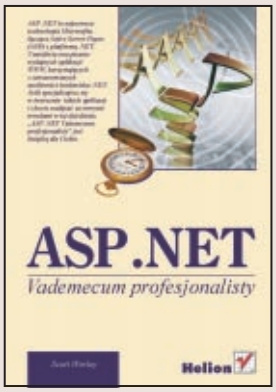
ASP .NET. Vademecum profesjonalisty



Autor: Scott Worley
Tłumaczenie: Paweł Janociński
ISBN: 83-7197-691-7
Tytuł oryginału: Inside ASP.NET
Format: B5, stron: 482
Przykłady na ftp: 385 kB

ASP .NET to najnowsza technologia Microsoftu, będąca łącząca Active Server Pages (ASP) z platformą .NET. Umożliwia ona pisanie wydajnych aplikacji WWW, korzystających z zawansowanych możliwości środowiska .NET.

Jeśli specjalizujesz się w tworzeniu takich aplikacji i chcesz nadążać za nowymi trendami w tej dziedzinie, „ASP .NET Vademecum profesjonalisty" jest książką dla Ciebie. Znajdziesz tu w zwięzłej i przystępnej formie kompletny opis ASP .NET, a także innych powiązanych z nią technologii.

Książka opisuje między innymi:

- Klasy bazowe i podstawowe obiekty ASP .NET
- Szczegółowe wskazówki dotyczące projektowania i konfigurowania aplikacji ASP .NET
- Nową technologię tworzenia interfejsu użytkownika: WebForms i zaawansowane aspekty jej użycia
- Dostęp do danych za pomocą ADO .NET
- Użycie języka XML w połączeniu z ASP .NET
- Tworzenie usług sieciowych (web services), użycie protokołów SOAP i UDDI
- Model bezpieczeństwa aplikacji ASP .NET
- Obsługiwanie wiadomości
- Użycie usług katalogowych Active Directory
- Programowanie urządzeń przenośnych w ASP .NET

Książkę uzupełniają dodatki omawiające architekturę platformy .NET, najczęściej używane obiekty ASP .NET i ADO – kontrolki serwera i Microsoft Mobile Internet Toolkit. Zawarta jest w niej także przykładowa kompletna aplikacja, w praktyczny sposób ilustrująca działanie ASP .NET.

Jeśli programowałeś wcześniej w ASP, „ASP .NET. Vademecum profesjonalisty" to jedyna książka, która jest Ci potrzebna, by w pełni wykorzystać możliwości tej technologii. To książka napisana przez profesjonalistów dla profesjonalistów. To książka dla Ciebie.

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

Spis treści

Rozdział 2.

Projektowanie aplikacji ASP.NET

Ten rozdział wprowadza kilka tematów związanych z tworzeniem aplikacji ASP.NET. Są to:

- Pliki ustawień aplikacji
- Składnia stron WWW
- Najczęściej stosowane obiekty i klasy w ASP.NET
- Funkcje śledzenia
- Sprawy dotyczące migracji do ASP.NET

Pliki ustawień aplikacji

Aplikacje ASP.NET mogą być konfigurowane za pomocą dwóch mechanizmów, opartych na plikach:

- *global.asax* — plik globalnych ustawień aplikacji.
- *web.config* — plik w formacie XML zawierający konfigurację aplikacji.

Plik global.asax

Plik *global.asax* służy dwóm głównym celom:

- Umożliwia definiowanie zmiennych, obiektów i danych o zasięgu aplikacji i sesji.
- Umożliwia definiowanie procedur obsługi zdarzeń, które mają miejsce w zakresie aplikacji, na poziomie sesji i aplikacji.

Dyrektywy stosowane w pliku global.asax

Plik *global.asax* może zawierać następujące elementy:

- dyrektywę `@Assembly` — stosowana do rejestrowania skompilowanych zbiorów zawierających aplikacje WWW; innym sposobem rejestrowania zbiorów jest użycie pliku *web.config*, który jest omówiony później w podrozdziale „Plik konfiguracyjny web.config”

```
<%@ Assembly Name="MyAssembly.dll" %>
```

- dyrektywę `@Import` — importuje jawnie do aplikacji przestrzeń nazw. Wszystkie klasy i interfejsy zdefiniowane w tej przestrzeni stają się dostępne dla aplikacji

```
01 <%@ Import Namespace="System.IO" %>
02 <%@ Import Namespace="System.Collections" %>
```

- dyrektywę `@Application` — definiuje specyficzną dla aplikacji informację używaną przez aplikację ASP.NET; jedyną funkcją tej dyrektywy jest dziedziczenie właściwości innego obiektu. Ma dwa atrybuty — pierwszy to `Inherits`, gdzie podaje się nazwę klasy, którą rozszerzamy, zaś drugi — `Description`, zawiera tekstowy opis aplikacji (opis ten jest ignorowany zarówno przez parser, jak i kompilator, programista może go jednak użyć do własnych celów).

```
<%@ Application Inherits="myClass" Description="Description" %>
```

Zmienne o zakresie aplikacji lub sesji

Zmienne o zakresie aplikacji mogą być tworzone za pomocą obiektów `Application`, zaś te o zakresie sesji za pomocą obiektu `Session`. Dokładniejszy opis tych obiektów i ich pozostałych zastosowań znajduje się w dalszej części tego rozdziału.

Aby zadeklarować zmienną:

```
01 Application("myApplicationScopeVar") = "MyValue"
02 Session("MySessionScopeVar") = "MyValue"
```

Aby pobrać wartość zmiennej:

```
01 MyValue = Application("myApplicationScopeVar")
02 MyValue = Session("MySessionScopeVar")
```

Obiekty statyczne i komponenty COM/COM+ mogą być deklarowane w pliku *global.asax* za pomocą znacznika `Object`. Takie obiekty i komponenty mogą mieć zakres sesji lub aplikacji.

Zakres `pipeline` określa, że obiekt lub komponent należą do bieżącej instancji `HttpApplication` i nie są współdzielone.

```
01 <Object id="id" runat="server" class="myClassName" scope="pipeline">
02 <Object id="id" runat="server" progid="myCOM-ProgID" scope="session"/>
03 <Object id="id" runat="server" classid="myCOM-ClassID" scope="application"/>
```

Zdarzenia sesji

W pliku *global.asax* można zdefiniować obsługę następujących zdarzeń sesji:

- `Session_OnStart` — zdarzenie generowane, gdy klient rozpoczyna sesję,
- `Session_OnEnd` — zdarzenie generowane, gdy klient zamyka sesję.

Bloki skryptu

W pliku *global.asax* można używać także bloków skryptu. Generalnie jednak bloki skryptu w *global.asax* służą do deklarowania zmiennych aplikacji i sesji oraz do obsługi zdarzeń dla platformy .NET. Przeanalizujcie przykład pliku *global.asax* umieszczony na wydruku 2.1.

Wprowadzanie zmian do pliku global.asax

Gdy zmienia się aktywny plik *global.asax*, ASP.NET wykrywa zmianę pliku, kończy obsługę wszystkich bieżących żądań, generuje zdarzenie `Application_OnEnd` do wszystkich odbiorców i restartuje aplikację.

W efekcie następuje zresetowanie aplikacji WWW lub witryny, zamknięcie wszystkich sesji i wyczyszczenie wszystkich informacji o stanie.

Gdy nadchodzi nowe żądanie, ASP.NET kompiluje ponownie plik *global.asax* i generuje zdarzenie `Application_OnStart`.

Z oczywistych powodów zmian w pliku *global.asax* należy dokonywać, gdy aplikacja WWW jest w okresie braku lub niewielkiego poziomu ruchu.

Poniżej, na wydruku 2.1 przedstawiono przykład pliku *global.asax*.

Wydruk 2.1. *Przykładowy plik global.asax*

```
<Script language="VB" runat="server">


 'Kod wykonywany na starcie aplikacji
 Sub Application_OnStart()
   Application("MyApplicationScopeVar") = "MyValue"
   Session("MySessionScopeVar") = "MyValue"
 End Sub

 'Kod sprzątający po aplikacji
 Sub Application_OnEnd()
 End Sub

 'Kod wykonywany na początku sesji
 Sub Session_OnStart()
 End Sub

 'Kod sprzątający po sesji
 Sub Session_OnEnd()
 End Sub


</script>
```

Wprowadzanie zmian w pliku global.asax

Zmiany w plikach konfiguracyjnych ASP.NET są automatycznie wykrywane przez system i natychmiast stosowane.

Kluczowe informacje o pliku global.asax

Plik *global.asax* musi znajdować się w katalogu bazowym witryny lub aplikacji WWW.

Aplikacja WWW czy witryna mogą mieć tylko jeden plik *global.asax*.

Plik *global.asax* jest kompilowany przez ASP.NET do klasy (dziedziczącej po `HttpApplication`) w momencie pierwszej aktywacji zasobów czy żądania URL należącego do aplikacji.

Plik *global.asax* jest skonfigurowany tak, że bezpośrednie żądanie URL dotyczące tego pliku jest automatycznie odrzucane; blokuje to nieautoryzowany dostęp do pliku *global.asax*.

Plik konfiguracyjny web.config

Informacje konfiguracyjne ASP.NET są przechowywane w pliku konfiguracyjnym w formacie XML. Oznacza to, że użytkownik może go przeglądać i edytować. Dzięki temu administratorzy i programiści mogą w łatwy sposób zmieniać ustawienia konfiguracji aplikacji WWW czy witryny. Programiści nie muszą czekać, aż administrator zmieni ustawienia IIS (*Internet Information Server*) i uruchomi ponownie serwer WWW, aby zmiany odniosły skutek. Jest więc oczywiste, że zwiększa to wydajność pracy programisty.

System konfiguracji jest w pełni rozszerzalny, ponieważ daje możliwość umieszczania własnych danych konfiguracyjnych dla własnych aplikacji WWW i witryn w zaprojektowanym dla nich pliku *web.config*. Hierarchiczna infrastruktura konfiguracji pozwala na definiowanie i używanie w ASP.NET dodatkowych danych konfiguracyjnych. ASP.NET zawiera również bogaty zestaw wstępnych ustawień konfiguracyjnych.

Szczegółowy opis pliku *web.config* znajduje się w rozdziale 3., „Konfigurowanie aplikacji ASP.NET”.

Składnia strony

Przy programowaniu form WWW ważna jest znajomość składni strony w pliku ASP.NET. W ASP.NET istnieje dziesięć różnych elementów składni. Zostały one omówione poniżej.

Dyrektywy strony

Dyrektywy strony pozwalają na podanie opcjonalnych ustawień używanych przez kompilator stron podczas przetwarzania plików ASP.NET. Mogą one być umieszczone w dowolnym miejscu pliku *.aspx*. Każda dyrektywa może zawierać jeden lub więcej atrybutów lub par wartości specyficznych dla tej dyrektywy.

Ogólna składnia dyrektywy strony ma postać:

```
<%@ directive attribute=value [attribute=value ... ]%>
```

Formy WWW obsługują następujące dyrektywy:

- `@ Page`
- `@ Control`

- `@ Import`
- `@ Register`
- `@ Assembly`
- `@ OutputCache`

Dyrektywa @ Page

Dyrektywa `@ Page` definiuje atrybuty strony stosowane przez parser i kompilator stron ASP.NET. Ma ona następującą składnię:

```
<%@ Page attribute=value [attribute=value ... ]%>
```

W pliku *.aspx* może znajdować się tylko jedna dyrektywa `@ Page`. Aby zdefiniować wiele atrybutów, należy wymienić je, oddzielając spacjami, bez spacji przy znaku równości, jak w `TRACE="True"`.

Dyrektywa `@ Page` ma następujące atrybuty:

- `AspCompat` — określa, czy strona jest kompatybilna wstecz z ASP. Jeśli nie jest ustawiony na `true`, biblioteki DLL ActiveX napisane dla ASP nie będą działać. Atrybut ten należy ustawić na `true`, jeśli strona musi być wykonywana w wątku STA lub wywołuje komponent COM+ 1.0 wymagający dostępu do wbudowanych cech ASP przez kontekst obiektu lub przez `OnStartPage`. Wartością domyślną jest `false`.
- `Buffer` — definiuje semantykę buforowania odpowiedzi HTTP. Jeśli buforowanie ma być dostępne, ma wartość `true`, w przeciwnym wypadku `false`. Wartością domyślną jest `true`.
- `CodePage` — określa wartość strony kodowej dla strony ASP.NET. Obsługuje wszystkie prawidłowe wartości strony kodowej.
- `ContentType` — definiuje typ zawartości HTTP w odpowiedzi, zgodny z typami MIME (*Multipurpose Internet Mail Extension*). Obsługuje wszystkie wartości definiujące prawidłowe typy zawartości HTTP.
- `Culture` — wskazuje ustawienia kulturowe dla strony. Obsługuje wszystkie wartości definiujące prawidłowe ustawienia kulturowe.
- `Description` — dostarcza tekstowego opisu strony. Może zawierać dowolny tekst.
- `EnableViewState` — wskazuje, czy informacja o właściwościach strony ma być przechowywana pomiędzy żądaniami strony. Jeśli ma wartość `true` — informacja będzie przechowywana; jeśli `false` — nie będzie. Wartością domyślną jest `true`.
- `EnableSessionState` — definiuje wymagania stanu sesji dla strony. Jeśli `true`, stan sesji jest dostępny, `ReadOnly` określa, że stan sesji może być czytany, ale nie zmieniany, w innych przypadkach `false`. Wartością domyślną jest `true`.
- `ErrorPage` — definiuje docelowy URL dla przekierowania, jeśli wystąpi błąd braku strony.
- `Inherits` — zewnętrzna klasa, po której strona dziedziczy. Może to być dowolna klasa dziedzicząca po klasie `Page`.

- `Language` — język stosowany do kompilacji wszystkich bloków `<% %>` i `<%= %>` wewnątrz strony. Może to być Visual Basic, C# lub JScript.
- `Local identifier (LCID)` — definiuje identyfikator lokalizacji dla kodu strony.
- `ResponseEncoding` — kodowanie zawartości zwracanej strony. Obsługuje wartości z `Encoding.GetEncoding`.
- `SRC` — zewnętrzna klasa do skompilowania (nazwa pliku źródłowego).
- `Trace` — wskazuje, czy śledzenie jest włączone. `true`, jeśli śledzenie jest dostępne, w przeciwnym wypadku `false`. Wartością domyślną jest `false`.
- `Transaction` — wskazuje, czy strona obsługuje transakcje. Możliwymi wartościami są `NotSupported`, `Supported`, `Required` i `RequiresNew`.
- `WarningLevel` — definiuje poziom ostrzegawczy kompilatora, którego osiągnięcie powoduje przerwanie kompilacji strony. Możliwe są wartości od `0` do `4`.

Dyrektywa @ Control

Jako dodatek do istniejących kontrolek HTML i serwera WWW, można zdefiniować *kontrolki użytkownika* (*user controls*). Tworzy się je przy użyciu tych samych technik, które służą do projektowania stron za pomocą form WWW.

Kontrolki użytkownika oferują łatwy sposób na podzielenie i wielokrotne wykorzystywanie najpowszechniejszych funkcji interfejsu użytkownika w różnych aplikacjach ASP.NET. Podobnie jak w przypadku form WWW można je tworzyć za pomocą edytora tekstowego lub projektować z zastosowaniem gotowych klas. Analogicznie jak w przypadku form, kontrolki użytkownika są kompilowane w momencie wystąpienia pierwszego żądania, które ich dotyczy, i przechowywane w pamięci serwera, co pozwala skrócić czas reakcji przy następnych żądaniach. W przeciwieństwie do form WWW kontrolki użytkownika nie mogą być wywoływane samodzielnie; aby działać, muszą być umieszczone w stronie formy WWW.

Szersze omówienie kontrolek użytkownika można znaleźć w rozdziale 15., „Tworzenie kontrolek użytkownika i kontrolek dostosowanych dla ASP.NET”.

Dyrektywa `@ Control` pozwala na zdefiniowanie atrybutów kontrolki stosowanych przez parser stron ASP.NET i kompilator. Dyrektywę tę można stosować tylko z kontrolkami użytkownika. Ma ona następującą składnię:

```
<%@ Control attribute=value [attribute=value ...] %>
```

Dyrektywa `@ Control` obsługuje te same atrybuty co `@ Page`, oprócz `AspCompat` i `Trace`. Aby umożliwić śledzenie, należy zdefiniować atrybut `Trace` dla dyrektywy `@ Page` w pliku definiującym formę WWW zawierającą kontrolkę użytkownika. W jednym pliku *.ascx* może się znajdować tylko jedna dyrektywa `@ Control`.

Dyrektywa `@ Control` ma następujące atrybuty:

- `AutoEventWireup` — wskazuje, czy zdarzenia strony są powiązane z tą stroną. `true` oznacza możliwość powiązania, `false` przeciwny przypadek. Wartością domyślną jest `true`.

- `ClassName` — pozwala na podanie nazwy klasy automatycznie kompilowanej dynamicznie podczas wystąpienia żądania strony. Wartością może być dowolna prawidłowa nazwa klasy.
- `CompilerOptions` — łańcuch znaków zawierający opcje kompilacji dla strony.
- `Debug` — określa, czy strona ma być kompilowana z symbolami debuggera. `true` oznacza włączenie tych symboli, `false` — wyłączenie.
- `Description` — dostarcza tekstowego opisu strony. Może zawierać dowolny tekst.
- `EnableSessionState` — definiuje dla strony wymagania dotyczące stanu sesji. Wartość `true` oznacza, że stan sesji jest dostępny, wartość `ReadOnly` — że stan sesji może być odczytywany, ale nie modyfikowany, zaś wartość `false` odpowiada pozostałym przypadkom. Wartością domyślną jest `true`.
- `Explicit` — określa, że strona powinna być kompilowana w trybie Visual Basic Option Explicit. `true` oznacza włączenie tej opcji, `false` — wyłączenie. Wartością domyślną jest `false`.
- `Inherits` — zewnętrzna klasa, po której strona dziedziczy. Może to być dowolna klasa dziedzicząca po klasie `Page`.
- `Language` — język stosowany do kompilacji wszystkich bloków `<% %>` i `<%= %>` wewnątrz strony. Może to być Visual Basic, C# lub JScript.NET.
- `Strict` — określa, że strona powinna być kompilowana w trybie Visual Basic Option Strict. `true` oznacza włączenie tej opcji, `false` — wyłączenie. Wartością domyślną jest `false`.
- `SRC` — zewnętrzna klasa do skompilowania (nazwa pliku źródłowego).
- `WarningLevel` — definiuje poziom ostrzegawczy kompilatora, którego osiągnięcie powoduje przerwanie kompilacji kontrolki użytkownika. Możliwe są wartości od `0` do `4`.

Dyrektywa @ Import

Dyrektywa `@ Import` dołącza jawnie do strony przestrzeń nazw, co umożliwia elementom strony dostęp do wszystkich klas i interfejsów zdefiniowanych w tej przestrzeni. Importowana przestrzeń może być częścią biblioteki klas platformy .NET lub przestrzenią zdefiniowaną przez użytkownika.

Dyrektywa `@ Import` ma następującą składnię:

```
<%@ Import namespace="the namespace to be imported" %>
```

Dyrektywa `@ Import` ma tylko jeden atrybut, `namespace`, który zawiera nazwę przestrzeni nazw importowanej do strony.

Zwróćcie uwagę na fakt, że dyrektywa `@ Import` ma tylko jeden atrybut `namespace`. Aby zaimportować większą liczbę przestrzeni nazw, należy użyć wielu dyrektyw `@ Import`.

Poniższe przestrzenie nazw są automatycznie importowane do wszystkich stron:

- `System`
- `System.Collections`
- `System.Collections.Specialized`
- `System.Configuration`
- `System.IO`
- `System.Text`
- `System.Text.RegularExpressions`
- `System.Web`
- `System.Web.Caching`
- `System.Web.Security`
- `System.Web.SessionState`
- `System.Web.UI`
- `System.Web.UI.HtmlControls`
- `System.Web.UI.WebControls`

Poniższy przykładowy kod przedstawia zaimportowanie przestrzeni nazw klas bazowych platformy .NET, `System.Net` i zdefiniowanej przez użytkownika przestrzeni `WebTools`:

```
01 <%@ Import Namespace="System.Net" %>
02 <%@ Import Namespace="WebTools" %>
```

Dyrektywa @ Register

Dyrektywa `@ Register` pozwala na określenie aliasów przestrzeni nazw i nazw klas dla skrócenia notacji składni własnych kontrolek serwera. Dyrektywa ta ma następującą składnię:

```
01 <%@ Register Tagprefix="tagprefix" Namespace="namespace" %>
02
03 <%@ Register Tagprefix="tagprefix" Tagname="tagname" Src="pathname" %>
```

Dyrektywa `@ Register` ma następujące atrybuty:

- `Tagprefix` — alias powiązany z przestrzenią nazw;
- `Tagname` — alias powiązany z klasą;
- `Namespace` — przestrzeń nazw powiązana z `tagprefix`;
- `SRC` — ścieżka (względna lub bezwzględna) kontrolki użytkownika powiązanej z `tagprefix:tagname`.

Poniższy przykładowy fragment kodu ilustruje zastosowanie dyrektyw `<% Register %>` do zadeklarowania aliasów `tagprefix` i `tagname` dla kontrolek serwera i użytkownika. Pierwsza dyrektywa deklaruje alias `MyTag` jako prefiksu dla wszystkich kontrolek znajdujących się w przestrzeni nazw `MyCompany:MyNameSpace`. Druga dyrektywa deklaruje

`Acme:AdRotator` jako parę `tagprefix:tagname` dla kontrolek użytkownika znajdujących się w pliku *adrotator.ascx*. Aliasy te są później użyte w składni kontrolek serwera wewnątrz formy do wstawiania instancji każdej z kontrolek serwera.

```
01 <%@ Register Tagprefix="MyTag" Namespace="MyCompany:MyNameSpace" %>
02 <%@ Register Tagprefix="Acme"
03   Tagname="AdRotator" Src="AdRotator.ascx" %>
04 <HTML>
05 ...
06   <form runat="server">
07      <MyTag:MyControl id="Control1" runat="server" /><BR>
08      <Acme:AdRotator file="myads.xml" runat="server" />
09   </form>
10
11 ...
12 </HTML>
```

Dyrektywa @ Assembly

Dyrektywa `@ Assembly` deklaruje stworzenie połączenia pomiędzy zbiorem a bieżącą stroną, co pozwala na wykorzystywanie na stronie klas i interfejsów zdefiniowanych w tym zbiorze. Dyrektywy tej można również użyć do rejestrowania zbiorów w pliku konfiguracyjnym, co umożliwi dostęp do nich w całej aplikacji.

Dyrektywa ta ma następującą składnię:

```
<%@ Assembly Name="assemblyname" %>
```

lub

```
<%@ Assembly Src="pathname" %>
```

Dyrektywa ta ma atrybuty `Name` i `Src`. `Name` oznacza łańcuch określający nazwę zbioru łączonego ze stroną. `Src` zawiera ścieżkę dostępu do pliku źródłowego, który ma być dynamicznie skompilowany i dołączony.

Zbiory znajdujące się w katalogu *\bin* aplikacji są automatycznie łączone ze stronami tej aplikacji. Takie zbiory nie wymagają dyrektywy `@ Assembly`.

Automatyczne łączenie może zostać wyłączone przez usunięcie z sekcji `<assembly>` pliku *web.config* następującego wiersza:

```
<add assembly="*"/>
```

Jako alternatywę użycia dyrektywy `<%@ Assembly %>`, zbiory można zarejestrować w pliku *web.config*, co spowoduje ich połączenie z całą aplikacją.

Poniższy fragment kodu ilustruje użycie dyrektywy `<%@ Assembly %>` do stworzenia połączenia z *MyAssembly.dll*:

```
<%@ Assembly Name="myassembly.dll" %>
```

Dyrektywa @ OutputCache

Dyrektywa `@ OutputCache` deklaruje sposób obsługi buforowania wyjścia dla strony. Składnia wygląda następująco:

```
01 <%@ OutputCache Duration="#ofseconds"
02 Location="Any | Client | Downstream | Server | None" VaryByControl="controlname"
03 VaryByCustom="browser | customstring" VaryByHeader="headers"
04 VaryByParam="parametername" %>
```

Dyrektywa ta ma następujące atrybuty:

- `Duration` — czas buforowania strony lub kontrolki użytkownika w sekundach. Ustawienie tego atrybutu dla strony lub kontrolki użytkownika ustala ustawienia wygasania odpowiedzi HTTP generowanej przez obiekt i automatycznie buforuje wyjście strony lub kontrolki użytkownika.

  Atrybut ten jest wymagany. Jego brak spowoduje zgłoszenie błędu parsera.

- `Location` — jedna z listy wartości OutputCacheLocation. Wartością domyślną jest `Any`.

  Uwaga: atrybut ten jest wymagany dla buforowania wyjścia strony ASP.NET lub kontrolki użytkownika. Jego brak spowoduje zgłoszenie błędu parsera.

- `VaryByCustom` — dowolny tekst reprezentujący własne wymagania dotyczące buforowania wyjścia. Jeśli atrybut ma wartość `browser`, buforowanie jest uzależnione od informacji o nazwie i wersji przeglądarki. Jeśli podano własny łańcuch, w pliku *global.asax* aplikacji należy nadpisać metodę `HttpApplication.GetVaryByCustomString`.

- `VaryByHeader` — lista oddzielonych przecinkami nagłówków HTTP stosowanych dla zróżnicowania buforowania wyjścia. Jeśli w tym atrybucie podano kilka nagłówków, bufor wyjścia zawiera różne wersje żądanego dokumentu dla każdego z podanych nagłówków.

  Warto zauważyć, że ustawienie atrybutu `VaryByHeader` umożliwia buforowanie we wszystkich buforach HTTP/1.1 a nie tylko w buforach ASP.NET. Atrybut ten nie jest obsługiwany przez dyrektywy `@ OutputCache` w kontrolkach użytkownika.

- `VaryByParam` — lista oddzielonych przecinkami łańcuchów różnicujących buforowanie wyjścia. Domyślnie łańcuchy te odpowiadają wartościom zapytania przesłanego przez atrybuty metody `GET` lub parametrom wysłanym przez metodę `POST`. Jeśli atrybut ten zawiera wiele parametrów, bufor wyjścia zawiera różne wersje żądanego dokumentu dla każdego podanego parametru. Możliwymi wartościami są `none`, * oraz wszystkie poprawne zapytania i nazwy parametru `POST`.

  Uwaga: atrybut ten jest konieczny do buforowania wyjścia stron ASP.NET i kontrolek użytkownika. Jego brak spowoduje zgłoszenie błędu parsera. Jeśli nie chcecie podać parametrów różnicujących buforowaną zawartość, należy ustawić wartość na `none`. Jeśli buforowanie wyjścia ma być uzależnione od wszystkich wartości parametrów, wartość należy ustawić na *.

- `VaryByControl` — lista oddzielonych przecinkami łańcuchów różnicujących buforowanie wyjścia. Łańcuchy te reprezentują pełne ścisłe nazwy właściwości kontrolek użytkownika. Jeśli atrybut ten zostanie zastosowany do kontrolki użytkownika, jej wyjście jest buforowane w zależności od wartości podanych właściwości.

```
<%@ OutputCache Duration="10" %>
```

Blok deklaracji

Fragmentem składni strony stosowanym niemal zawsze jest blok deklaracji. *Blok deklaracji* definiuje wewnętrzne zmienne i metody które są kompilowane podczas tworzenia dynamicznej klasy `Page` reprezentującej stronę ASP.NET.

```
01 <script runat="server" language="codelanguage" Src="pathname" >
02   Tutaj umieszczamy kod...
03 </script>
```

Blok deklaracji ma następujące atrybuty:

- `language` — podaje język używany w tym bloku deklaracji. Wartość może reprezentować dowolny kompatybilny z .NET język programowania, taki jak Visual Basic, C# czy JScript.NET. Jeśli nie podano języka, wartością domyślną jest język podany w dyrektywie `@ Page`. Jeśli także ta dyrektywa nie zawiera informacji o języku, domyślnie jest używany Visual Basic.
- `src` — podaje ścieżkę dostępu i nazwę pliku zawierającego skrypt do wczytania. Jeśli zastosowano ten atrybut, pozostały kod zawarty w bloku deklaracji jest ignorowany.

Blok odrysowania

Blok odrysowania definiuje kod lub wyrażenie wstawiane (inline), które będą wykonywane w momencie odrysowywania strony. Istnieją dwa style — kod i wyrażenie. Używając zwykłego stylu, można zdefiniować zwykły blok kodu lub blok kontroli przepływu. Drugi może być stosowany jako skrót dla wywołania `Response.Write`.

Kod wstawiany ma następującą składnię:

```
<% Inline code or expression %>
```

Warto zauważyć, że w bloku odrysowania nie może wystąpić jawnie sekwencja znaków `%>`. Może być ona zastosowana tylko do zakończenia tego bloku. Przykładowo, poniższy fragment kodu spowoduje błąd:

```
01 <%
02   Response.Write(" %>")
03 %>
```

Aby osiągnąć ten efekt można zbudować łańcuch znaków zawierający niedozwolony łańcuch:

```
01 <%
02   Dim s As String = "%" + ">"
03   Response.Write(s)
04 %>
```

Komentarze po stronie serwera

Komentarze po stronie serwera pozwalają na umieszczanie komentarzy wewnątrz plików *.aspx*. Jakikolwiek tekst zawarty pomiędzy znacznikami otwierającym i zamykającym komentarza, czy będzie to kod ASP.NET czy zwykły tekst, nie będzie przetwarzany przez serwer ani wypisany na wynikowej stronie.

```
<%- Anulowany kod lub inna zawartość -%>
```

Bloki komentarza po stronie serwera w ASP.NET mają te same zastosowania co bloki komentarza w językach programowania, łącznie z dokumentacją i testowaniem.

Wewnątrz bloków `<script runat="server"> </script>` i `<% %>` można stosować składnię komentarza języka programowania, w którym napisano ten kod.

Jeśli wewnątrz `<% %>` zostanie zastosowany komentarz serwera, kompilator zgłosi błąd.

Otwierający i zamykający znacznik komentarza mogą pojawiać się w tym samym wierszu kodu lub mogą być oddzielone wierszami anulowanego kodu.

Komentarze po stronie serwera nie mogą być zagnieżdżane.

Poniższy przykład ilustruje usunięcie kodu kontrolki definiującego przycisk HTML:

```
<%-
<button runat="server" id="MyButton" OnServerClick="MyButton_Click">
Click here for enlightement!
</button>
-%>
```

Składnia niestandardowych kontrolek serwera

Niestandardowe kontrolki serwera są najczęściej stosowane do enkapsulowania podobnych funkcji programistycznych. Można tworzyć własne niestandardowe kontrolki serwera lub używać kontrolek dostarczonych z platformą .NET.

Poniżej przedstawiono szczegółowo składnię stosowaną przy niestandardowych kontrolkach serwera.

Składnia deklaracji niestandardowych kontrolek Web Forms

Kontrolki Web Form mogą być deklarowane przez wymienienie tych elementów w pliku *.aspx*. Znacznik otwierający takiego elementu musi zawierać atrybut lub parę wartości `runat="server"`. Aby umożliwić programistyczne odwoływanie się do kontrolki, należy nadać jej unikalną wartość atrybutu `id`.

Aby umieścić w Web Form kontrolkę serwera (włączając kontrolki stworzone przez użytkownika i wszystkie kontrolki sieci WWW), należy zastosować następującą składnię:

```
<tagprefix:tagname id="OptionalID" attributename="value" attributename-
propertyname="value" eventname="eventhandlermethod" runat="server" />
```

lub

```
<tagprefix:tagname id="OptionalID" runat="server"> </tagprefix:tagname>
```

Możliwe są następujące atrybuty:

- `tagprefix` — alias pełnej nazwy przestrzeni nazw kontrolki. Aliasy kontrolek stworzonych przez użytkownika są deklarowane w dyrektywie `@ Register`.
- `tagname` — nazwa klasy zawierającą kontrolkę.

- `id` — unikalny identyfikator umożliwiający programistyczne odnoszenie się do kontrolki.
- `attributename` — nazwa atrybutu.
- `propertyvalue` — wartość przypisana do `attributename`.
- `propertyname` — nazwa definiowanej właściwości.
- `supropertyvalue` — wartość przypisana do `propertyname`.
- `eventname` — nazwa zdarzenia kontrolki.
- `eventhandlermethod` — nazwa metody obsługi zdarzenia zdefiniowanej w kodzie Web Form.

Przykładowo, poniższy kod wstawia kontrolkę sieciową `Text Box`:

```
<ASP:TextBox id="MyTextBox" runat="server" />
```

Wyrażenia wiążące dane

Wyrażenia wiążące dane tworzą połączenia pomiędzy kontrolkami serwera i źródłami danych. Wyrażenia wiążące dane mogą być umieszczane po stronie wartości atrybutu lub pary wartości w znaczniku otwierającym kontrolki Web Form lub w dowolnym miejscu strony.

Wyrażenie wiążące dane ma następującą składnię:

```
<tagprefix:tagname property=<%# databinding expression %> runat="server" />
```

lub

```
literal text <%# databinding expression %>
```

Możliwe są następujące parametry:

- `tagprefix` — alias pełnej nazwy przestrzeni nazw kontrolki. Aliasem dla kontrolki sieciowej jest `ASP`. Aliasy dla kontrolek stworzonych przez użytkownika są deklarowane za pomocą dyrektywy `@ Register`.
- `tagname` — nazwa klasy platformy .NET zawierającej kontrolkę.
- `property` — właściwość kontrolki, dla której deklarowane jest powiązanie danych.
- `expression` — dowolne wyrażenie spełniające wymienione poniżej wymagania.

Wszystkie wyrażenia wiążące dane, bez względu na miejsce ich umieszczenia, muszą być zawarte pomiędzy `<%#` a `%>`.

ASP.NET obsługuje hierarchiczny model wiązania danych, który umożliwia asocjacyjne łączenie właściwości kontrolek serwera i nadrzędnych źródeł danych. Każda właściwość kontrolki serwera może być powiązana z danymi. Właściwości kontrolek mogą być powiązane z dowolnymi publicznymi polami lub właściwościami zarówno na zawierającej je stronie, jak i w bezpośrednio wymienionym pojemniku.

Stosowanie DataBinder.Eval

Platforma Web Form obsługuje również statyczną metodę wykonującą wyrażenie wiążące dane i opcjonalnie formatującą wynik jako łańcuch tekstowy. Metoda `DataBinder.Eval` jest wygodna, ponieważ eliminuje konieczność jawnego rzutowania koniecznego dla skonwertowania wartości do konkretnego typu danych. Jest to szczególnie użyteczne dla list kontrolek należących do szablonu list, ponieważ często zarówno wiersz danych, jak i pole danych muszą być rzutowane.

W poniższym przykładzie liczba całkowita jest wyświetlana w notacji walutowej. Przy standardowej składni wiązania danych w ASP.NET należy najpierw zrzutować typ danych, aby otrzymać wartość pola `IntegerValue`. Wartość ta jest następnie argumentem metody `String.Format`:

```
<%# String.Format("{0:c}", ((DataRowView)Container.DataItem)["IntegerValue"]) %>
```

Powyższy kod może być złożony i trudny do zapamiętania. Dla kontrastu, `DataBinder.Eval` jest po prostu metodą z trzema argumentami: nazwą pojemnika zawierającego dane, nazwą pola danych i łańcuchem formatującym. W szablonach list, takich jak klasy `DataList`, `DataGrid` czy `Repeater`, nazwą pojemnika jest zawsze `Container.DataItem`. Innym pojemnikiem, który można stosować w `DataBinder.Eval`, jest `Page`.

```
<%# DataBinder.Eval(Container.DataItem, "IntegerValue", "{0:c}") %>
```

Argument zawierający łańcuch formatujący jest opcjonalny. Jeśli nie zostanie podany, `DataBinder.Eval` zwróci wartość obiektu `type`. Przykładowo:

```
<%# (bool)DataBinder.Eval(Container.DataItem, "BoolValue") %>
```

Składnia znaczników obiektów serwera

Znaczniki obiektów serwera deklarują i tworzą instancje obiektów COM i .NET. Składnia wygląda następująco:

```
<object id="id" runat=server class=".NET Framework Class Name">
<object id="id" runat=server class="COM ProgID"/>
<object id="id" runat=server class="COM ClassID"/>
```

Dostępne są następujące parametry:

- `Id. Unique name` — referencja do obiektu w późniejszym kodzie.
- `NET Framework class name` — identyfikuje nazwę klasy platformy .NET, której instancję tworzymy.
- `COM ProgID` — identyfikuje komponent COM, którego instancję tworzymy.
- `COM ClassID` — identyfikuje komponent COM, którego instancję tworzymy.

Gdy parser lub kompilator strony napotka znacznik obiektu serwera w pliku ASP.NET, generuje na stronie właściwość odczytu (*read*) lub zapisu (*write*), używając atrybutu `id` znacznika jako jej nazwy. Właściwość odczytu jest skonfigurowana tak, aby stworzyć instancję obiektu w momencie pierwszego użycia. Otrzymana instancja nie jest dodawana jako obiekt do hierarchicznego drzewa kontrolek serwera tej strony; jest traktowana jako deklaracja zmiennej nie należącej do interfejsu użytkownika.

Atrybuty `classid`, `progid` i `class` wykluczają się wzajemnie. Błędem jest użycie więcej niż jednego z tych atrybutów w jednym znaczniku obiektu serwera.

Poniższy kod prezentuje przykład zastosowania znacznika obiektu serwera:

```
01 <html>
02 <object id="MyDatabase" class="Microsoft.OLEDBAdaptor" runat"server"/>
03 <script language="VB" runat=server>
04 Sub Page_Load(Sender as Object, e as EventArgs)
05   Dim StockDetails as RecordSet
06   Set StockDetails = MyDatabase.Execute("DSN:money", "select * from stocks")
07 End Sub
08 </script>
09 </html>
```

Składnia dyrektywy dołączania po stronie serwera

Dyrektywy dołączania po stronie serwera wstawiają zawartość podanego pliku do strony ASP.NET.

```
<!-- #include pathtype = filename -->
```

Parametr *pathtype* podaje typ ścieżki do pliku *filename*. Może mieć wartość `File` lub `Virtual`.

Jeśli *pathtype* ma wartość `File`, nazwa pliku zawiera ścieżkę względem katalogu zawierającego dokument z dyrektywą `#include`. Włączany plik może być w tym samym katalogu lub w podkatalogu; nie może jednak znajdować się w katalogu powyżej zawierającego stronę z dyrektywą `#include`.

Jeśli *pathtype* ma wartość `Virtual`, nazwa pliku jest pełną wirtualną ścieżką do wirtualnego katalogu wewnątrz serwisu WWW.

Parametr *filename* podaje nazwę dołączanego pliku. *filename* musi zawierać rozszerzenie pliku i musi być ujęty w cudzysłów (").

Zwróćcie uwagę na fakt, że znacznik dołączania pliku jest wykonywany przed uruchomieniem dynamicznego kodu (działa podobnie do preprocesora C).

Znacznik `#include` musi być ujęty w znaki komentarza HTML lub XML; w przeciwnym wypadku zostanie zinterpretowany jako tekst.

Poniższy kod przedstawia przykład zastosowania dyrektywy dołączania po stronie serwera:

```
01 <html>
02   <body>
03      <!-- #Include file="header.inc" -->
04      Oto główna zawartość pliku:
05      <% For I=0 to 10 %>
06         <!-- #Include virtual="/Includes/Foobar.inc" -->
07      <% Next %>
08      <!-- #Include virtual="footer.inc" -->
09   </body>
10 </html>
```

Najczęściej stosowane obiekty i klasy ASP.NET

Do tej pory poznaliście pliki używane w aplikacjach ASP.NET i zaznajomiliście się z podstawami Visual Basica. Nadszedł więc czas, aby poznać niektóre podstawowe obiekty stosowane w aplikacjach ASP.NET.

Statyczny obiekt Application (Klasa HttpApplication)

Klasa `HttpApplication` definiuje metody, właściwości i zdarzenia wspólne dla wszystkich obiektów `HttpApplication` w platformie ASP.NET.

Klasa `HttpApplication` zawiera wiele innych klas jako właściwości, które rozpoznają użytkownicy klasycznego ASP, oraz inne, które nie są im znane. Ponieważ obiekt `Application` stanowi integralną część projektu aplikacji czy witryny WWW, jego opisowi poświęcono wiele miejsca.

Właściwości obiektu Application

Obiekt `Application` ma kilka właściwości wykorzystywanych przez system:

- `Application` — zwraca referencję do instancji `HttpApplicationState`, obiektu używanego w pliku *global.asax* oraz w każdym Web Form w ASP.NET. Jest to zbiór zawierający zmienne, obiekty i komponenty o zasięgu aplikacji.

  Klasa `HttpApplicationState` umożliwia programistom współdzielenie globalnych informacji pomiędzy wieloma żądaniami, sesjami i potokami wewnątrz aplikacji ASP.NET. (*Aplikacja ASP.NET* to suma wszystkich plików, stron, procedur obsługi zdarzeń, modułów i kodu w wirtualnym katalogu i jego podkatalogach na jednym serwerze WWW).

- `Context` — umożliwia dostęp do obiektu `HTTPContext` bieżącej instancji aplikacji.
- `Request` — umożliwia dostęp do obiektu `HttpRequest Intrisic`, który udostępnia dane nadchodzących żądań HTTP.
- `Response` — umożliwia dostęp do obiektu `HttpResponse Intrisic`, który umożliwia transmisję danych odpowiedzi HTTP do klienta.
- `Server` — umożliwia dostęp do obiektu `Server Intrinsic`.
- `Session` — umożliwia dostęp do obiektu `Session Intrisic`.

Zdarzenia aplikacji

Przez system używane są następujące zdarzenia aplikacji:

- `Application.OnStart()` — zdarzenie generowane, gdy aplikacja jest uruchamiana na serwerze WWW.
- `Application.OnEnd()` — zdarzenie generowane, gdy aplikacja kończy działanie lub przetwarzanie na serwerze WWW.

- `Application.OnError()` — zdarzenie generowane, gdy aplikacja napotyka błąd; może być zastosowane do stworzenia lub przypisania nowej procedury obsługi błędów dla aplikacji WWW.
- `Application.BeginRequest()` — zdarzenie generowane, gdy aplikacja odbiera nowe żądanie.
- `Application.EndRequest()` — zdarzenie generowane, gdy aplikacja kończy obsługę nowego żądania.
- `AuthenticateRequest()` — zdarzenie generowane, gdy aplikacja otrzymuje nowe żądanie i jest gotowa do uwierzytelnienia.

Statyczny obiekt Request (Klasa HttpRequest)

Fakt dotarcia klienta do strony sieci WWW na serwerze WWW nazywa się *żądaniem*, ponieważ — technicznie rzecz ujmując — użytkownik żąda od serwera przesłania strony do jego przeglądarki. Obiekt `Request` jest stosowany do pobierania informacji z serwera WWW i przeglądarki klienta.

Właściwości obiektu Request

Obiekt `Request` posiada następujące właściwości:

- `AcceptTypes` — zwraca tablicę łańcuchów znaków obsługiwanych przez klienta typów MIME. Jest to właściwość tylko do odczytu.
- `ApplicationPath` — zwraca ścieżkę do katalogu głównego wirtualnej aplikacji.
- `Browser` — dostarcza informacji na temat możliwości przeglądarki przychodzącego klienta.
- `ClientCertificate` — zwraca informacje o certyfikacie bezpieczeństwa klienta, który zgłosił bieżące żądanie.
- `ContentEncoding` — opisuje zestaw znaków dostarczanych przez klienta. Jest to właściwość tylko do odczytu.
- `ContentType` — opisuje typ zawartości MIME odebranego żądania. Jest to właściwość tylko do odczytu.
- `Cookies` — zwraca zbiór zmiennych cookie klienta.
- `FilePath` — wskazuje wirtualną ścieżkę bieżącego żądania. Jest to właściwość tylko do odczytu.
- `Files` — zwraca zbiór plików przekazanych przez klienta (w formacie MIME multipart).
- `Form` — zwraca zbiór zmiennych formularza.
- `HttpMethod` — opisuje metodę transferu danych użytą przez klienta (`GET`, `POST`).
- `InputStream` — umożliwia dostęp do surowych składników nadchodzącej encji HTTP.
- `IsAuthenticated` — wskazuje, czy nastąpiło uwierzytelnienie połączenia HTTP.

- `IsSecureConnection` — wskazuje, czy połączenie HTTP jest bezpieczne (HTTPS).
- `Path` — wskazuje wirtualną ścieżkę bieżącego żądania. Jest to właściwość tylko do odczytu.
- `QueryString` — zwraca zbiór zmiennych `QueryString`.
- `RequestType` — opisuje metodę transferu danych użytą przez klienta (`GET`, `POST`).
- `ServerVariables` — zwraca zbiór zmiennych serwera WWW. (Stosowany ze względu na kompatybilność; generalnie lepiej jest używać zdefiniowanych właściwości obiektu `Request`).
- `TotalBytes` — zwraca liczbę bajtów w bieżącym strumieniu wejściowym.
- `Url` — zwraca informację dotyczącą URL bieżącego żądania.
- `UrlReferrer` — zwraca informację dotyczącą URL poprzedniego żądania klienta, które doprowadziło do bieżącego URL.
- `UserAgent` — zwraca surowy User Agent String przeglądarki klienta.
- `UserHostAddress` — zwraca adres IP zdalnego klienta.
- `UserHostName` — zwraca nazwę DNS zdalnego klienta.
- `UserLanguages` — zwraca posortowaną tabelę preferencji językowych klienta.

Metody obiektu Request

Obiekt `Request` ma następujące metody:

- `BinaryRead(int32 numBytes)` — odczytuje binarnie podaną liczbę bajtów z bieżącego strumienia wejściowego.
- `MapPath(String VirtualPath)` — przekłada wirtualną ścieżkę (w żądaniu URL) na fizyczną ścieżkę na serwerze dla bieżącego żądania.
- `SaveAs(String filename, Boolean incHeaders)` — zapisuje żądanie HTTP na dysk.

Statyczny obiekt Response (Klasa HttpResponse)

Obiekt `Response` jest stosowany do wysyłania informacji z serwera sieci WWW do przeglądarki klienta.

Właściwości obiektu Response

Obiekt `Response` ma następujące właściwości:

- `Buffer` — umożliwia dostęp do bufora `HttpResponse`, co pozwala na złożenie całej strony na serwerze i wysłanie jej w całości do klienta, zamiast przesyłania przetworzonych sekcji z osobna.
- `BufferOutput` — ustawia wartość wskazującą, czy wyjście HTTP jest buforowane.
- `Cache` — zwraca informacje o buforowaniu strony WWW (czas wygaśnięcia, prywatność, klauzule warunkowe).

- `Charset` — ustala zestaw znaków dla wyjścia HTTP.
- `ContentEncoder` — umożliwia dostęp do bieżącego kodera zawartości.
- `ContentEncoding` — ustala zestaw znaków dla wyjścia HTTP.
- `ContentType` — ustawia typ MIME wyjścia HTTP.
- `Cookies` — pobiera zbiór `HttpCookie` wysłany przez bieżącą odpowiedź.
- `Expires` — ustala ustawienia wygasania dla klienta.
- `IsClientConnected` — pobiera wartość wskazującą, czy klient jest jeszcze połączony z serwerem.
- `StatusCode` — ustala kod stanu HTTP dla wyjścia zwracanego do klienta.
- `StatusDescription` — ustala komunikat stanu HTTP zwracany do klienta.

Metody obiektu Response

Obiekt `HttpResponse` ma następujące metody:

- `AppendToLog(String LogEntry)` — dodaje własną informację do pliku dziennika IIS.
- `Pics(String PicsValue)` — dodaje etykietę PICS (*Platform for Internet Content Selection*) nagłówka HTTP do strumienia wyjściowego.
- `Redirect(String newURL)` — przekierowuje przeglądarkę klienta na nowy adres URL.
- `Write(String outputString)` — zapisuje wartość do strumienia wyjściowego zawartości HTTP.
- `WriteFile(Filename)` — zapisuje plik bezpośrednio do strumienia wyjściowego zawartości HTTP.

Klasa Server (Klasa HttpServerUtility)

Obiekt `ServerUtility` jest niemal zupełnie tym samym co obiekt `Server` klasycznego ASP. Umożliwia dostęp do użytecznych narzędzi po stronie serwera do stosowania w aplikacjach ASP.NET.

Właściwości obiektu Server

Krótki spis właściwości obiektu `Server` zawiera:

- `MachineName` — pobiera nazwę serwera sieci WWW.
- `ScriptTimeout` — wymaga podania w sekundach limitu czasu dla skryptów serwera.

Metody klasy HttpServerUtility

Poniżej przedstawiono krótki przegląd metod klasy `HttpServerUtility`:

- `CreateObiect(String progid)` — tworzy instancję obiektu COM identyfikowanego przez progid.

- `Execute` — uruchamia inną stronę WWW na tym serwerze, wstrzymując wykonywanie bieżącej strony do momentu, aż serwer zakończy przetwarzanie nowej.
- `GetLastError` — zwraca ostatni odnotowany wyjątek bieżącej aplikacji WWW lub strony.
- `HtmlEncode` — koduje łańcuch i zwraca zakodowany łańcuch.
- `HtmlDecode` — dekoduje łańcuch i zwraca zdekodowany łańcuch.
- `Mappath` — przekłada ścieżkę wirtualną na ścieżkę fizyczną.
- `Transfer` — kończy wykonywanie bieżącej strony i rozpoczyna wykonywanie nowego żądania, używając podanej ścieżki URL.
- `UrlEncode` — koduje łańcuch.
- `UrlDecode` — dekoduje łańcuch.
- `UrlPathEncode` — koduje część łańcucha URL zawierającego ścieżkę i zwraca zakodowany łańcuch.

Śledzenie aplikacji ASP.NET

Podczas projektowania aplikacji ASP.NET macie możliwość *śledzenia* stanu lub postępu działania aplikacji w czasie rzeczywistym. Jest to możliwe dzięki funkcjom śledzenia ASP.NET.

Zwykłą metodą osiągnięcia tego efektu w aplikacjach klasycznego ASP było umieszczanie niemal w każdym miejscu kodu instrukcji `response.write()`. Technika ta jednak wpływa na:

- **wydajność** — większa ilość kodu na stronie WWW powoduje, że wczytuje się on wolniej;
- **czytelność i łatwość obsługi kodu** — częste stosowanie w kodzie strony internetowej funkcji `response.write()`, zarówno do wypisywania treści strony, jak i informacji służącej do kontroli działania, może początkowo uczynić kod trudnym do czytania i obsługi;
- **możliwość wyłączania** — prawdopodobnie najważniejszą możliwością, której brakuje metodzie `response.write`, jest włączanie i wyłączanie wyjścia do przeglądarki.

ASP.NET pozwala na dynamiczne śledzenie stanu aplikacji WWW dzięki swojemu systemowi śledzenia. System, w który wyposażony jest ASP.NET, wykonuje dwa typy operacji:

- **śledzenie na poziomie strony** (*Page-level Tracing*) — pozwala na śledzenie na poziomie strony WWW na podstawie systemu „strona po stronie”;
- **śledzenie na poziomie aplikacji** (*Application-level Tracing*) — pozwala na śledzenie na poziomie aplikacji.

Oba typy śledzenia są konfigurowane za pomocą pliku konfiguracyjnego *web.config*.

Informacje śledzenia

Na razie omówiliśmy śledzenie na poziomie abstrakcyjnym. ASP.NET automatycznie śledzi stan aplikacji i stron WWW na serwerze sieci WWW i gdy śledzenie jest włączone, większość z tych informacji jest dostępnych dla programisty. Dostępne informacje pochodzące z operacji śledzenia są omówione w następnych podrozdziałach.

Przed szczegółowym przedstawieniem każdej sekcji wyników śledzenia warto stworzyć prostą stronę internetową z włączonym śledzeniem. Aby to zrobić, wystarczy umieścić na początku strony *.aspx* dyrektywę:

```
<%@page Trace="true" %>
```

Podczas przetwarzania generowana jest lista śledzenia, umieszczona później na końcu strony. Lista ta składa się z wymienionych poniżej sekcji.

Sekcja *Request Details* (szczegóły żądania) zawiera następujące elementy:

`Session ID`	Identyfikator sesji dla podanego żądania.
`Time of Request`	Czas pojawienia się żądania.
`Request Encoding`	Kodowanie znaków zastosowane w żądaniu.
`Request Type`	`GET | POST`
`Status Code`	Wartość kodu stanu powiązanego z odpowiedzią. Więcej informacji na ten temat można znaleźć pod adresem *http://www.w3c.org* w dokumencie RFC 2616 dla HTTP 1.1
`Response Encoding`	Kodowanie znaków dla odpowiedzi

Sekcja *Trace Information* (informacje śledzenia) zawiera następujące elementy:

`Category`	Kategoria śledzenia podana za pomocą metody `Trace.Warn` lub `Trace.Write`.
`Message`	Wiadomość śledzenia podana za pomocą metody `Trace.Warn` lub `Trace.Write`.
`From First(s)`	Czas w sekundach od wyświetlenia pierwszej informacji.
`From Last(s)`	Czas w sekundach od wyświetlenia ostatniej informacji.

Sekcja *Control Tree* (drzewo kontrolek) zawiera następujące elementy:

`Control ID`	Identyfikator kontrolki. Jeśli nie podano ID kontrolki, ASP.NET wygeneruje je automatycznie za pomocą właściwości `ClientID`.
`Type`	Pełna nazwa typu kontrolki.
`Render Size Bytes`	Rozmiar w bajtach odrysowywanej kontrolki (włącznie z kontrolkami potomnymi).
`ViewState Size Bytes`	Rozmiar w bajtach informacji o wartościach atrybutów kontrolki (wyłączając kontrolki potomne).

Sekcja *Session State* (stan sesji) zawiera następujące elementy:

`Session Key`	Klucz stanu sesji.
`Type`	Typ danych składowanego obiektu.
`Value`	Aktualnie składowany obiekt.

Sekcja *Application State* (stan aplikacji) zawiera następujące elementy:

`Application Key`	Klucz stanu aplikacji.
`Type`	Typ danych składowanego obiektu.
`Value`	Aktualnie składowany obiekt.

Sekcja *Cookies Collection* (zbiór cookies) zawiera następujące elementy:

`Name`	Nazwa cookie.
`Value`	Dane przechowywane w cookie.
`Size`	Rozmiar cookie w bajtach.

Sekcja *Headers Collection* (zbiór nagłówków) zawiera następujące elementy:

`Name`	Nazwa elementu nagłówka.
`Value`	Dane elementu nagłówka.

Sekcja *Form Collection* (zbiór formularza) zawiera następujące elementy:

`Name`	Nazwa zmiennej formularza.
`Value`	Dane zmiennej formularza.

Sekcja *Server Variables* (zmienne serwera) zawiera następujące elementy:

`Name`	Nazwa zmiennej serwera.
`Value`	Dane zmiennej serwera.

Statyczny obiekt Trace (Klasa TraceContext)

Obiekt `Trace` jest nową wbudowaną kontrolką ASP.NET. Służy on do zmieniania ustawień śledzenia strony WWW lub do wysyłania informacji śledzenia z aplikacji WWW do systemu śledzenia.

Właściwości klasy TraceContext

Poniżej przedstawiono właściwości obiektu `Trace`:

- `IsEnabled` — wskazuje, czy dla bieżącego żądania WWW śledzenie jest włączone.
- `TraceMode` — ustawia porządek, w jakim wiadomości śledzenia będą przekazywane do żądającej przeglądarki.

Metody obiektu Trace

Obiekt `Trace` ma następujące metody:

- `Write` — zapisuje informacje śledzenia do dziennika śledzenia.
- `Warn` — zapisuje informacje śledzenia do dziennika śledzenia. W przeciwieństwie do `Write`, wszystkie ostrzeżenia będą oznaczone w dzienniku kolorem czerwonym.

Stosowanie obiektu Trace

Obiekt `Trace` jest stosowany w Web Forms tylko do czterech zadań. Są to:

- **zmiana porządku wyświetlania informacji śledzenia** — temu celowi służy właściwość `trace.TraceMode`. Poniżej przedstawiono przykład, który ilustruje uporządkowanie informacji śledzenia według kategorii:

```
Trace.TraceMode="sortByCategory"
```

- **sprawdzenie, czy śledzenie bieżącej strony jest włączone** — w tym celu stosuje się właściwość `trace.isEnabled` obiektu `Trace`. Poniżej przedstawiono przykład takiego zastosowania, gdzie flaga `trace` działa jako przełącznik dla własnego kodu służącego do usuwania błędów:

```
01 If trace.enabled then
02   'kod służący do usuwania błędów
03 End if
```

- **wysyłanie informacji** (*Trace Information*) **do systemu śledzenia** — za pomocą metody `trace.write()` można wysłać tekst do systemu śledzenia. Metoda ta pozwala na podanie samego łańcucha oraz kategorii, w której zostanie on umieszczony. Poniższy przykład ilustruje dodanie informacji do kategorii `"USER DEFINIED"`:

```
Trace.write("USER DEFINIED", "this is a user definied trace string")
```

- **wysyłanie ostrzeżeń** (*Warning Trace Information*) **do systemu śledzenia** — działa tak samo jak metoda `Write`, z jednym wyjątkiem — wysłany łańcuch będzie wyświetlony w wypisanej informacji śledzenia na czerwono. Oto przykład ilustrujący to zastosowanie:

```
Trace.warn("USER DEFINIED", "this is a user definied trace string")
```

Stosowanie śledzenia na poziomie strony

Aby stosować śledzenie na poziomie strony, należy umieścić atrybut `Trace` w dyrektywie `@ Page`. Oto przykład:

```
<%@ Page Trace="true" %>
```

Po umieszczeniu tego kodu strona WWW wypisuje całą informację śledzenia na końcu dyrektywy podczas jej działania.

Gdy atrybut `Trace` jest ustawiony na `true`, właściwość `Trace.isEnabled` jest również ustawiona na `true`.

Dyrektywa `@ Page` obsługuje jeszcze jeden atrybut do stosowania w systemie śledzenia — `TraceMode`. Atrybut ten jest stosowany do ustawiania kolejności wyświetlenia informacji systemu śledzenia. Ma on tylko dwa możliwe atrybuty:

- `SortByTime`
- `SortByCategory`

Poniższy przykład ilustruje użycie atrybutu `TraceMode` w dyrektywie `@ Page`:

```
<%@ Page Language="VB" Trace="True" TraceMode="SortByCategory" %>
```

Gdy dla strony włączono śledzenie, informacje śledzenia są wyświetlane w każdej przeglądarce, która żąda od serwera tej strony.

Przykład śledzenia na poziomie strony przedstawiono na wydruku 2.2.

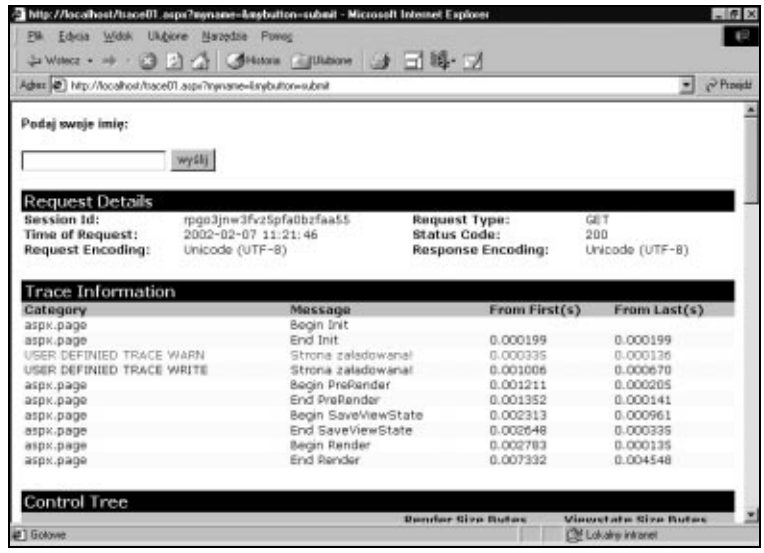
Wydruk 2.2. *Przykład śledzenia strony (trace01.aspx)*

```
01 <%@ Page Trace="True" TraceMode="SortByTime"%>
02
03 <Script runat="Server">
04
05 Sub Page_Load( s As Object, e As EventArgs )
06 trace.warn("USER DEFINIED TRACE WARN", "Strona załadowana!" )
07 trace.write("USER DEFINIED TRACE WRITE", " Strona załadowana!" )
08 End Sub
09
10
11 </Script>
12
13 <html>
14 <body>
15 <b>Podaj swoje imię:</b>
16 <br>
17 <form runat="server">
18  <asp:textbox id="myname" runat="server" />
19  <asp:button id="mybutton" text="wyślij" runat="server" />
20  <br>
21 </form>
22 </body>
23 </html>
```

W kodzie przedstawionym na wydruku 2.2 system śledzenia zostaje uruchomiony z porządkiem wyświetlania wyników zależnym od czasu. Następnie w obsłudze zdarzenia `page_load` tworzona jest informacja i ostrzeżenie systemu śledzenia. Wynik działania tego kodu przedstawiono na rysunku 2.1.

Śledzenie na poziomie aplikacji

Aby umożliwić śledzenie na poziomie aplikacji, należy dodać do pliku *web.config* sekcję `Trace`.



Rysunek 2.1. *Przykład zastosowania śledzenia na poziomie strony*

Sekcja `Trace` służy do konfigurowania systemu śledzenia ASP.NET:

```
<configration>
   <system.web>
      <customErrors mode="Off"/>
      <trace
       enabled="true"
       requestLimit="10"
       pageOutput="true"
       traceMode="SortByTime"
       localOnly="true"
      />
   </system.web>
</configuration>
```

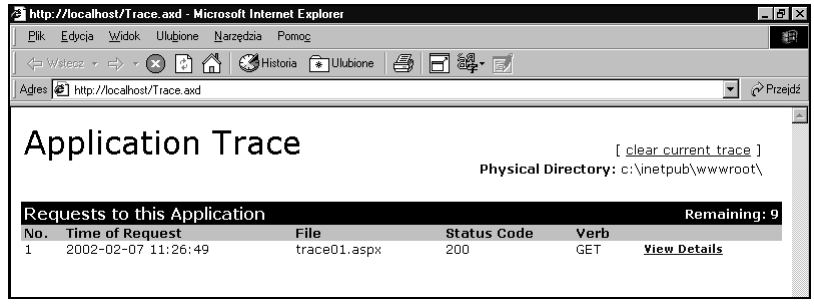
Poniżej przedstawiono składnię sekcji `Trace`:

- `enabled` — należy ustawić na `true`, jeśli możliwe jest śledzenie aplikacji; w przeciwnym wypadku `false`. Wartością domyślną jest `false`.
- `requestLimit` — liczba śledzonych żądań składowanych na serwerze. Wartością domyślną jest `10`.
- `pageOutput` — jeśli informacja śledzenia ma być wyświetlana zarówno na stronie aplikacji, jak i w narzędziu śledzenia, *.axd* należy ustawić na `true`; w przeciwnym wypadku na `false`.
  Opcja ta nie ma wpływu na strony z włączonym śledzeniem.
- `traceMode` — wskazuje, czy informacja śledzenia powinna być wyświetlona w kolejności, w jakiej była przetwarzana, według czasu (`SortByTime`) czy w kolejności alfabetycznej zgodnie z kategorią zdefiniowaną przez użytkownika (`SortByCategory`).
- `localOnly` — jeśli podgląd śledzenia (*trace.axd*) ma być dostępny tylko na komputerze, na którym działa serwer, należy ustawić na `true`; w przeciwnym wypadku na `false`.

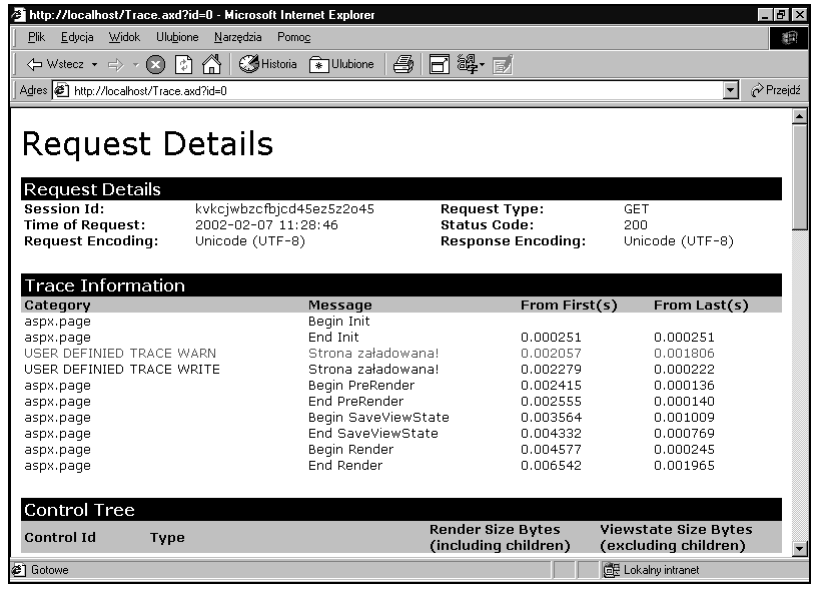
Użycie podglądu śledzenia (trace.axd)

Po włączeniu śledzenia dla całej aplikacji WWW każda strona przetwarza żądania stron tak, jakby włączone było ich śledzenie. Żądania te są przechowywane na serwerze i mogą być przeglądane za pomocą aplikacji podglądu śledzenia (*trace.axd*). Jest to plik umieszczony w katalogu głównym aplikacji, tworzony w momencie, gdy w pliku konfiguracyjnym *web.config* umieszczana jest zdefiniowana powyżej sekcja. Działanie podglądu przedstawiają rysunki 2.2 i 2.3.

Rysunek 2.2. *Strona śledzenia aplikacji*



Rysunek 2.3. *Strona szczegółów żądania*



Przechodzenie do ASP.NET

Jednym z problemów, z którymi spotka się większość użytkowników, jest przejście z klasycznego ASP do ASP.NET. Ten podrozdział ma na celu przedstawienie obszarów możliwej niekompatybilności pomiędzy tymi odmianami ASP.

ASP i ASP.NET mogą bez problemu działać obok siebie na jednym serwerze IIS; nie istnieje zatem możliwość zniszczenia aplikacji ASP przez instalację ASP.NET.

Niezależność ta jest osiągnięta przez zastosowanie różnych rozszerzeń plików dla stron ASP i ASP.NET. Oznacza to, że migracja będzie w większości polegała na skopiowaniu plików *.asp* do plików *.aspx*, przetestowaniu ich, wyeliminowaniu problemów i publikacji.

Poniżej wymieniono listę możliwych problemów z kompatybilnością pomiędzy klasycznym ASP i ASP.NET.

Zmiany w <script> i <% %>

Wszystkie procedury i zmienne globalne muszą być deklarowane wewnątrz bloków `<script runat=server>`, a nie w blokach odrysowujących ASP: `<% %>`.

Strona może zawierać więcej niż jeden blok `<script>`, ale we wszystkich blokach na stronie musi być stosowany ten sam język programowania.

Cały kod umieszczony w bloku `<script> </script>`, oprócz definicji globalnych zmiennych, musi być zawarty wewnątrz procedur.

W klasycznym ASP kod był umieszczony pomiędzy znacznikami `<%` i `%>`, a wszystkie elementy strony umieszczone przed pierwszym znacznikiem `<%` były wykonywane zaraz po załadowaniu się strony. Aby uzyskać kod wykonujący się zaraz po załadowaniu strony w ASP.NET, należy zastosować obsługę zdarzenia `page_load`.

Funkcje odrysowujące skryptu

Funkcje odrysowujące nie są obsługiwane przez ASP.NET. W klasycznym ASP można było umieszczać kod HTML wewnątrz ciała funkcji, jak przedstawiono w poniższym przykładzie:

```
01 <%
02 Function myRenderFunction()
03 %>
04 <table>
05  <tr>
06   <td>
07    <% Response.Write "Render function sample text" %>
08   </td>
09  </tr>
10 </table>
11 <%
12 End Function
13 %>
```

Na stronie ASP.NET taki kod wywołałby błąd. Aby tego uniknąć, należałoby zapisać ten skrypt następująco:

```
01 <SCRIPT LANGUAGE="VB" runat=server>
02 Function myRenderFunction()
03  Response.Write("<table><tr><td>")
04  Response.Write("Render function sample text")
05  Response.Write("</td></tr></table>")
06 End Function
07 </SCRIPT>
```

Aby wygenerować ten fragment kodu HTML, wystarczy teraz wywołać powyższą funkcję.

Obsługiwane języki stron internetowych

Cały kod na stronie ASP.NET musi być napisany w tym samym języku programowania. Obecnie ASP.NET obsługuje Visual Basic, C# i JScript. Przykładowo:

```
<%@Language="Jscript" %>
```

Nazwa języka może być zadeklarowana jako atrybut znacznika `<script>`, jeśli jest to wymagane, ale musi być to ten sam język, w którym napisano resztę kodu na stronie.

Jeśli w różnych blokach skryptu na tej same stronie podano różne języki, zostanie zgłoszony błąd, ponieważ na jednej stronie dopuszczalny jest tylko jeden język. Można jednak stosować kontrolki użytkownika napisane w dowolnym języku kompatybilnym z ASP.NET.

Nigdy więcej VBScript

VBScript nie jest już obsługiwany, ale jego składnia jest bardzo podobna do składni Visual Basic.NET. Poniżej przedstawiono najważniejsze różnice pomiędzy nimi:

- Nie istnieje już typ danych `Variant`. Został on zastąpiony przez typ `object`. Typy obiektowe muszą być rzutowane jawnie na którykolwiek podstawowy typ danych.
- Lista parametrów wywołania każdej funkcji musi być ujęta w nawiasy. Dotyczy to również funkcji bez parametrów.
- Domyślnie, argumenty są przekazywane przez wartość, a nie przez referencję, jak w poprzednich wersjach Visual Basica.
- Obiekty nie mają już domyślnych właściwości. Wszystkie właściwości muszą być podane jawnie. Przykładowo, do tekstu zapisanego w polu tekstowym należy się odwoływać następująco:

```
Dim str As String = TextBox1.Text
```

- Typ danych `Integer` ma teraz 32 bity, typ `Long` zaś ma 64 bity.
- Typy danych powinny zawsze być jawnie rzutowane na inne typy danych. Rzutowanie niejawne jest niebezpieczne. Przykładowo, jeśli konieczna jest wartość łańcuchowa, liczbę należy zrzutować jawnie:

```
Response.Write("Number Entered is: " + Ctype(count, string))
```

- Zmienne wymienione w tej samej instrukcji `Dim` będą tego samego typu. Przykładowo, w instrukcji `Dim i, j, k As Integer`, zmienne `i`, `j` i `k` będą typu `Integer`. W poprzednich wersjach Visual Basica zmienne `i` i `j` byłyby typu `Variant`, zaś `k` typu `Integer`.
- Nie są obsługiwane polecenia `Set` i `Let`. Przypisywanie obiektów odbywa się za pomocą prostej operacji przypisania:

```
myObject1 = myObject2
```

- Zmieniła się składnia właściwości klas. Nie ma już `Property Let`, `Property Get` ani `Property Set`.

```
01 Public Property myProperty As String
02
03    Get
04       myProperty = _Value
05    End Get
06
07    Set
08       _Value = myProperty
09   End Set
10
11 End Property
```

- Podczas konkatenacji łańcuchów znaków po obu stronach operatora `&` muszą wystąpić spacje. VBScript umożliwiał zapis `a&b&c`; teraz jednak, aby uniknąć wystąpienia błędu składni, należy zapisać `a & b & c`.
- Wszystkie instrukcje `If` muszą składać się z wielu wierszy. W języku VBScript możliwe było napisanie `If` w jednym wierszu.

Dyrektywy stron sieci WWW

Poprzednie wersje ASP wymagały umieszczania dyrektyw stron, jeśli występowały, w pierwszym wierszu strony wewnątrz tego samego bloku. Przykładowo:

```
<@LANGUAGE="VBScript" CODEPAGE="932" %>
```

W ASP.NET dodano kilka nowych dyrektyw. Atrybut `Language` musi być teraz umieszczony wewnątrz dyrektywy `Page`. Ilustruje to poniższy przykład:

```
01 <%@Page Language="VB" Codepage="932"%>
02 <%@OutputCache Duration="10" VaryByParam="location"%>
```

Dyrektywy mogą być umieszczone w dowolnym miejscu strony *.aspx*, jednak standardową praktyką jest umieszczanie ich na początku strony. W dyrektywach ASP.NET wielkość znaków nie ma żadnego znaczenia; nie jest również wymagane ujmowanie wartości atrybutów w cudzysłowy.

Współpraca z COM+

Większość komponentów COM współpracuje z ASP.NET. Możliwe jest również implementowanie powiązań za pomocą `Server.CreateObject`, podobnie jak w poprzednich wersjach ASP.