Apostila

ASP.NET 2.0

com C#

Adenízio Carlos Pereira
adenizio@yahoo.com.br

Sumário

Introdução

O objetivo deste treinamento é fornecer um conteúdo para que você possa desenvolver aplicações em ASP.NET 2.0 com a linguagem C#.

Iremos aprender conceitos e aplicá-los em exercícios objetivos para que a partir deste conhecimento você possa ter condições de desenvolver habilidades avançadas e posteriormente construir aplicações com todos os recursos que o ASP.NET e a linguagem C# oferecem.

Capítulo 1

Conceitos fundamentais sobre Microsoft .NET

O que é .NET: É a tecnologia da Microsoft para conectar informação, pessoas, sistemas, e dispositivos através de software. Integrada através da plataforma Microsoft, a tecnologia .NET fornece a habilidade de rapidamente construir, instalar, gerenciar e conectar soluções com segurança em serviços Web. As soluções .NET permitem negócios para integrar seus sistemas mais rapidamente e de uma maneira mais ágil ajudar a realizar a promessa da informação, em qualquer lugar, a qualquer momento e em qualquer dispositivo.

É impossível falar em .Net sem falar em SOA. (Service Oriented Architecture) ou (Arquitetura orientada a serviço). A idéia da Microsoft é exatamente unir várias linguagens de programação em uma mesma plataforma. Todo o design da aplicação é feito usando-se recursos da Framework e independente de linguagem. As regras de negócio são definidas então em uma linguagem compatível com a Framework.

.NET Framework: O .Net Framework é uma infra-estrutura para toda a plataforma .NET. Isto inclui bibliotecas de classe base, como ADO.NET e ASP.NET, assim como a CLR.

É um ambiente para construção, implantação e execução de aplicações. O ASP.NET faz parte deste ambiente e é utilizado para desenvolvimento de aplicações Web. O ASP.NET gera código HTML e Javascript que são adaptados para serem visualizados em todos os browsers.

O .NET Framework se divide em 3 partes: CLR(Common Language Runtime, as classes Framework, FCL(Framework Class Library) e ASP.NET.

Common Language Runtime (CLR): É o mecanismo responsável pela execução das aplicações .NET Framework. Para que você possa desenvolver em uma linguagem usando os recursos do .NET Framework, esta linguagem deverá suportar a CLR pois o compilador gerará um código para o suporte CLR. Este código é chamado código gerenciado.

Veja algumas funções da CLR:

- Gerenciamento automático de memória;
- Verificação de segurança de tipos;
- Gerenciamento de exceções;
- Acesso a Metadados.

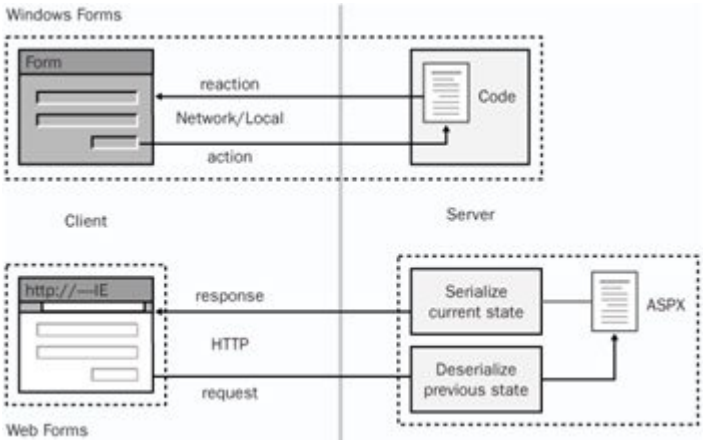
Linguagem Intermediária: O compilador não gera código nativo, mas Linguagem Intermediária (IL).

IL é independente de plataforma, assim só precisamos de um compilador para converter código IL em código nativo na máquina-alvo.

JIT Compiler: Compila o IL em código nativo na primeira vez que o código é executado. É importante citar que a compilação não acontece em todo a classe mas sim no método que é chamado. Sendo assim, se for chamado apenas um método em uma classe de 50 métodos, somente este método será compilado. Da próxima vez que o método for chamado, já estará compilado em código nativo desta forma a aplicação será executada mais rapidamente. Este é o grande diferencial entre linguagens como ASP, PHP, JSP entre outras que não são compiladas mas sim interpretadas.

Garbage Collection: Mecanismo que descarta de forma automática os objetos que não são mais utilizados por uma aplicação. Quando a CLR não encontra memória livre suficiente para atender uma solicitação de um aplicativo, a mesma inicia uma coleta de lixo(garbage collection) na esperança de reaver áreas de memória ocupadas pôr objetos que porventura não estejam mais sendo utilizados.

Programando sobre HTTP: Web Forms se comunicam com os servidores web através do protocolo HTTP que trafega sobre uma conexão TCP (Transmission Control Protocol).



Comparando os modelos Windows Forms e Web Forms na Framework .NET.

HTTP Request: Quando digitamos um endereço no browser, ele usa um servidor DNS para transformar o nome do domínio em um endereço IP. Depois o browser se conecta à porta 80 do endereço.

GET e POST: Os comandos HTTP mais usados são GET e POST. O GET retém qualquer informação identificada pelo request da URL. O POST é usado para solicitar que o servidor de origem aceite o conteúdo no request e processe-o. O POST é usado usado tipicamente para prover um bloco de dados a um processo manipulador de dados, ou seja, o resultado de um submit em um form.

View State: Páginas ASP.NET fornecem a propriedade ViewState para permitir às aplicações uma chamada de contexto e reter os valores através de 2 chamadas sucessivas para a mesma página. O ViewState representa o estado da página quando foi processada por último no servidor. O estado é persistido – usualmente, mas não necessariamente, no lado cliente – e é restaurado antes da requisição da página ser processada.

Estrutura de uma página ASP.NET: Uma página ASP.NET é um arquivo de texto no lado servidor com a extensão .aspx. A estrutura interna da página é extremamente modular e compreende três sessões distintas.

Diretivas de página: Configuram o ambiente no qual a página irá rodar, especifica como o HTTP processará a página, e determina quais os pressupostos sobre a página são seguros para fazer. Além de permitir importar namespaces, carregar assemblies, e registrar novos controles.

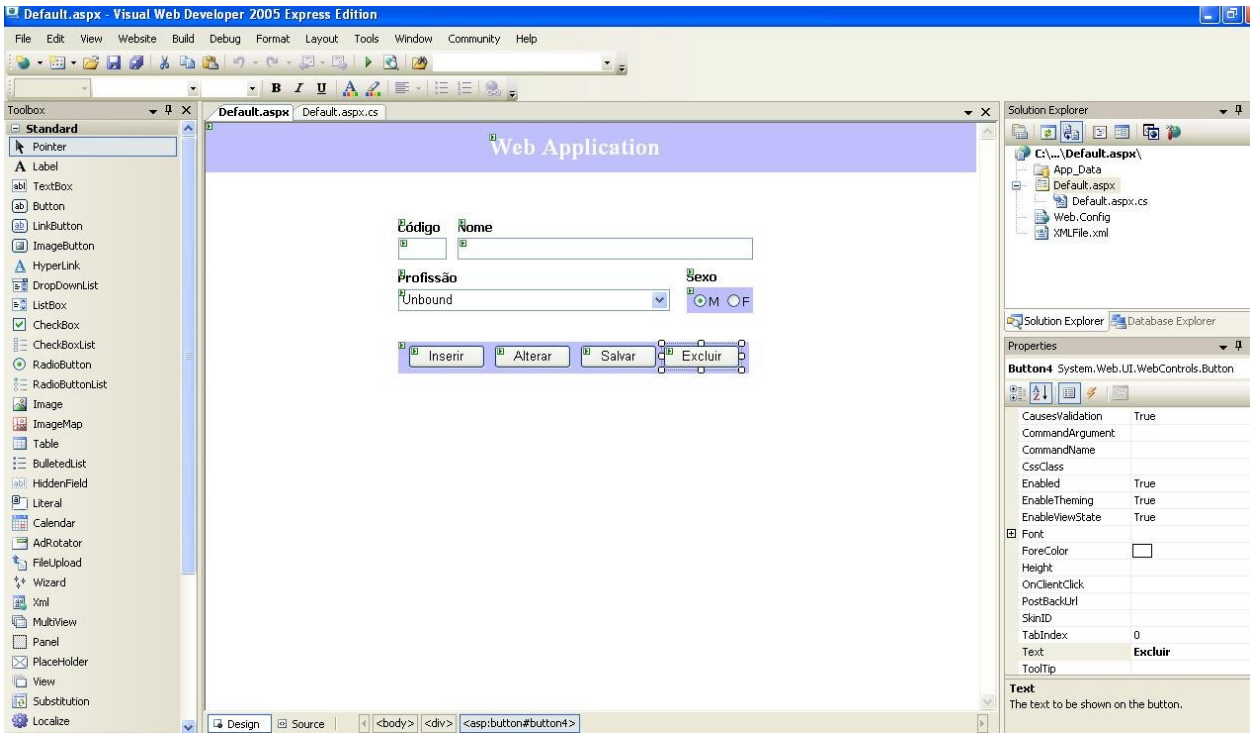
Sessão de código: Contém manipuladores para a página e eventos de controles, mais rotinas opcionais.

Layout de página: Representa o esqueleto de uma página. Isto inclui server controls, textos literais a tags HTML.

Capítulo 2

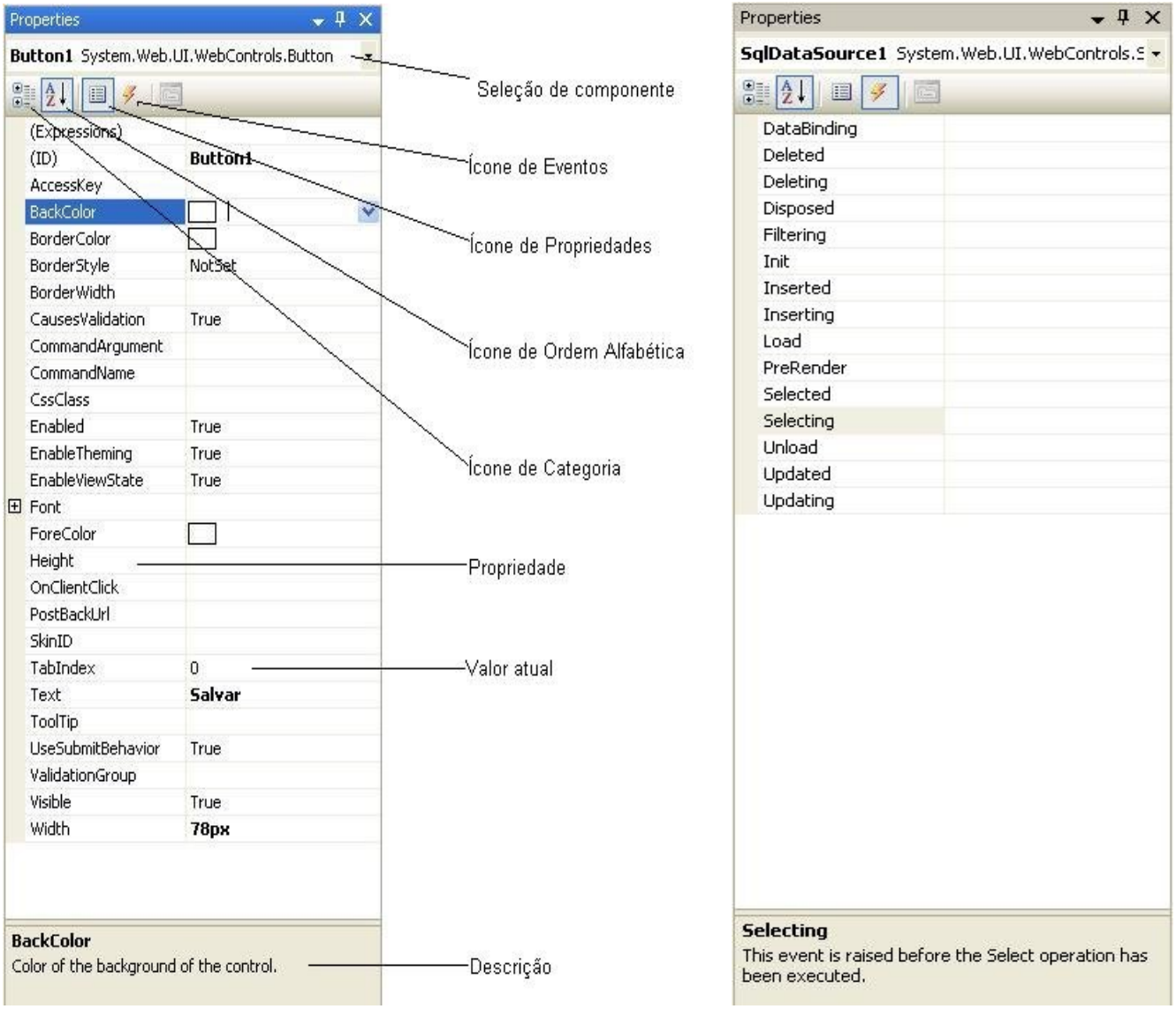
Visão Geral do Visual Studio 2005

O Visual Studio 2005 é uma IDE desenvolvida pela Microsoft para desenvolver sites e aplicações usando a tecnologia .NET com todos os recursos da plataforma .NET Framework.



Exemplo de uma página usando o VisualStudio 2005.

À esquerda temos a paleta toolbox que contém os principais componentes para desenvolvimento da aplicação. À direita na parte superior temos a aba Solution Explorer e abaixo a aba Properties.



Seleção de componente: Aqui você pode selecionar o componente ao qual deseja alterar as propriedades e eventos.

Ícone de eventos: Ao clicar neste ícone a janela altera para os eventos do componente. Conforme desenho a direita.

Ícone de propriedades: Ao clicar neste ícone a janela altera para as propriedades do componente. Conforme desenho acima.

Ícone de ordem alfabética: Ordena as propriedades por ordem alfabética.

Ícone de categorias: Organiza as propriedades por categoria.

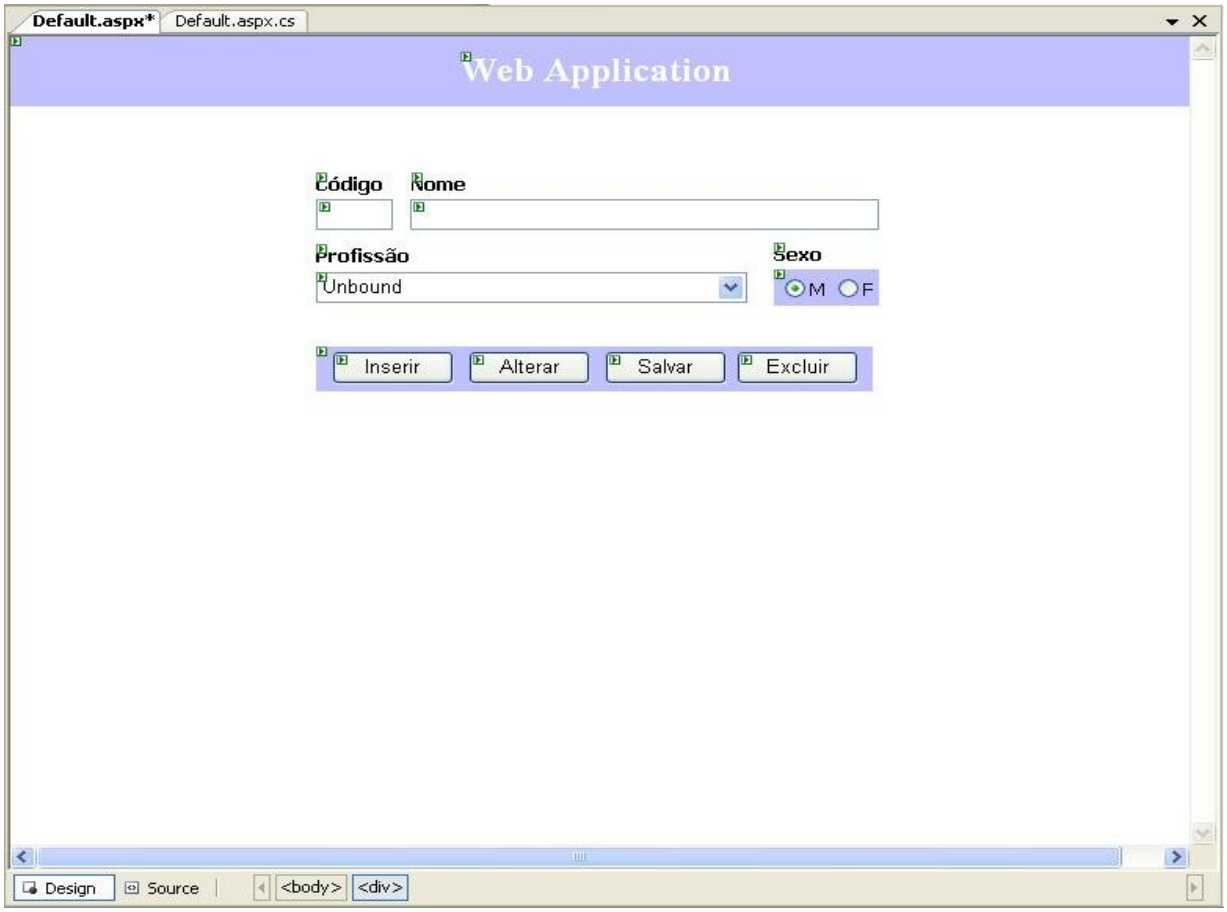
Propriedade: Define qual propriedade do componente estamos alterando.

Valor atual: Valor corrente da propriedade.

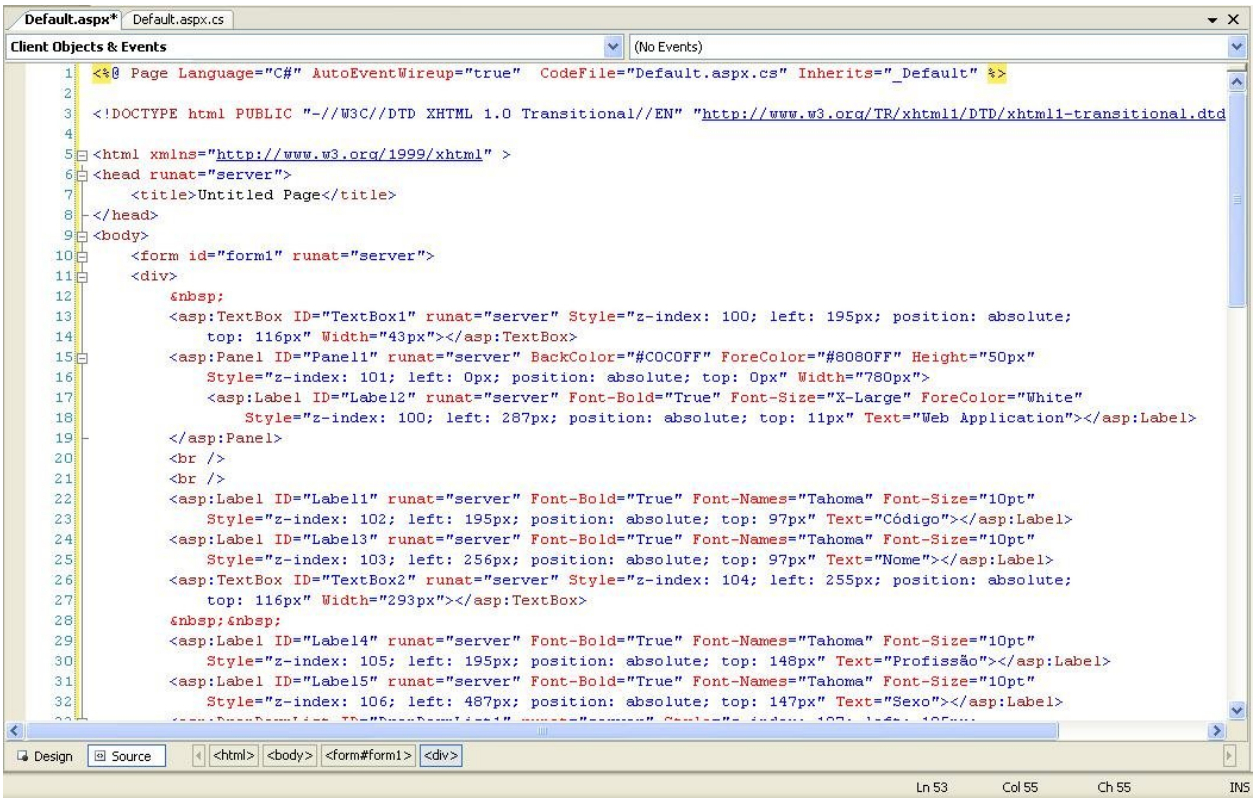
Descrição: Breve descrição da propriedade selecionada.

Na próxima figura temos a parte na qual desenvolvemos o design do site. Neste parte podemos arrastar os componentes da toolbox e se alterarmos a opção CSS positioning para Absolutely positioned podemos posicionar os componentes com o mouse. Se esta opção estiver desativada o componente será colado ao lado o último.

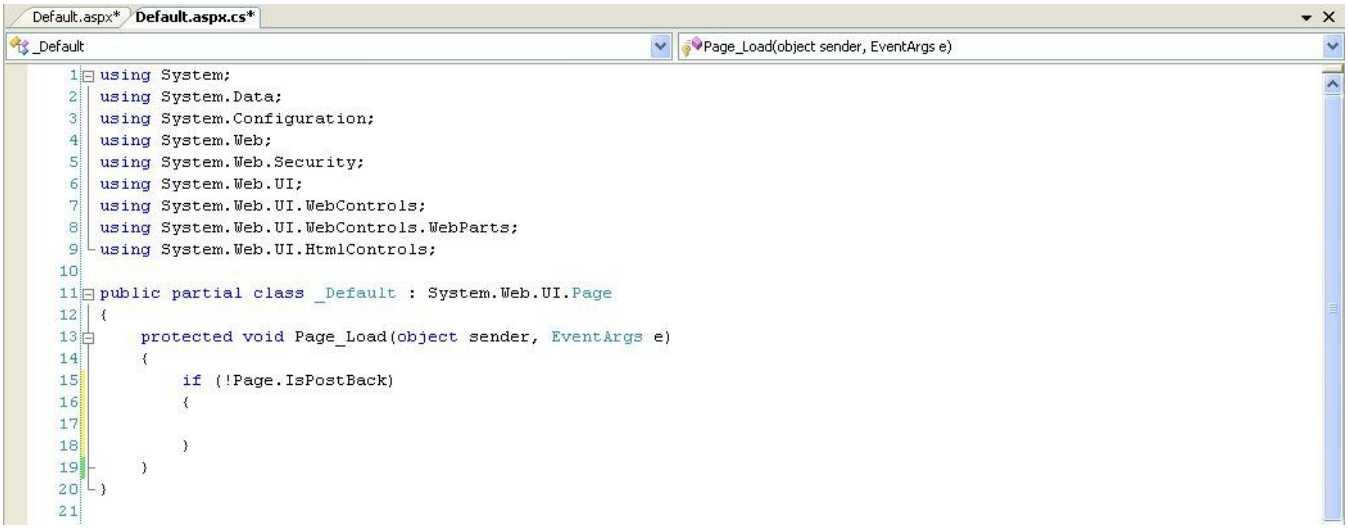
Para editar propriedades como nome, texto, cores entre outros, basta selecioná-los neste área e alterar as propriedades na paleta Properties.



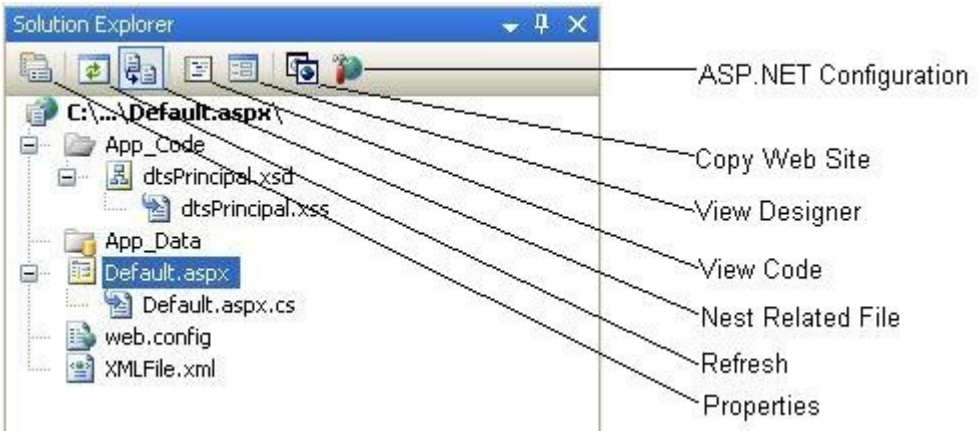
Na parte inferior temos a opção Source. Aqui nós temos o código HTML gerado pelo WebDeveloper quando arrastamos e configuramos os componentes. Se você tiver conhecimentos de HTML e quiser alterar ou criar algum componente tem toda liberdade nesta área.



Em ASP.NET diferentemente de outras linguagens, o código HTML fica em um arquivo separado do arquivo com o código da linguagem escolhida. Se observar o desenho anterior verá que existem 2 abas sendo uma Default.aspx e outra Default.aspx.cs que no caso é o arquivo que contém o código C# desta página.



Abaixo temos a paleta Solution Explorer onde podemos visualizar e gerenciar a aplicação. A partir desta paleta podemos adicionar, renomear e remover arquivos e referências.



ASP.NET Configuration: Abre o site para configuração do ASP.NET.

Copy Web Site: Abre uma janela para que você faça o deployment da aplicação sendo que do lado esquerdo aparecerão os arquivos de origem e a direita os arquivos no servidor Web para onde será feita a atualização.

View Designer: Se você estiver com o arquivo de código aberto (aspx.cs), ao clicar neste botão será aberto o arquivo com o designer (aspx).

View Code: Este botão é o inverso do anterior, ou seja, abrirá o arquivo aspx.cs quando estiver com o arquivo aspx.

Nest Related File: Aninhar arquivos relacionados. O arquivo aspx.cs ficará indentado abaixo do arquivo aspx. Como o desenho anterior, observe que existe um sina de menos ao lado do arquivo Default.aspx e o arquivo Default.aspx.cs está abaixo. Se clicarmos neste sinal o arquivo Default.aspx.cs irá desaparecer e mudará o sinal para mais.

Refresh: Atualiza a listagem dos arquivos.

Properties: Abre a paleta Properties.

Para finalizar temos a paleta ToolBox que é onde encontramos os componentes que serão inseridos em nossa página. Basicamente a ToolBox é dividida em:

Standard: Onde estão os componentes padrões e que são mais utilizados, como Label, TextBox, Button, CheckBox, etc.

Data: Aqui ficam os componentes relacionados a dados. Componentes de conexão e DataControls como GridView e DetailsView que veremos no capítulo 6.

Validation: Aqui ficam os componentes de validação do formulário que veremos no capítulo 8.

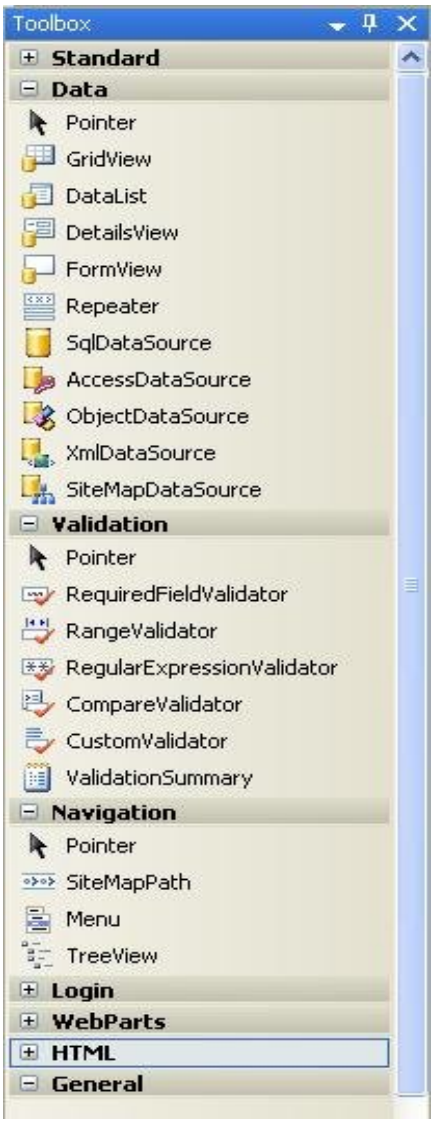
Navigation: Aqui ficam componentes de navegação como o Menu que veremos no capítulo 3.

Login: Nesta parte temos toda uma estrutura montada para criação e manutenção de login na aplicação que veremos no capítulo 7.

WebParts: Você consegue criar diferentes zonas com controles e disponibilizá-las ao internauta para customizar a exibição do layout dos sites. Com isso, cada internauta pode ter um visual diferenciado. Não será abordado em nosso curso.

HTML: Componentes html.

General: Aqui nós podemos colocar componentes e scripts criados por nós.



Capítulo 3

Interface Gráfica

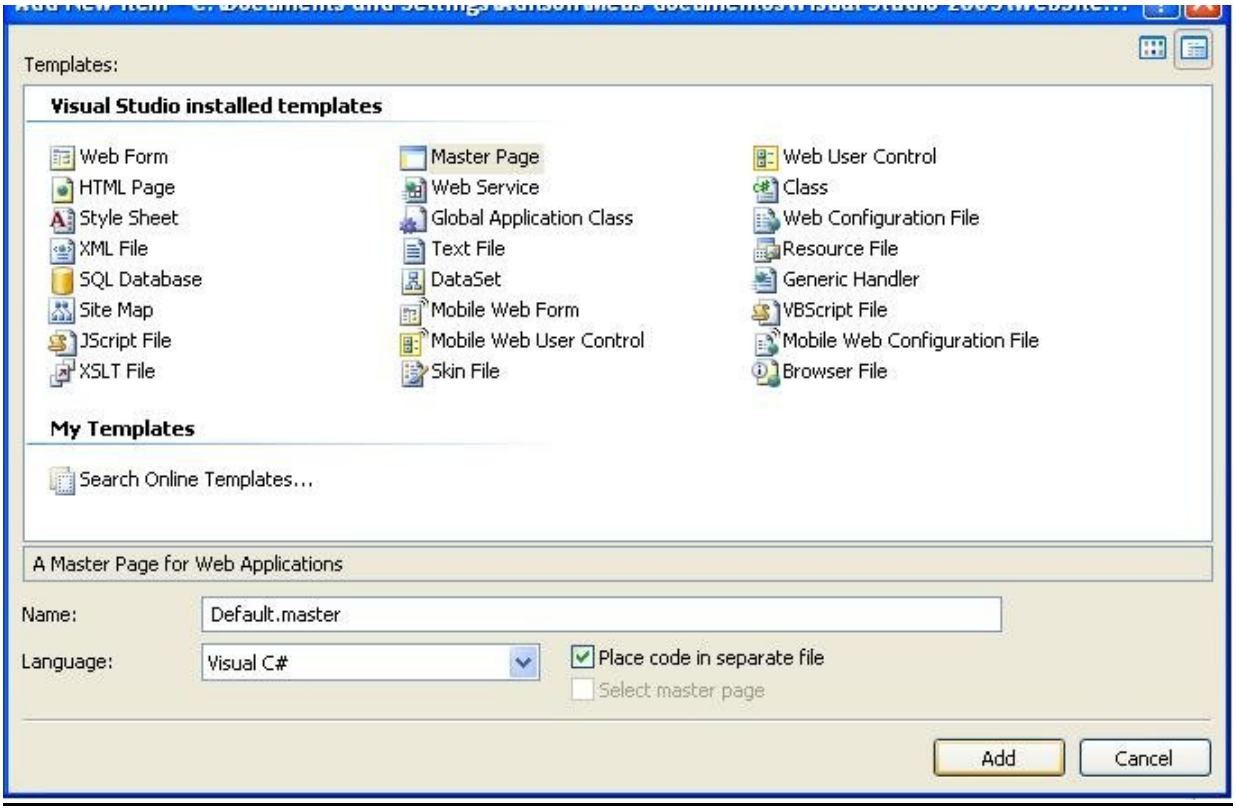
Introdução: A interface gráfica permite que um usuário interaja com um programa visualmente. São menus, botões, caixas de texto, listas entre outros. Através da interface gráfica podemos capturar as ações do usuário, processar os dados e retornar a informação processada.

Propriedades de controles: Quando nos referimos a um carro por exemplo, dizemos que ele possui algumas propriedades como marca, modelo, cor, motorização entre outras. Com os controle não é diferente. Os botões por exemplo, possuem nome, texto, cor, bordas e assim acontece com cada controle da interface gráfica.

Manipulação de eventos: Quando o usuário executa alguma ação em um controle podemos capturar esta ação e executar alguns comandos. Por exemplo quando clica em um botão podemos fazer o código para inserção de dados em um banco de dados. É como o exemplo do carro. Ele possui alguns eventos como acelerar, frear, mudar a marcha, entre outros. No decorrer do curso mostraremos alguns eventos de componentes.

Master Pages: Nas versões anteriores do ASP.NET bem como em outras linguagens como ASP, PHP e JSP, sempre que desejamos repetir um cabeçalho em todas as páginas da nossa aplicação tínhamos que usar um include para adicionar este cabeçalho em cada página, mas não tínhamos relacionamento entre o cabeçalho e a página. Na versões anteriores do ASP.NET já existia um recurso que permanece na versão atual chamado Web User Control mas na versão 2.0 foi criado um novo recurso chamado Master Page. Com o Master Page passamos a interagir entre o cabeçalho e a página e obtivemos várias melhorias.

Para criar uma Master Page vá até a Solution Explore e clique com o botão direito sobre o caminho da aplicação, no menu suspenso que aparecerá escolha Add New Item... e aparecerá uma janela com os tipos possíveis. Escolha Master Page, defina um nome para o arquivo, para a linguagem escolha C# e marque a opção Place code in separate file. Esta opção fará com que ele gere um arquivo separado para o código C#. Depois é só clicar em Add.



Será criado um arquivo com Default.master. Repare que ele possui um objeto chamado ContentPlaceHolder. Tudo o que você criar fora deste objeto será replicado em cada página que usar esta Master Page. Portanto você pode colocar a logomarca, o nome da empresa, um menu, enfim aquilo que você desejar que esteja nas páginas da sua aplicação.

Como uma página html só tem uma tag body, significa que as páginas criadas dentro do MasterPage não terão a tag body visto que a MasterPage já a possui. Se observarmos o código aspx gerado pela página filha, poderemos confirmar isto.

Mas como poderíamos então definir atributos específicos de uma página visto que os atributos da página estão definidos na MasterPage?

Para fazermos isto devemos na MasterPage definir a tag body como runat=”server” e definirmos um id. Nas páginas filhas devemos criar um objeto do tipo HtmlGenericControl com o mesmo nome da tag body.

Dentro do evento PageLoad podemos definir os atributos da seguinte forma:

```
this.corpo.Attibutes.Add("bgcolor", "#eeeeee"); //corpo é o nome do objeto.
```

Menus: Todos nós estamos familiarizados com menus. Desde um editor de texto como o Word, ou um gerenciador de arquivos como o Windows Explorer usam menus. No ASP.NET 2.0 é muito fácil criar um menu. Na toolbox na aba Navigation existe um componente Menu. Podemos arrastá-lo para nosso arquivo Default.master e configurá-lo conforme desejamos. Para começar vamos clicar em Auto Format e escolher um formato que nos agrade. Clique em Edit Menu Itens e vamos adicionar os itens do menu. Clique no primeiro botão e será criado um item. Mude a propriedade Text para Cadastro. Com este item selecionado clique no segundo botão e vamos adicionar um sub-item. Nas propriedades vamos preencher a opção NavigateUrl com o texto cad_funcionario.aspx e a propriedade Text com Funcionários. Podemos adicionar quantos itens quisermos e indentá-los conforme a necessidade da aplicação. Quando concluir clique no botão OK. Voltando ao design clique no menu e vamos na paleta Properties e vamos mudar a propriedade Orientation para horizontal. Pronto nossa aplicação já possui um menu e podemos voltar aqui quando quisermos alterá-lo.

Label: É um rótulo que colocamos em objetos que usaremos nas páginas. Para alterar o texto basta selecionar o Label e na paleta Properties muda a propriedade Text para o texto que queremos. Podemos configurar várias propriedade como tipo da fonte, tamanho, cor entre outras.

```
lblExemplo.Text = "Teste"; // Em tempo de execução.
```

TextBox: É uma caixa de texto na qual o usuário fornece dados a partir do teclado. Pode exibir também informações fornecidas pela aplicação. Assim como Label, possui várias propriedades entre elas temos TextMode que pode ser SingleLine, MultiLine e Password sendo que na última ao invés de aparecer o texto digitado, aparecem asteriscos (*) por se tratar de uma senha.

```
txtExemplo.Text = "Teste"; // Em tempo de execução.
```

Button: Um área que chama um evento quando clicada. Também possui várias propriedades que podem ser alteradas na paleta Properties bem como em tempo de execução. Existem outros tipos de botões que veremos mais adiante.

```
btnExemplo.Text = "Teste"; // Em tempo de execução.
```

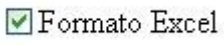
Panel: Um contêiner no qual podem ser colocados componentes. A vantagem em se usar um Panel é que todos os componentes que estiverem contidos nele serão movimentados juntamente com ele.

CheckBox: Caixa de seleção. Um controle que está ou não selecionado. Um CheckBox é utilizado normalmente quando você fornece um opção ao usuário para que possa escolher se deseja ou não marca esta opção. Os parâmetros principais são:

Checked que pode ser true quando selecionado e false quando não selecionado

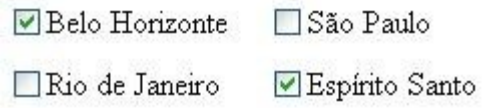
Text que é o texto que aparecerá à direita da caixa.

E o evento CheckedChanged que é disparado quando o usuário seleciona o CheckBox.




```
if (ckbExemplo.Checked) // Em tempo de execução.
```

CheckBoxList: Lista de caixas de seleção. Semelhante ao CheckBox, a diferença é que aqui temos uma lista indexada e podemos identificar o itens selecionados através do índice.

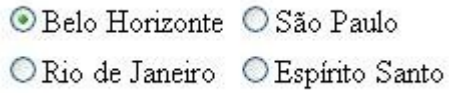



```
for (int i = 0; i < cblExemplo.Items.Count; i++)
  if (cblExemplo.Item[i].Selected)
     ...
```

ListBox: Caixa de lista. Uma área na qual uma lista de itens é exibida, E o usuário pode fazer uma seleção clicando uma vez em qualquer elemento. Vários elementos podem ser selecionados se setarmos a propriedade SelectionMode para Multiple.

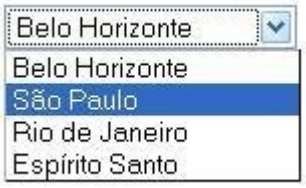
lbxExemplo.SelectedValue;
lbxExemplo.SelectedIndex;
lbxExemplo.SelectedItem;

RadioButtonList: Uma lista de botões do tipo radio. Somente uma opção poderá ser escolhida por vez. Com isto, se selecionarmos a opção São Paulo no exemplo abaixo, automaticamente a opção Belo Horizonte será desmarcada. Para identificarmos qual a opção escolhida podemos usar as propriedades SelectedIndex ou SelectedItem onde temos Value, Text entre outras.



rblExemplo.SelectedValue;
rblExemplo.SelectedIndex;
rblExemplo.SelectedItem;

DropDownListBox: Caixa de combinação. Uma lista suspensa de itens na qual o usuário pode fazer uma seleção clicando em um item da lista ou digitando na caixa, se for permitido. Para identificarmos qual a opção escolhida podemos usar as propriedades SelectedIndex ou SelectedItem onde temos Value, Text entre outras.



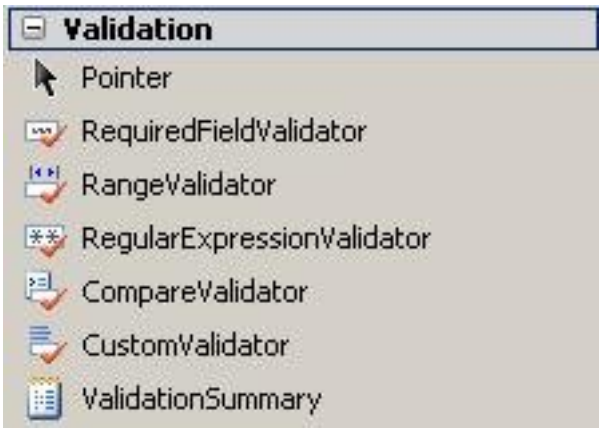
ddlExemplo.SelectedValue;
ddlExemplo.SelectedIndex;
ddlExemplo.SelectedItem;

Calendar: Um calendário onde normalmente inicializamos com o objeto oculto, ao clique de um botão o exibimos para que o usuário selecione uma data, permitindo que navegue nos meses e anos e no evento SelectionChanged verificamos a data selecionada através da propriedade SelectedDate.


```
protected void cldDataIni_SelectionChanged(object sender, EventArgs e)
{
    txtExemplo.Text = cldDataIni.SelectedDate.ToString("dd/MM/yyyy");
}
```

Capítulo 4

Controles de Validação



Para quem já está acostumado com o desenvolvimento web sabe o quanto é importante a validação de dados antes que seja feita uma requisição ao servidor. Para isto criamos funções JavaScript para atender às validações dos componentes de data, e-mail, etc.

Agora com ASP.NET isto é muito fácil de ser feito apenas arrastando os componentes e configurando-os na paleta Properties.

Neste capítulo vamos falar de cada um e veremos como ficou fácil e prático validar componentes no cliente.

RequiredFieldValidator: Para começar vamos falar do RequiredFieldValidator que talvez seja o mais usado. A tradução é “Validador de campo requerido”. Com este componente podemos verificar se um determinado campo está ou não preenchido. Isto é muito comum quando temos por exemplo a exigência do preenchimento de uma data ou e-mail ou CPF, etc.

Para usá-lo basta arrastá-lo para a tela e configurar a propriedade ControlToValidade com o nome do controle requerido. Podemos também digitar o texto que será exibida no validator pela propriedade Text. Posteriormente veremos também que o texto na propriedade ErrorMessage poderá ser exibido ao usuário.

Com a propriedade SetFocusOnError determinado que o cursor será posicionado no campo caso este não esteja preenchido.

Posteriormente veremos a função da propriedade Display.

RangeValidator: É um validador de escala, isto quer dizer que podemos colocar uma escala ou limites para os valores de um componente.

Imagine que você tenha um campo onde o usuário tenha que colocar a idade. Um valor menor que 0 ou maior que 130 é inválido. Entra em ação então o RangeValidador.

Na propriedade ControlToValidate definimos o componente que será validado.

Para definirmos os valores ou a escala permitida devemos preencher as propriedades MaximumValue e MinimumValue. No exemplo anterior temos MinimumValue=0 e MaximumValue= 130.

Podemos também digitar o texto que será exibido no validator pela propriedade Text. Posteriormente veremos também que o texto na propriedade ErrorMessage poderá ser exibido para o usuário.

Com a propriedade SetFocusOnError determinamos que o cursor será posicionado no campo caso este esteja fora da escala.

Posteriormente veremos a função da propriedade Display.

RegularExpressionValidator: É um validador de expressões pré-definidas pelo desenvolvedor. Quando queremos validar expressões que pode seguem determinados formatos ou máscaras específicos. Como exemplo temos telefone, CEP, Ur, etc.

O RegularExpressionValidador tem as mesmas propriedades do RangeValidator com excessão do MaximumValue e MinimumValue e com o acréscimo da propriedade ValidationExpression. É através desta propriedade que definimos a máscara ou expressão que será exigida no campo validado.

Se clicarmos no botão correspondente a esta propriedade na paleta Properties veremos que já estão disponíveis algumas expressões.

CompareValidator: É um validador de comparação. Quando disponibilizamos 2 componentes para o usuário selecionar um período de datas precisamos validar se a data inicial não é maior que a data final.

O CompareValidator faz este papel.

Através da propriedade ControlToCompare e da propriedade ControlToValidate determinamos os componentes envolvidos na comparação sendo ControlToValidade o componente que deverá atender às exigências configuradas.

Após definirmos os componentes devemos usar a propriedade Operator para selecionar o tipo de comparação que deverá ser feita podendo ser Equal(igual), NotEqual(diferente), GreatherThan(maior que), LessThan(Menor que), DataTypeCheck, etc.

Podemos especificar um valor a ser comparado em ValueToCompare, porém se especificarmos também um ControlToValidate o segundo será usado e o ValueToCompare ignorado.

CustomValidator: As vezes queremos criar uma função para que faça validações que não estão disponíveis em nenhum componente Validator. Para isto usamos o CustomValidator.

Através da propriedade ClientValidationFunction definimos uma client script validation function.

ValidationSummary: O último componente de validação é o ValidationSummary. Através deste componente podemos agrupar as mensagens em um mesmo local.

Quando usamos os validators e preenchemos a propriedade Text, o texto digitado é exibido no próprio validator, mas quando preenchemos a propriedade ErrorMessage o texto é exibido no ValidationSummary. Sendo assim, podemos colocar um asterisco(*) na propriedade Text do validator e um texto para o erro em si na propriedade ErrorMessage.

O ValidationSummary tem duas propriedades muito importantes que são ShowMessageBox e ShowSummary.

A primeira define se será exibida uma caixa de mensagem com os ErrorMessages e a segunda se os erros serão exibidos no próprio ValidationSummary. Podemos deixar as duas como true e os erros serão exibidos nas duas formas, mas você certamente escolherá apenas uma.

Ficamos de explicar a função da propriedade Display dos validators. Pois bem, esta propriedade define se o a exibição do validator será Static, Dynamic ou None. Se deixarmos Static o texto será exibido na página, porém se mudarmos para None só exibiremos a ErrorMessage no ValidationSummary.

Outra propriedade importante do ValidationSummary é DisplayMode. Se escolhermos List, os erros serão exibidos como uma lista. Para BulletList teremos uma lista com uma bola ao lado e se escolhermos SingleParagraph teremos um parágrafo único com os erros concatenados na mesma linha.

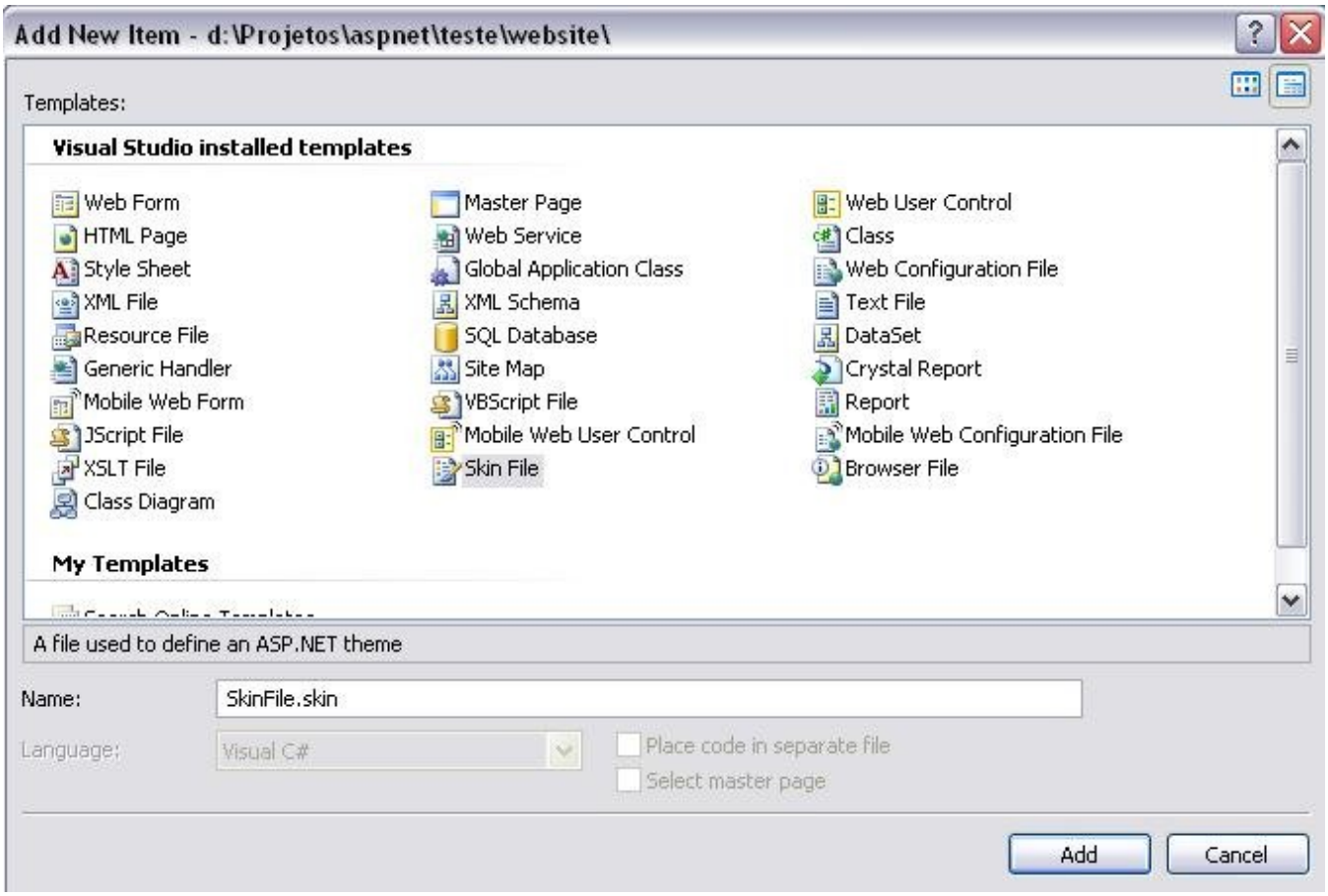
Capítulo 5

Trabalhando com temas (themes)

Conceito: Se você já está acostumado com o desenvolvimento para ambientes web, então já deve ter trabalhado com arquivos CSS. Eles são usados para definir padrões de cores, fontes, bordas, etc. de websites que mesmo podendo ser selecionados programaticamente no servidor, em seu núcleo o CSS continua a ser uma tecnologia do lado cliente, consebida e implementada para aplicar skins a elementos HTML.

Agora no Asp.NET podemos trabalhar com um novo conceito chamado theme. Um theme (tema) é composto por arquivos skin, imagens e arquivos CSS.

Imagine você poder definir propriedades como côr, fonte, fundo, borda e ao arrastar um botão ou qualquer controle em sua página ele tome a forma que você já predefiniu sem que você tenha que ficar mudando estes atributos. Este é o objetivo de trabalhar com themes.



Tela para adicionar um novo arquivo do tipo Skin File

Criando um Skin File: Para adicionar um novo skin file clique com o botão direito do mouse sobre o projeto na SolutionExplorer e depois em Add new item. Quando aparecer uma tela semelhante à tela acima, selecione Skin File e dê um nome ao arquivo. Clique em Add.

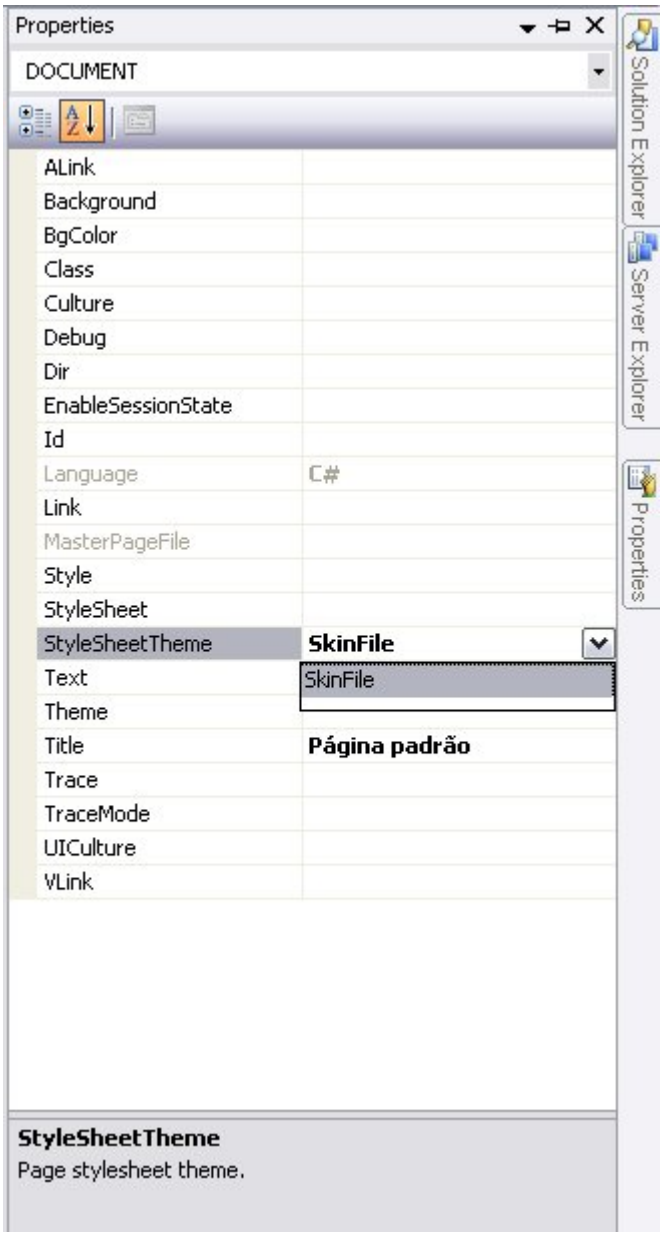
Se for a primeira vez que estiver adicionado um arquivo de skin você receberá uma mensagem de que o arquivo deve ser criado em uma pasta específica chamada App_Themes. Selecione Sim e será criada esta pasta em sua aplicação juntamente com o arquivo skin.

Crie um arquivo do tipo WebForm em sua aplicação e dê qualquer nome a ele. Depois iremos removê-lo. Arraste vários objetos para este arquivo aspx que acabamos de criar e defina as propriedades de cada objeto como você deseja que fica seu website. Por exemplo, coloque em um botão a côr de fundo, o nome e o tamanho da fonte, a côr e o tamanho da borda, etc. Faça o mesmo em todos objetos com TextBox, Calendar, RadioButtonList, CheckBoxList, DropDownList, etc.

Mude a visualização para Source e veremos o código asp. Selecione todo o texto, copie e cole no arquivo skin. Como o skin é um arquivo de configuração para todos os objetos da página não

podemos deixar a propriedade Id nos objetos. Então remova a propriedade Id de todos os objetos colados no arquivo skin.

Agora podemos excluir o arquivo aspx que criamos anteriormente. Salve o arquivo skin e abra um arquivo aspx qualquer. Na paleta properties selecione o DOCUMENT e na propriedade StyleSheetTheme selecione o arquivo de skin que você criou.



Paleta Properties

Agora basta selecionar na ToolBox um objeto que você já configurou previamente e ele automaticamente receberá a formatação definida.

Você pode ainda definir mais de uma configuração para o mesmo tipo de objeto. Imagine que você deseja ter uma configuração para Labels de título e outra para Labels de campos. Para isso basta criar duas tags <asp:Label no arquivo skin e definir a propriedade skinId para cada label. No arquivo aspx devemos então definir qual skinId iremos aplicar ao labe que estamos editando. Se você tiver uma formatação que use com mais freqüência então deixe-a sem skinId e ela será selecionada automaticamente sem que precise ficar definindo manualmente.

Capítulo 6

Introdução à linguagem C#

Visão Geral: O C# é a fase seguinte na evolução C e C++ e foi desenvolvido expressamente para a plataforma .NET da Microsoft. É impossível não citar que existe uma grande semelhança com a linguagem Java da Sun.

O C# é uma linguagem orientada a objetos, sendo assim possui todas as características como Herança, Polimorfismo, Encapsulamento, Sobrecarga de métodos entre outras.

Outra característica importante é que o C# é case sensitive, ou seja, as letras maiúsculas se diferem das minúsculas. Nesta caso a variável Nome é diferente de nome.

Variáveis: Vamos começar falando das variáveis. Ao contrário do que acontece em ASP com VBScript, as variáveis precisam ser declaradas com seus respectivos tipos, mas ao contrário do Delphi, não precisam ser declaradas em locais específicos.

Podemos declarar uma variável ao criarmos um laço for, por exemplo.

Sintaxe: tipo nome = [valor]; string sEndereco = “Rua José Geraldo”;

Onde tipo pode ser int, string, float, decimal, etc;

Nome pode ser um nome qualquer para identificar a variável;

[valor] é o valor que a variável recebe no momento de criação sendo que não é obrigatório podendo ser definido ou alterado posteriormente.

Constantes: São valores que serão armazenados em memória para ser aproveitados posteriormente. Ao contrário das variáveis, constantes devem ter um valor definido no momento de criação e não podem ser alteradas depois.

Tipos de Dados: No C# temos vários tipos de dados divididos em classes.

Decimal: Formato muito exato com precisão de 28 casas decimais.

Ex.:
```
decimal moeda = 3,10M;
```

Char: Um único caracter.

Ex.:
```
char teste = 'T';
```

Bool: Valor verdadeiro ou falso.

Ex.:
```
bool existe = true;
```

Ponto flutuante – float

Float: Precisão de 7 dígitos.

Ex.:
```
float valor = 1.34567;
```

Double: Precisão de 16 dígitos.

Ex.:
```
double valor = 1.34567890123456;
```

Byte: Valor entre 0 e 255;

Ex.:
```
byte teste = 18;
```

Long: Inteiro de 64 bits com sinal. Valores entre –9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.

Ex.:
```
long valor = 405801;
```

Inteiro: Um inteiro possui 32 bits com valores entre –2,147,483,648 a 2,147,483,647;

Ex.:
```
int valor = 10;
```

Short: Inteiro de bits com sinal. Valores entre –32,768 a 32,767.

Ex.:
```
short valor = 15;
```

String: O mais usado visto que todas as variáveis que não usam número são declaradas como string.

Ex.:
```
string teste = "texto de teste";
```

Para concatenar uma string use o sinal de + (mais).

Para extrair basta colocar a posição do caracter.
```
char s = teste[3];
```

Array: Coleção de elementos armazenados em seqüência. O primeiro elemento contém o índice 0.

Ex.:
```
int[] teste = new int[6] { 23, 10, 87, 16, 99, 5 };
string[] teste = new string[3] {"Adenízio","Carlos","Pereira"};

int[,] num = new int[2, 3] { { 2, 5, 1 }, { 19, 32, 4 } };
```

Operadores

Unários: + - ! ++x --x x++ x—
Aritméticos: * / % + -
Relacional: <> <= >= is
Igualdade: == !=
E condicional: &&
OU condicional: ||

Estrutura de controle

If: Com certeza a instrução mais utilizada e presente em praticamente todas as linguagens. É utilizada para testar um condição e executar um ou vários comandos se a condição for satisfeita. Pode ser utilizado com a instrução else para que se a condição não for satisfeita executar outros comandos.

Sintaxe:

```
if (condição) {
    //Executa comandos

} // Se a condição não for verdadeira não fará nada.

if (condição) {
    //Executa comandos

}
else {
    //Executa comandos
} // Se a condição não for verdadeira executa comandos dentre do else.
```

Switch: Em determinadas situações testar tudo com if pode ser cansativo, neste caso utilizamos o switch.

Sintaxe:

```
switch (expressão)
{
    case constante - expressão:
        //Executa comandos;
        break;
    case constante - expressão:
        //Executa comandos;
        break;
    default:
        //Executa comandos;
        break;
}
```

Ex.: Suponhamos que você tenha um comboBox onde o usuário define seu estado civil e temos os valores: 0 : Solteiro; 1 – Casado; 2 – Viúvo; 3 – Divorciado. Teríamos o código:

```
switch (varEstadoCivil)
{
    case 0:
        //Código para quem é solteiro;
        break;  // Interrompe o laço caso a condição seja verdadeira.
    case 1:
        //Código para quem é casado;
        break;
    case 2:
        //Código para quem é viúvo;
        break;
    case 3:
        //Código para quem é divorciado;
        break;
    default:
        //Código para outros.
        break;
}
```

While: Executará um ou mais comandos até que a condição não seja mais satisfeita.
Sintaxe:

```
while (condição)
{
    //Executa comandos;
}
```

Ex.:

```
int x = 0;
while (x < 10)
{
    //Executa comandos;
    x++;
}
```

Do while: Diferentemente do while que só executará os comandos se a condição for verdadeira, o do while executará pelo menos uma vez os comandos independentemente da condição ser verdadeira ou não.

Sintaxe:

```
do
{
    //Executa comandos;
}
while (condição);
```

For: Utilizada para percorrer um determinado número de registros ou itens sendo que você deve saber exatamente quantos registros deve percorrer.
Sintaxe:

```
for (inicializador; condição; repetidor)
{
    //Executa comandos;
}
```

Ex.:

```
for (int i = 0; i < 10; i++)
{
    //Executa comandos;
}
```

Foreach: Utilizado para enumerar itens de um array ou coleção.
Sintaxe:

```
foreach (TipodeDados NomedaVariável in NomeArray / NomeColeção)
{
    //Executa comandos;
}
```

```
int par = 0, impar = 0;
int[] numeros = new int[] {1,2,4,4,9,3,6}

foreach (int i in numeros)
{
    if (i%2 == 0)
        par++;
    else
        impar++;
}
```

Capítulo 7

Programação Orientada a Objetos em C#

Classes: A classe é um modelo para um objeto; ela descreve a estrutura básica do objeto. Uma analogia para entendermos classe e objeto é considerar esses conceitos em termos de casas e da construção delas. A classe seria o projeto para a casa, e a casa propriamente dita seria um objeto. Muitas casas poderiam ser criadas com base no mesmo projeto, e muitos objetos podem ser gerados com base na mesma classe. A maneira de instanciar uma classe é usando-se a palavra-chave new.

Classes são compostas de métodos e atributos que podem ser privados, públicos ou protegidos.

```
using System;
using System.Data;

namespace prjTeste {

  public class Veiculo {

    // aqui são declarados os atributos
    private double dblVelAceleracao;

    // construtor
    public Veiculo()
    {
    }

    // métodos
    public double getAceleracao() {
        return dblVelAceleracao;
    }
  }
}
```

Métodos: Todas as funções são declaradas no contexto de uma classe, portanto são métodos. Todo método tem um tipo de retorno, e caso não tenha, deve ser usada a palavra void. É possível passar parâmetros para os métodos, colocando-os entre parênteses separados por vírgulas.

O cabeçalho do método main é definido da seguinte forma:

```
public static void main(String[] args)
```

public: O método é público e pode ser acessado por qualquer outro método ou classe.

static: Significa que não é necessário instanciar a classe para se usar o método pois este método não está aplicado a objeto algum.

void: O tipo de retorno que pode ser int, string, double entre outros. Void significa que não há retorno de dados. Quando retornamos valores devemos usar return valor.

main: O nome do método. Quando criamos métodos devemos colocar nomes explicativos. Por exemplo se criamos uma função para calcular salário devemos colocar algo como CalculaSalario.

String[] args: Aqui são colocados os parâmetros que serão utilizados dentro do método. Obviamente podemos utilizar outros tipos além de string como int, double, float entre outros.

Quando utilizamos a palavra ref antes do parâmetro significa que o estamos passando por referência, ou seja, se o valor deste parâmetro for alterado dentro do método, também será alterado em toda a aplicação. Quando não usamos ref e colocamos apenas o tipo e o nome do parâmetro, o valor será alterado somente dentro do método.

Modificadores de Acesso: Definem a visibilidade dos membros para o mundo externo à classe e às classes derivadas. Veja o quadro abaixo com os principais modificadores de acesso.

Modificador de Acesso	Mundo Externo à Classe	Classe Derivada
Public	Sim	Sim
Protected	Não	Sim
Private	Não	Não

Herança: Quando criamos a classe Veiculo definimos somente o atributo velAceleracao que é comum à todo veículo. Porém se começarmos a especificar um pouco mais um veículo, podemos perceber que ele pode ou não possuir um motor. Poderíamos então criar a classe VeiculoAutomotor. Graça à herança podemos reaproveitar os atributos e os métodos da classe Veiculo, sendo necessário apenas definir na criação de nossa classe que ele herda da classe Veiculo.

Entendemos então que herança é a possibilidade de usarmos recursos criados em outras classes e a partir desta classe que chamando de base criarmos novos atributos e métodos.

Uma grande vantagem da herança em .Net é que podemos herdar classes escritas em outras linguagens, ou seja, podemos herdar em uma classe C# outra classe em VB.Net ou J#, etc.

```
using System;
using System.Data; ...

namespace prjTeste {
    public class VeiculoAutomotor : Veiculo {
        // novos atributos

        // novos métodos
    }
}
```

Classe abstrata: Às vezes podemos definir o cabeçalho de um método mas não implementá-lo na classe que está sendo criada. Então quando alguma classe herdar esta classe pai, terá que fazer a implementação deste método. Este tipo de classe é chamado classe abstrata. Por ser uma classe abstrata não pode ser implementada, ou seja, não é correto criar um objeto de uma classe abstrata já que existem métodos que não foram instanciados. Por exemplo, temos uma classe funcionário onde definimos alguns métodos que são comuns a todos funcionários como cadastra dados pessoais, define função, etc. Mas existem métodos que pode mudar de acordo com o funcionário. Ele pode ser por exemplo horista ou mensalista e neste caso o cálculo do salário é diferenciado.

```
public abstract class Funcionario{
}
```

Interface: Uma interface poderia ser definida como uma classe abstrata onde todos os métodos são abstratos se não fosse o fato de que uma classe filha não herda uma interface mas na verdade a implementa. Sendo assim uma classe filha pode implementar várias interfaces ao passo que só pode herdar uma classe. Uma interface funciona como um contrato onde a classe implementadora terá que cumprir ou implementar os métodos da interface. Por exemplo, comportamentos de um veículo incluem acelerar, frear, virar à direita, virar à esquerda. Mas podemos ter veículos que fazem esta mesma função de uma forma diferente.

Algumas diferenças entre classes abstratas e interfaces são listadas abaixo:

- Classes abstratas podem conter algumas implementações. Interfaces não têm implementações;
- Classes abstratas podem herdar outras classes e interfaces. Interface só pode herdar outra interface;
- Classes abstratas possuem construtores e destrutores. Interfaces não possuem nenhum dos dois.

Sobrecarga de métodos: Às vezes, você cria vários métodos que realizam funções muito próximas sob diferentes condições. Por exemplo, imagine métodos que calculem a área de um triângulo. Um método poderia tomar as coordenadas cartesianas dos três vértices, e um outro poderia tomar as coordenadas polares. Um terceiro método poderia tomar os comprimentos de todos os três

lados, enquanto um quarto poderia tomar três ângulos e o comprimento de um lado. Todos esses métodos poderiam realizar a mesma função essencial, e assim, é inteiramente apropriado usar o mesmo nome para os métodos sendo que a seqüência e o tipo de parâmetros definem qual método usar.

```
public void areaTriangulo(int vert1x, int vert1y, int vert2x, int vert2y,
                          int vert3x, int vert3y);
public void areaTriangulo(int polar1x, int polar1y, int polar2x, int polar2y);
public void areaTriangulo(int comp1, int comp2, int comp3);
public void areaTriangulo(double ang1, double ang2, double ang3, int comp1);
```

Sobrescrita de métodos: Quando herdamos uma classe podemos reescrever um método que foi implementado na classe base. Por exemplo um método implementado na classe Veiculo pode ser reescrito na classe VeiculoAutomotor mas para isto o método precisa ser criado com o atributo virtual. Então entendemos que podemos garantir que uma método não será reescrito na classe que herdar nossa classe apenas não definindo o método como virtual, ou vice-versa.

```
public class Veiculo {
    public virtual float getAceleracao() {
        //código
    }
}

public class VeiculoAutomotivo : Veiculo {
    public override float getAceleracao() {
        //novo código
    }
}
```

Encapsulamento: É fácil entendermos encapsulamento se pensarmos que o objetivo de uma cápsula é proteger e impedir que um objeto seja visto ou modificado fora dela. Quando definimos um atributo ou método como private, impedimos que seja visto ou acessado fora da classe. Para acessá-los, criamos métodos públicos que podem ser acessados externamente e garantimos a integridade dos dados.

```
public class Veiculo {

    private double dblVelocidadeAtual;

    public double VelocidadeAtual
    {
        get
        {
            return dblVelocidadeAtual;
        }
        set
        {
            // validações
            dblVelocidadeAtual = value;
        }
    }
}
```

Capítulo 8

Strings

A classe string possui diversos métodos que nos permitem manipular strings(texto). As vezes precisamos desmembrar uma string ou então colocá-lo em letras maiúsculas ou minúsculas. Vamos estudar alguns deste métodos a seguir.

Join: Usado para juntar partes distintas de um array de string.

```
string[] teste = new string[] { "Este é", "um teste", "do comando", "Join" };
string x = string.Join(" ", teste);
```

Sintaxe: `string.Join(delimitador; teste);`

Delimitador: É o caractere ou espaço em branco utilizado para separar palavras contidas em um Array.

Array: É a matriz que contém a seqüência de frases ou palavras utilizadas com o método join.

Split: O método Split é o oposto de Join. Utilizamos um delimitador para separar a string e partes.

Ex.:
```
string teste = "Este é um teste do comando, Split";
    string[] obj = teste.Split(',');
```

Length: Retorna o comprimento de uma string.

Ex.:
```
string teste = "Este é um teste do comando length";
    int tam = teste.Length;
```

Remove: Exclui um intervalo específico de uma string.

Sintaxe: varialveString.Remove(índice, quantidade);

Índice: É o índice do primeiro caracter da string a partir do qual o texto será excluído. (começa de 0);

Quantidade: É o número de caracteres que serão excluídos.

Ex.:
```
string frase = "Este é um pequeno teste do comando Remove";
    string teste = frase.Remove(10, 7);
    // o valor da variável teste é "Este é um teste do comando Remove".
```

Replace: O método Replace tem a função de substituir uma string por outra previamente definida.

Sintaxe: variável.Replace(substituir, substituído);

Substituir: É o texto antigo que deve ser substituído.

Substituído: É o novo texto que substitui o anterior.

Substring: Este método tem a função de encontrar trechos de uma string de acordo com os índices especificados.

Sintaxe: variavel.Substring(inicio, quantidade).

Inicio: Índice da posição inicial começando de 0.

Quantidade: Parâmetro que define a quantidade de caracteres a copiar.

Ex.:
```
string teste = "Este é um teste do comando Substring";
    string comando = teste.Substring(28, 9);
    // a variável comando terá o valor "Substring".
```

Concat: Concatena uma ou mais instâncias de uma string evitando o desperdício de memória.

Sintaxe: string.Concat(string1, string2);

Ex.:
```
string teste = "Adenízio";
string teste2 = " Carlos";
string teste3 = string.Concat(teste, teste2);
// A variável teste3 terá o valor "Adenízio Carlos";
```

Insert: Insere uma string no local especificado.
Sintaxe: variávelString.Insert(índice, texto).
Índice: A posição onde será inserido o texto.
Texto: A frase ou palavra que será inserida.

Ex.:
```
string teste = "Este é um teste";
teste.Insert(15, " do comando Insert");
```

Compare: Utilizado para comparar duas strings.
Sintaxe: string.Compare(string1, string2);

Se o valor retornado for menor que zero, então a string1 é menor que a string2.
Se o valor retornado for igual a zero, então a string1 é igual a string2.
Se o valor retornado for maior que zero, então a string1 é maior que a string2.

ToUpper e ToLower: O método ToUpper converte uma string em letras maiúsculas enquanto o método ToLower converte uma string para letras minúsculas.

Ex.:
```
string teste = "adenízio";
string maiúscula = teste.ToUpper(); // ADENÍZIO
string minúscula = teste.ToLower(); // adenízio
```

StringBuilder: Como vimos no uso do método Concat, uma string é imutável. Usando Concat nós evitamos a criação de várias instâncias de um objeto string na memória no momento de criação do mesmo. Porém quando queremos inserir ou alterar o valor de uma string, os métodos disponíveis retornam um outro objeto string. Para que possamos manipular uma string sem que haja a necessidade de criarmos outros objetos string devemos trabalhar com um objeto chamado StringBuilder. Este objeto está no namespace System.Text que deverá ser declarado no início da classe. Para instanciarmos um objeto StringBuilder seguiremos o padrão da POO.

```
// Criando uma instância do objeto StringBuilder com um buffer de 16 bytes.
StringBuilder str = new StringBuilder();
```

Para inserirmos os dados em um StringBuilder podemos usar os métodos:
Append: Inseri uma nova linha ao final do StringBuilder.
AppendFormat: Inseri uma nova linha formatada ao final do StringBuilder.
AppendLine: Inseri uma nova linha ao final do StringBuilder acrescida um CarriageReturn.

Quando queremos inserir um conteúdo à string em uma determinada posição usamos o método insert do StringBuilder.

```
str.Insert(10, "teste"); // Insere a palavra teste na posição 10;
```

Capítulo 9

ADO.NET

O ADO.NET foi criado pensando-se exclusivamente em aplicações Web. A primeira evidência disto é o fato de que o ADO.NET é um mecanismo para acesso a dados baseado em XML que é o formato padrão para tráfego de informações na Web.

Também é possível e bastante recomendável o trabalho com dados desconectados da origem. Nós veremos a seguir que é possível criar um banco em memória com recursos do ADO.NET.

É possível utilizar ADO.NET com qualquer linguagem disponível na plataforma.

DataProvider: Para começarmos a trabalhar com o ADO.NET temos que especificar um Provedor de Dados válido. Embutido na Framework do .NET já possuímos 2 tipos de provedores sendo o SQL para trabalharmos com banco SQL-Server e o OleDb para a versão 6.5 do SQL-Server ou para outros bancos de dados.

Existem outros provedores disponíveis para outros bancos de dados como é o caso do Oracle, Firebird e o DB2. Falaremos sobre conexões utilizando SQL, OleDb e Oracle.

Quando utilizamos provedores específico como o caso do SQL e do Oracle temos uma performance melhor por não haver uma camada intermediária. No caso do OleDb existe a facilidade de podermos acessar praticamente qualquer tipo de fonte de dados como os 2 citados anteriormente além de Access e XML mas por outro temos uma diminuição da performance visto que existe uma camada intermediária entre a aplicação e o banco de dados.

Todas as classes que utilizam o provedor SQL tem o prefixo Sql como SqlConnection, SqlCommand. Da mesma forma que as classes que utilizam Oracle tem o prefixo Oracle como OracleConnection e OracleCommand.

O provedor de dados SQL é referenciado pelo namespace System.Data.SqlClient e o Oracle por System.Data.OracleClient.

No caso do OleDb existe o uso do prefixo OleDb e o namespace System.Data.OleDb.

DbConnection(Sql, OleDb e Oracle): No ADO.NET utilizamos o objeto Connection para efetuar uma conexão com uma origem de dados. Para SQL usa-se SqlConnection, para Oracle OracleConnection e para OleDb OleDbConnection.

Ex.:
(Sql-Server):

```
string strConn = “DataSource=localhost; Integrated Security= SSPI; Initial Catalog=Nwind”;
SqlConnection conn = new SqlConnection(strConn);
conn.Open();
```

(Oracle):

```
string strConn = “Data Source=localhost; User ID=nome; Password=123; “+
                 “Persist Security Info=True;”;
OracleConnection conn = new OracleConnection(strConn);
conn.Open();
```

(Outros bancos):

```
string strConn = “Provider=MSDAORA.1; DataSource=localhost; User ID=nome;”+
                 “Password=123”;
OleDbConnection conn = new OleDbConnection(strConn);
conn.Open();
```

DbCommand(Sql, OleDb e Oracle): Feita a conexão com a fonte de dados podemos executar comandos e fazer consultas utilizando o objeto DbCommand e especificando a instrução SQL que deverá ser executada. Para Sql-Server (SqlCommand), Oracle (OracleCommand) e OleDb (OleDbCommand). É importante citar que o DbCommand deve ser do mesmo tipo do DbConnection, ou seja, se usarmos SqlConnection o command terá que ser SqlCommand.

As principais propriedade de DbCommand são:

ExecuteNonQuery: Executa um comando sem consulta e retorna a quantidade de registros afetados pelo comando. Por exemplo quando executamos um comando de update, não existe retorno de dados senão a quantidade de registros que foram alterados no banco de dados.

```
private int update()
{
   string strConn = "DataSource=localhost; Integrated Security= SSPI; Initial Catalog=Nwind";
   SqlConnection conn = new SqlConnection(strConn);
   conn.Open();

   string sql = "Update TABELA set CAMPO = VALOR "+
                " Where CONDICAO1 = PARAMETRO1";
   SqlCommand cmd = new SqlCommand(sql, conn);
   int cont = Convert.ToInt32(cmd.ExecuteExecuteNonQuery());
   conn.Close();
   return cont;
}
```

ExecuteScalar: Retorna o valor da primeira coluna do primeiro registro. Utilizado quando queremos fazer uma consulta que retorna um único valor. Por exemplo quando fazemos um consulta que retorna um count ou max.

```
 private int qtde()
 private int qtde()
{
   string strConn = "DataSource=localhost; "+
                    "Integrated Security= SSPI; Initial Catalog=Nwind";
   SqlConnection conn = new SqlConnection(strConn);
   conn.Open();
   string sql = "Select count(codigo) from TABELA"+
                " where CONDICAO1 = " + PARAMETRO1;
    SqlCommand cmd = new SqlCommand(sql, conn);
    return Convert.ToInt32(cmd.ExecuteScalar());
}
```

ExecuteReader: Executa a leitura de dados em um banco de dados e retorna um DataReader como resultado.

```
 public SqlDataReader exibeRegistro()
{
   string strConn = "DataSource=localhost; Integrated Security= SSPI; Initial Catalog=Nwind";
   SqlConnection conn = new SqlConnection(strConn);
   conn.Open();
   string sql = "Select codigo, nome from produtos where secao = 1";
    SqlCommand cmd = new SqlCommand(sql, conn);
    SqlDataReader dr = cmd.ExecuteReader();
    return dr;
 }
```

Inserção, atualização e deleção

Existem 3 maneiras distintas de fazermos estas operações. Veremos as 3 e ficará a sua escolha de acordo com a situação qual lhe será a melhor.

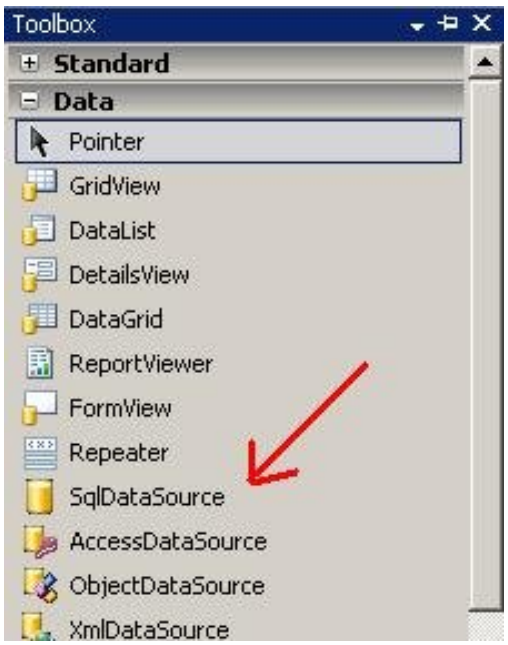
1. A primeira maneira é utilizarmos o objeto command para executar os comandos no banco de dados. Por exemplo ao inserirmos uma string sql com o comando a ser executado. (Ver ExecuteNonQuery).

Mas para usarmos a execução devemos trabalhar com parâmetros no command. O tipo do parâmetro está diretamente relacionado ao tipo do command, ou seja, se usarmos um SqlCommand devemos usar um SqlParameter, no caso do OracleCommand, OracleParameter e assim por diante.

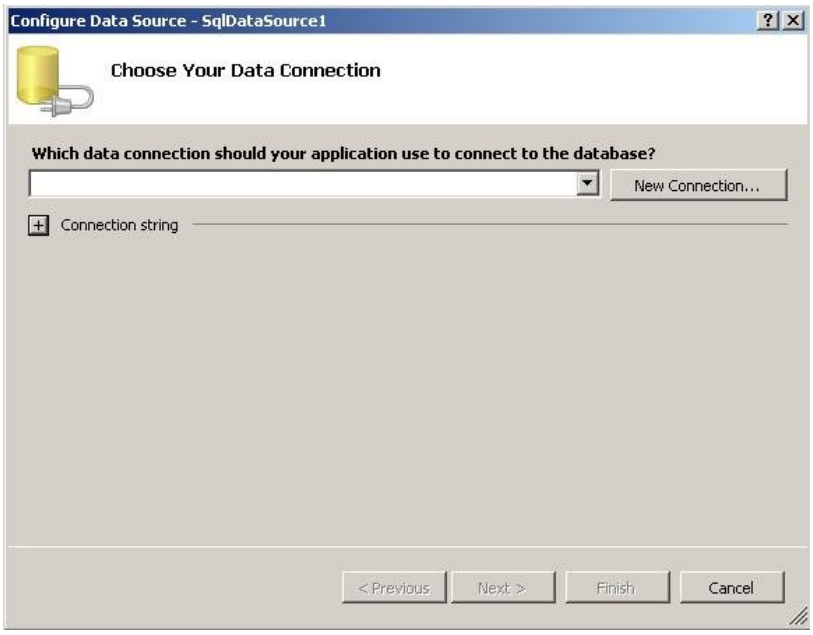
Vamos a um exemplo prático:

```
...
  string strConn = "DataSource=localhost; Integrated Security= SSPI; Initial Catalog=Nwind";
   SqlConnection conn = new SqlConnection(strConn);
   conn.Open();

   // Cria a string que terá o comando a ser executado no banco de dados.
   string sql = "Update produtos set nome = :nome, vlr = :vlr Where codigo = :codigo";

   // Cria e instancia um objeto SqlCommand
   SqlCommand cmd = new SqlCommand(sql, conn);

   // Cria os parâmetros no objeto SqlCommand
   cmd.Parameters.Add(new SqlParameter("nome", SqlDbType.Varchar, 50));
   cmd.Parameters.Add(new SqlParameter("vlr", SqlDbType.Decimal));
   cmd.Parameters.Add(new SqlParameter("codigo", SqlDbType.Int));

   // Atribui os valores aos respectivos parâmetros
   cmd.Parameters["nome"].Value = "Café Extra Forte";
   cmd.Parameters["vlr"].Value = 4;
   cmd.Parameters["codigo"].Value = 125;

   // Execute comando no banco de dados
   cmd.ExecuteNonQuery();
```

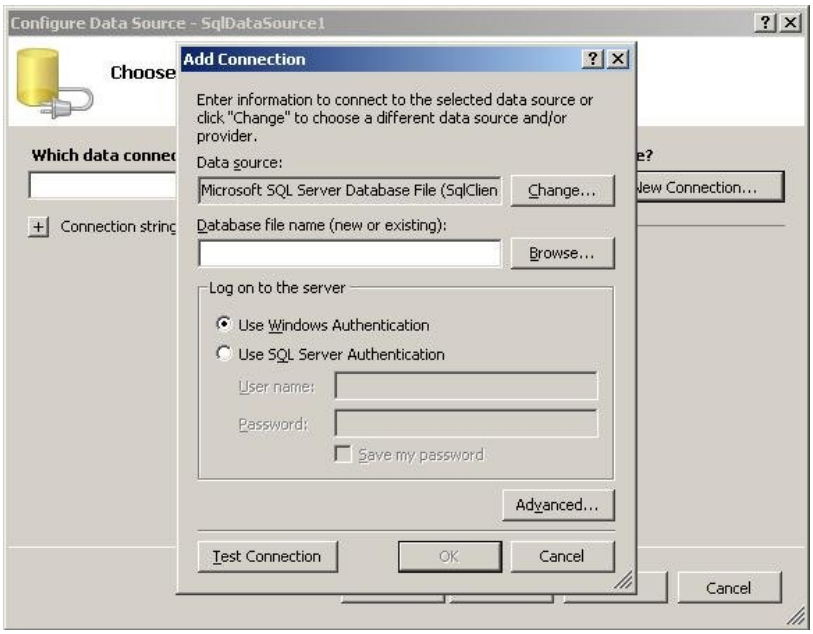
2. Existe um componente no .NET 2.0 chamado SqlDataSource:



Após arrastarmos um SqlDataSource para o palco, podemos clicar em ConfigureDataSource e teremos uma tela semelhante à próxima figura:

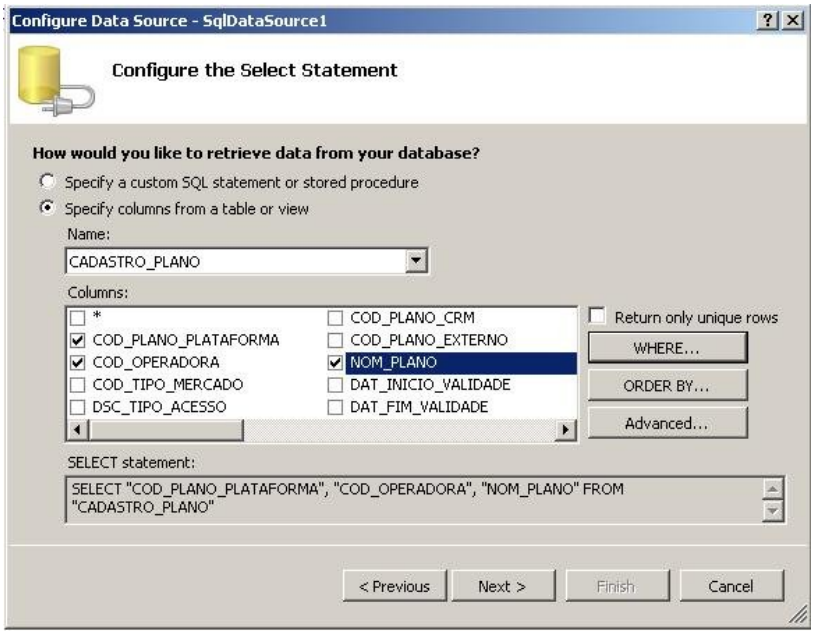


Se já tivermos uma configuração podemos selecioná-la no dropdown, caso contrário podemos clicar em New Connection...

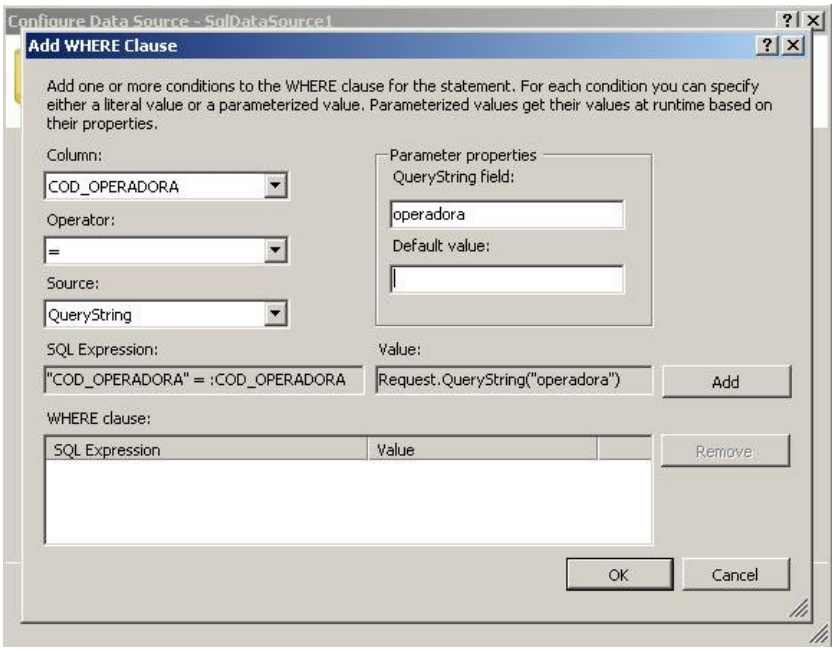


Aqui podemos escolher o tipo de conexão(Sql, Oracle, OleDb, etc.), a base de dados e o tipo de login. Ao clicarmos em OK, retornaremos para a tela anterior com a string de conexão cadastrada. Você terá a opção de incluir esta string no arquivo web.config que veremos no próximo capítulo.

Na próxima tela teremos as opções de montar a consulta selecionando os campos de uma determinada tabela ou de uma forma customizada.



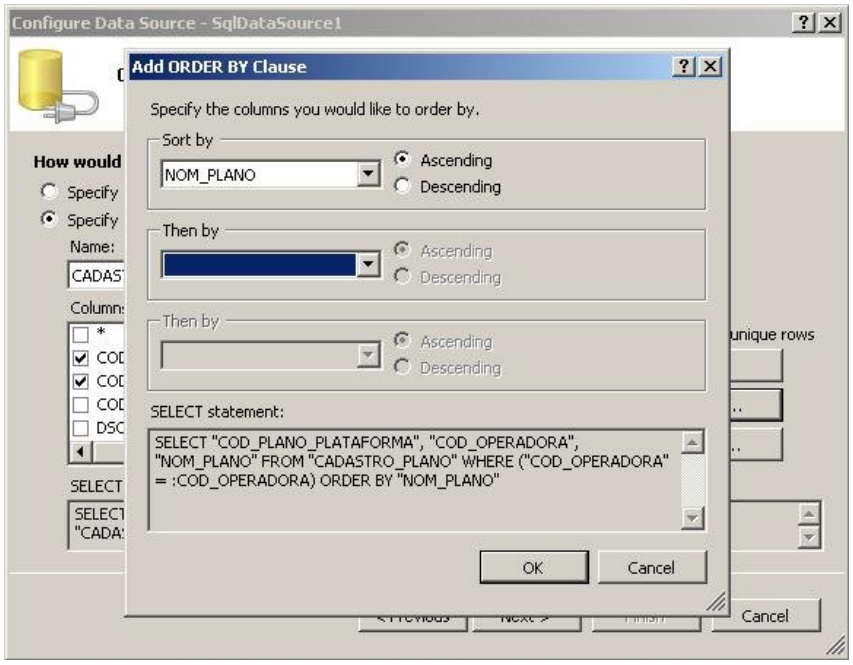
No caso de selecionarmos os campos, podemos clicar no botão WHERE... para colocarmos as condições para a consulta.



Observe que podemos selecionar a coluna, o operador e a fonte que pode ser: None, Control, Cookie, Form, Profile, QueryString e Session. Após montarmos o parâmetro, podemos adicioná-lo á cláusula WHERE.

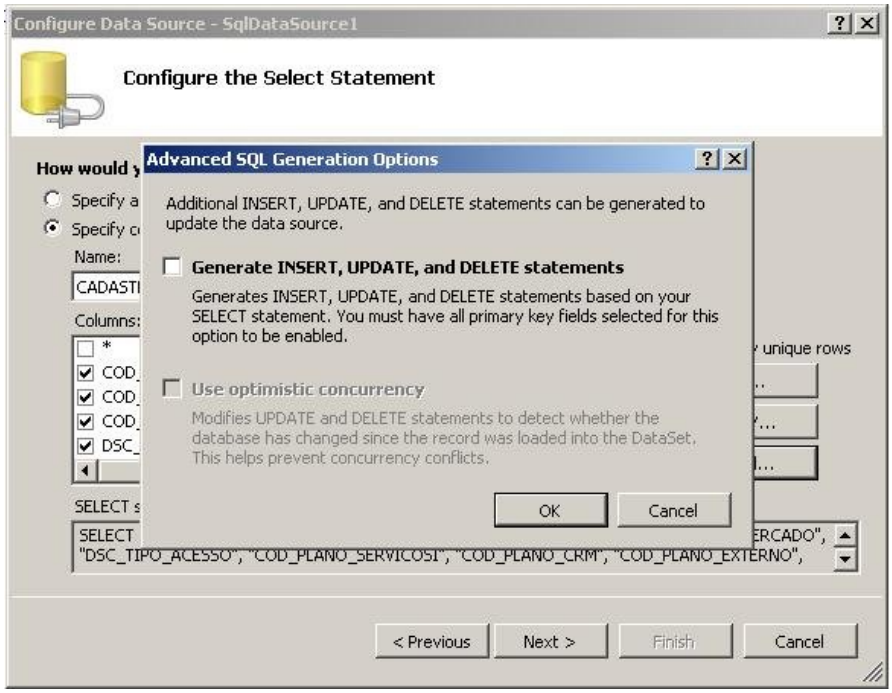
Depois de clicarmos em OK, voltamos para a tela anterior e podemos clicar em ORDER BY...

Nesta tela definimos os campos que serão usados de base para ordenar a consulta e a forma que será feita a ordenação também, sendo Ascending / Crescente ou Descending / Decrescente.



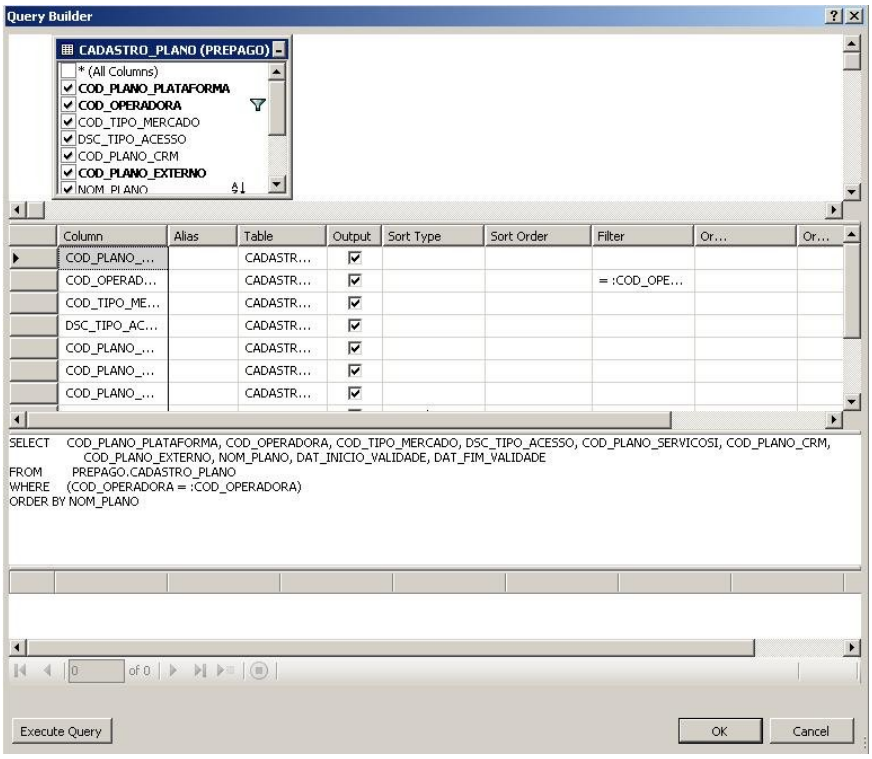
Ao voltarmos para a tela anterior ainda podemos clicar em Advanced...

Nesta tela podemos selecionar a opção Generate INSERT, UPDATE, and DELETE statements para que gere os comandos automaticamente.

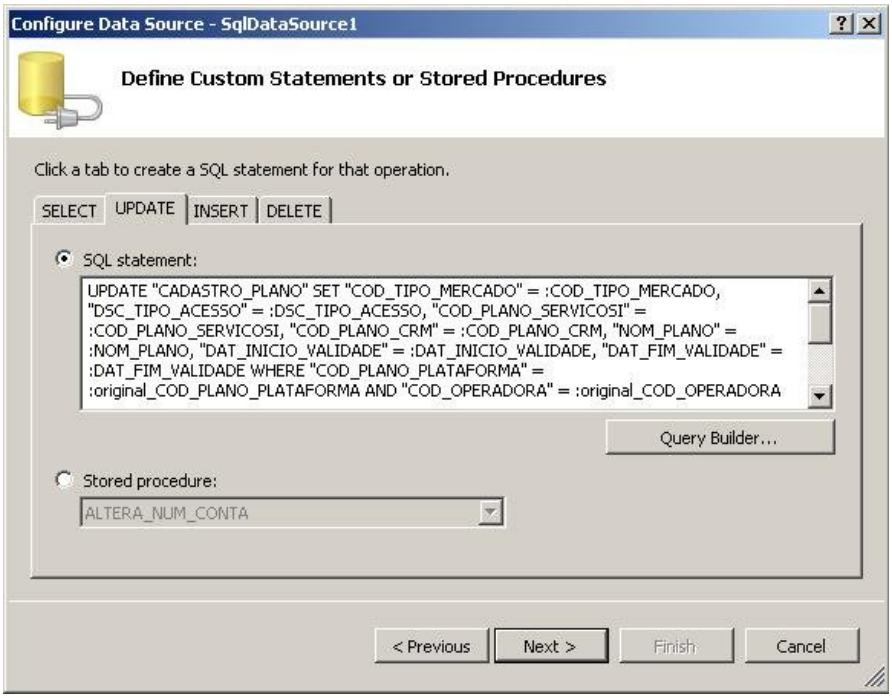


Podemos também selecionar a opção Use optimistic concurrency para que ele verifique antes do update e do delete se os valores foram mudados no banco antes executar o comando.

Podemos trabalhar de uma outra forma também que é escolhendo montar a consulta de uma forma customizada. Neste caso selecionamos a opção Specify a custom SQL e teremos a opção de digitarmos a consulta ou usar o QueryBuilder conforme a próxima figura:



Neste caso podemos selecionar uma ou mais tabelas, criarmos relacionamentos e definirmos os campos que farão parte das condições de consulta.



Observe que temos uma aba para cada comando: SELECT, UPDATE, INSERT, DELETE além da opção de selecionarmos uma Storedprocedure previamente compilada no banco de dados.

Depois é clicar em Next e Finish.

Entre as vantagens de se trabalhar com um SqlDataSource destacam-se o fato de montarmos a consulta de uma forma visual, sem que seja necessário digitar nenhuma linha de código, a definição dos parâmetros e seus campos para atualização sendo que não será necessário escrever um código para a atualização e o fato de que podemos trabalhar com objetos como o GridView ou FormsView que estão diretamente vinculados ao SqlDataSource.

3. A terceira forma veremos mais adiante quando estudarmos DataAdapter de DataSet.

Stored Procedures: Uma Stored Procedure é composta de comandos SQL, variáveis e comandos de fluxo lógico.

Como fica armazenada no banco de dados, a manutenção do sistema fica mais simples pelo fato de uma única alteração afetar imediatamente todos os usuários. Se uma Stored Procedure já foi executada, os acessos seguintes são mais rápidos, pois seu plano de execução fina na memória.

As Stored Procedures aceitam parâmetros de entrada e podem retornar valores como parâmetros de saída para o programa que as invocou.

Mesmo retornando parâmetros de saída, as Stored Procedures não podem ser utilizadas em expressões, pois são diferentes de funções.

Parâmetros: Por meio de parâmetros é possível estabelecer uma conexão entre o mundo exterior e a Stored Procedure. As principais definições de um parâmetro são: nome, tipo (int, varchar, etc.), valor, direção (input, output).

DbDataReader(Sql, OleDb, Oracle): Um DataReader é um componente do tipo RO/FO (Read Only/Forward Only), ou seja, (Somente leitura/Somente pra frente). Significa dizer que só podemos lêr e não gravar dados e só podemos navegar para frente, não podendo assim voltar aos registros anteriores.

```
...
   string strConn = "DataSource=localhost; Integrated Security= SSPI; Initial Catalog=Nwind";
   SqlConnection conn = new SqlConnection(strConn);
   conn.Open();
   string sql = "Select codigo, nome from produtos where secao = 1";
    SqlCommand cmd = new SqlCommand(sql, conn);
    SqlDataReader dr = cmd.ExecuteReader();
    while (dr.Read())
    {
       varcodigo = dr.GetInt32(0);  // Aqui usamos o GetInt que já converte para inteiro.
       varnome = dr["nome].ToString(); // Neste caso usamos o nome da coluna;
       // Obs.: Quando usamos Get + tipo, só podemos passar o índice da coluna e não o nome.
  }
...
```

DbDataAdapter(Sql, OleDb, Oracle): A principal função da interface IDbDataAdapter é mediar a interação entre a classe DataSet e o bancos de dados reais, para isso a classe que implementa IDbDataAdapter contém um comando Sql de consulta, usualmente um comando SqlSelect. Este comando é usado para preencher uma tabela dentro de um DataSet. Além disto ele possui os 3 comandos de atualização. Inserção, deleção e atualização.

O método Fill preenche o DataSet executando a conexão ao banco de dados, a consulta e a desconexão. Depois de alterado o DataSet chamamos o método Update para aplicar as modificações de volta ao banco de dados. Os comandos de alteração são gerados automaticamente mas podem ser alterados para atender alguma necessidade mais específica.

```
...
  string strConn = "DataSource=localhost; Integrated Security= SSPI; Initial Catalog=Nwind";
  SqlConnection conn = new SqlConnection(strConn);
  conn.Open();
  string sql = "Select codigo, nome from produtos where secao = 1";
  SqlDataAdapter da = new SqlDataAdapter(sql, conn); // Passamos o comando e a conexão
```

```
  // Escolhemos um objeto DataSet ou DataTable para popular
  DataSet ds = new DataSet() // ou DataTable dt = new DataTable();
  da.Fill(ds, “PRODUTOS”); // ou da.Fill(dt) no caso do DataTable
...
```

DataSet: Provavelmente a classe mais importante do ADO.NET. Funciona como um banco de dados em memória podendo conter várias tabelas e também relacionamento entre as tabelas. Estas podem conter campos de vários tipos, colunas calculadas, restrições e até chaves primárias. Um DataSet nunca está conectado a um banco de dados físico e nunca sabe a origem dos dados. É possível popular um DataSet utilizando um DataReader, DataAdapter, XML entre outros.

As tabelas em um DataSet são enxergadas como um Array e não através de cursor.

DataView: É uma visão dos dados em memória armazenados no DataSet. Com ele é possível fazer filtros e ordenações sem que seja necessário submeter uma nova consulta ao banco de dados.

Capítulo 10

Formulários e Controles Web

Cookies: Um cookie é uma informação que um servidor web pode armazenar temporariamente junto a um browser. Uma aplicação prática disto é no comércio via Internet. Suponhamos que alguém entre em uma loja virtual de discos, faça várias seleções para compra e vá navegar em outros sites. Ao voltar ao site de venda de cds, todas as suas seleções terão sido mantidas e ele poderá então fechar o seu negócio ou fazer mais aquisições.

As informações são guardadas pelo browser e não pelo servidor Web, o que não deixa de fazer sentido. Ficaria muito mais difícil para um servidor se lembrar dos milhares de browsers que o acessaram recentemente e exatamente o que cada um deles fez ou selecionou.

Os cookies são enviados para o seu browser e mantidos na memória. Ao encerrar a sua sessão com seu browser, todos os cookies que ainda não expiraram são gravados em um arquivo (cookie file).

Muitas pessoas julgam que os cookies possam ser usados pelo servidor para obter informações a seu respeito ou invadir o seu disco rígido e obter dados a partir de lá, o que não é verdade. Todas as informações gravadas em um cookie são informações que você forneceu voluntariamente ao servidor, de uma forma ou de outra.

Para criar um cookie, o servidor web envia uma linha de cabeçalho HTTP em resposta a um pedido de acesso a uma URL solicitada pelo browser:

Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure

NAME é o nome do valor que se está armazenando no browser e VALUE é o dado real sendo armazenado no cookie. DATE é a data na qual este cookie irá expirar.

DOMAIN indica um computador ou rede na qual este cookie é válido. Computadores fora deste domínio não conseguirão ver este cookie.

A diretiva "secure" indica que o cookie somente será transferido sobre conexões seguras (https) e nunca sobre uma conexão http normal.

De todos estes campos, apenas o campo NAME é obrigatório.

Bom, voltando ao browser. Sempre que um browser solicita uma URL a um servidor que nele tenha criado cookies anteriormente, é incluída, juntamente com a URL uma linha listando todos os cookies existentes. Esta informação será então utilizada pelo servidor Web para dar continuidade a transações iniciadas anteriormente. Esta linha possui um formato do tipo:

Cookie: NAME=VALUE; NAME=VALUE; ...

Objetos Application e Session: O objeto Application foi criado para armazenar propriedades (valores) ligados a um conjunto de usuários. No caso, os visitantes do site, de um modo geral. Como exemplo, podemos citar o número total de visitantes no site a partir de uma determinada data, ou o número de visitantes online no site.

O objeto Session foi criado para armazenar propriedades (valores) ligados a cada visitante, individualmente. Como exemplo, podemos citar o carrinho de compras de um site de comércio online. Uma Session é criada quando o visitante entra no site (cada visitante tem uma session e cada session recebe um ID), e é destruída quando o visitante sai do site (seja por logoff explícito ou por TimeOut). Já uma Application é iniciada ao haver o primeiro pedido de acesso ao objeto Application, e é encerrado quando o servidor for desligado.

Todo o código que se deseja executar ao criar ou destruir uma session, bem como uma Application devem estar contidos no arquivo global.asax, um arquivo texto no formato abaixo demonstrado, que deve ser colocado no diretório raiz do site.

As variáveis do objeto Application e do objeto Session são armazenadas no servidor, mas é necessário que o browser aceite cookies, pois um cookie com o ID da sessão é criado no computador do visitante, para identificá-lo.

Web.Config: Muitas vezes, você quer incluir configurações globais para sua aplicação, como uma string de conexão com uma base de dados. No ASP clássico (3.0), isso é possível de ser feito usando o arquivo global.asa. No .NET, existe um nova opção: o arquivo Web.Config file (XML-based), e uma seção especial chamada AppSettings.

O exemplo abaixo mostra um arquivo de configuração com uma string dsn.
No web.config:

```
<configuration>

<appSettings>
   <add key="PubsDSN" value="server=(local);database=pubs;uid=sa;pwd=;" />
</appSettings>

</configuration>
```

Na página:

```
<%@ Page Language="C#" %>

<html>
<body>
   <%=System.Configuration.ConfigurationSettings.AppSettings["PubsDSN"]%>
</body>
</html>
```

Global.asax: O Diretório Virtual no IIS é grande parte do Aplicativo ASP.NET. Independente da página, o aplicativo é iniciado na primeira vez que ela é solicitada. Enquanto os usuários navegam pelas páginas, o processamento ocorre em segundo plano para tratar do Aplicativo.

O Aplicativo pode cair e ser reiniciado da mesma forma que qualquer aplicativo tradicional, com a seguinte excessão: enquanto um aplicativo tradicional é iniciado e executado em um computador desktop permitindo a interação direta com o usuário, um Aplicativo ASP.NET é iniciado e executado em um servidor Web, e o usuário utiliza um browser para acessá-lo.
Lembrando que tudo em .NET Framework é um objeto, temos um objeto chamado *HttpApplication*, que nos fornece métodos e eventos. Com isso podemos deduzir que sempre que uma página em seu Diretório Virtual for solicitada pela primeira vez, um objeto do tipo *HttpApplication* é instanciado.

Como as páginas ASP.NET realizamos uma ou mais tarefas individualmente, elas não controlam o Aplicativo de modo geral, não podendo uma página afetar diretamente a outra. Deve existir uma localização central que controla a execução do Aplicativo. Tendo este cenário, entra em cena o arquivo *Global.asax*.

Conhecido como Arquivo de Aplicação de ASP.NET, o *Global.asax* permite-nos programar no lugar do objeto *HttpApplication* e com isso você poderá controlar o seu Aplicativo ASP.NET como faz com qualquer outro objeto por meio de métodos e eventos.

OBS.: Apesar do Visual Studio .NET incluir o arquivo *Global.asax* por default, ele é totalmente opcional. Se seu Aplicativo não conter um arquivo desse tipo, o programa opera de forma padrão. Desejando adicionar funcionalidades, a sua utilização torna-se essencial.

O arquivo *Global.asax* é colocado no Diretório raiz do Aplicativo (Exemplo: http://servidor/site/ - c:\inetpub\wwwwroot\site\). O ASP.NET controla o acesso à esse arquivo, de modo que ele não é acessível através do browser, o que garante a segurança.

Este arquivo é gerenciado pela .NET Framework. Sempre que ele for modificado, o CLR detecta isso e faz com que o Aplicativo seja reiniciado para que as novas mudanças tenham efeito. Isso pode ocasionar transtornos para os usuários finais. Modifique este arquivo somente o necessário.

Capítulo 11

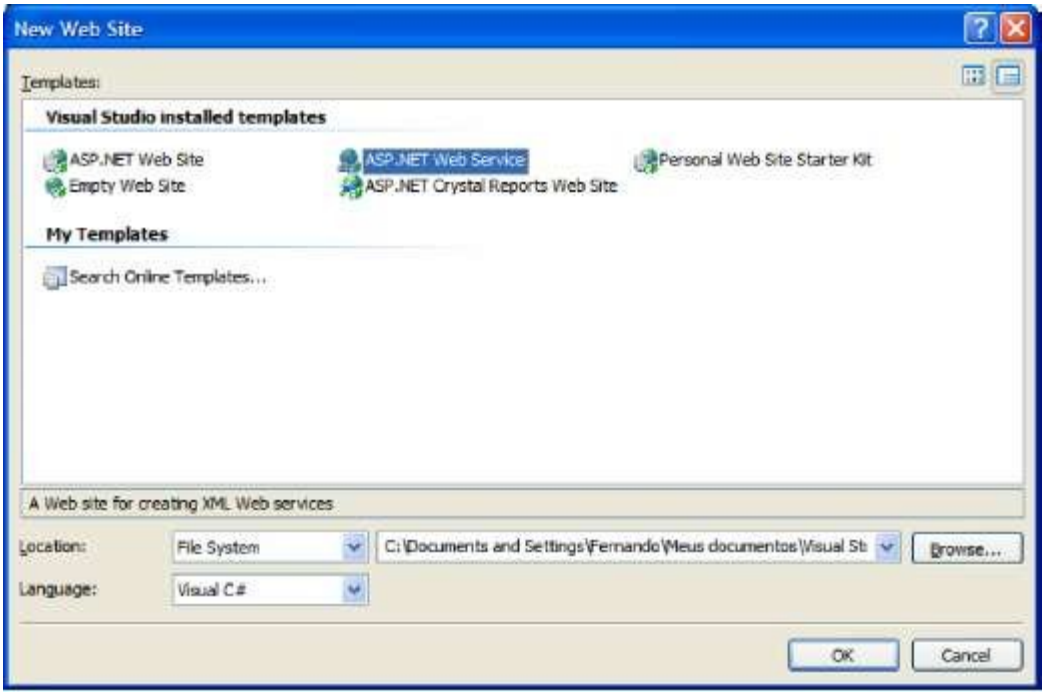
WebServices

O que são WebServices: São aplicações desenvolvidas com o intuito de fornecer serviços à outras aplicações que sejam desenvolvidas até mesmo em linguagens diferentes. Sendo assim, podemos usá-los na integração entre sistemas diferentes em plataformas diferentes.

Isso é possível porque WebServices recebem e enviam dados em formato XML (eXtensible Markup Language).

Os dados são transportados via protocolo HTTP ou HTTPS em formato XML e encapsulados pelo protocolo SOAP (Simple Object Access Protocol).

Criando um WebService: Para criarmos um WebService clique em File > New Web Site ...

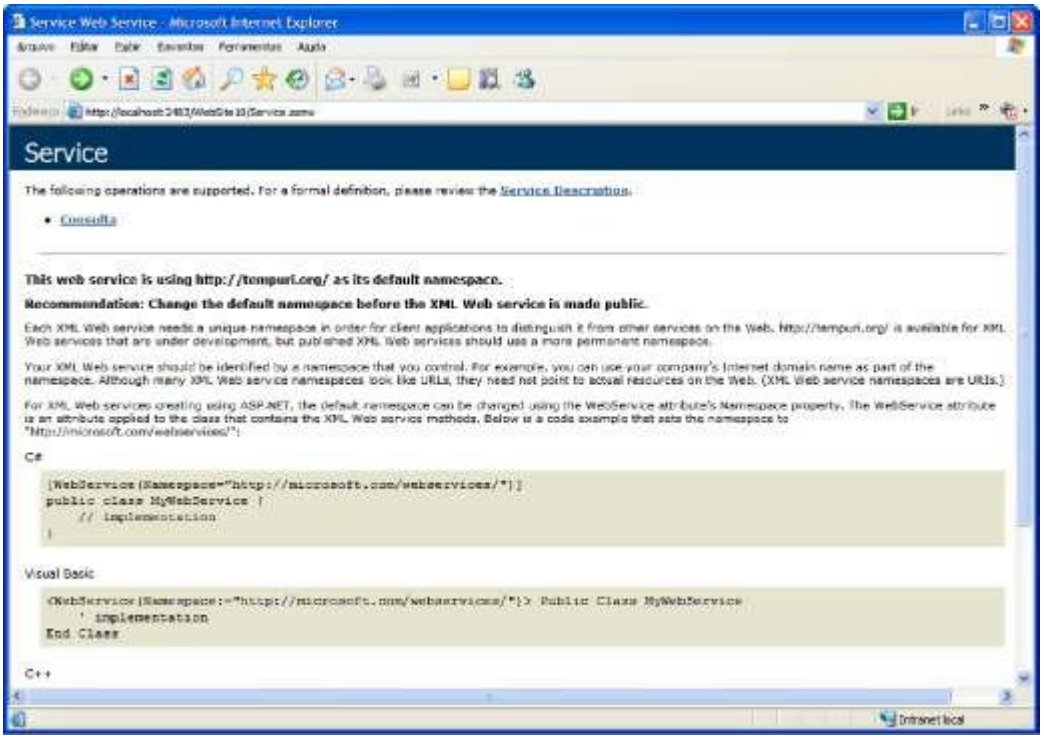


Por padrão é criado um arquivo asmx, que é equivalente a uma página aspx, porém sua função é prover serviços e não conteúdo. Cada arquivo asmx tem um arquivo .cs equivalente, onde são criados os métodos a serem publicados.

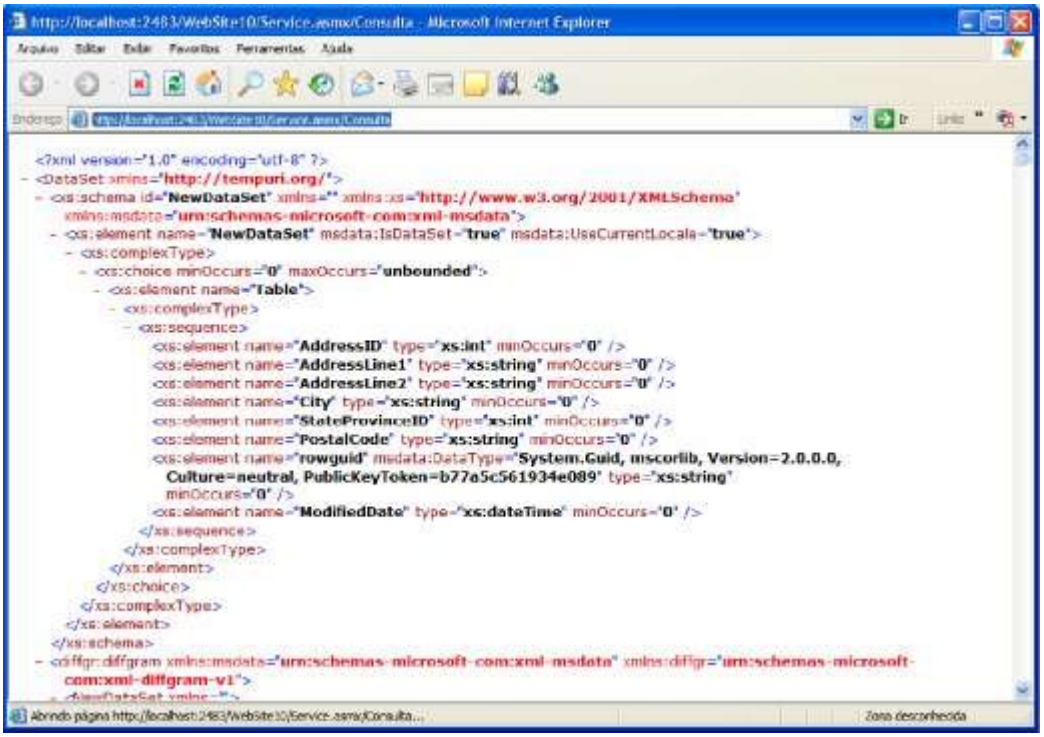
Um método deve conter um atributo WebMethod, caso contrário ele estará disponível dentro da aplicação mas não externamente a esta. Se você examinar o arquivo .cs criado, notará que já existe um método de demonstração criado automaticamente:

```
[WebMethod]
public string HelloWorld()
{
    return "Hello World";
}
```

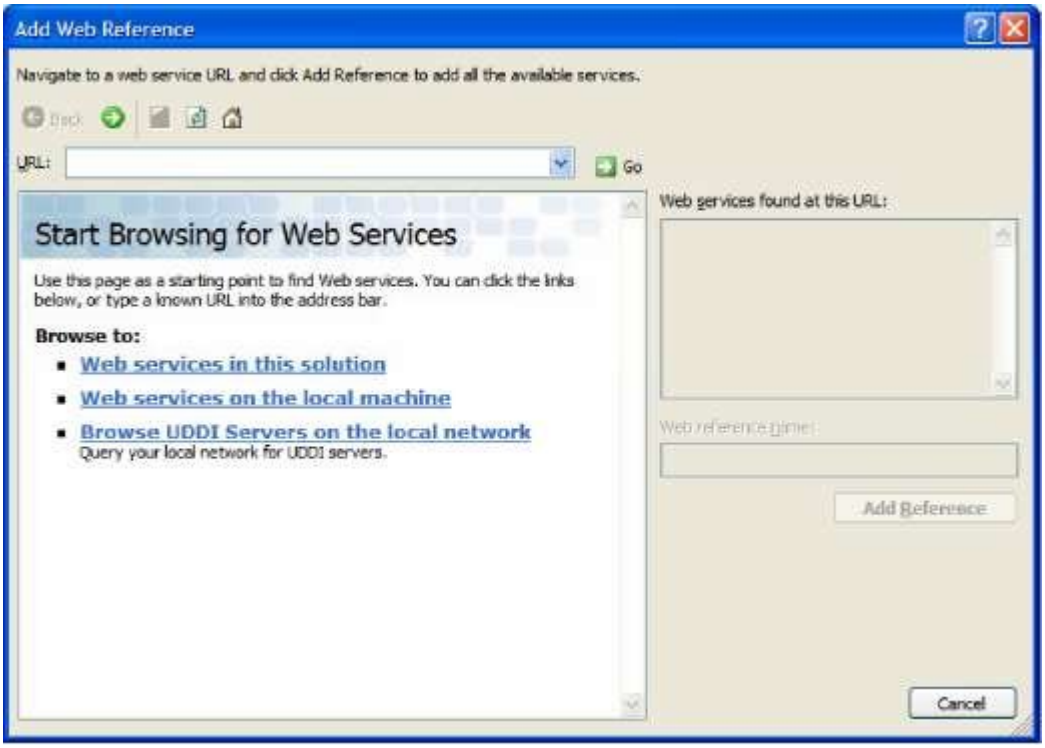
Na tela a seguir veremos um WebService sendo executado diretamente no Browser.



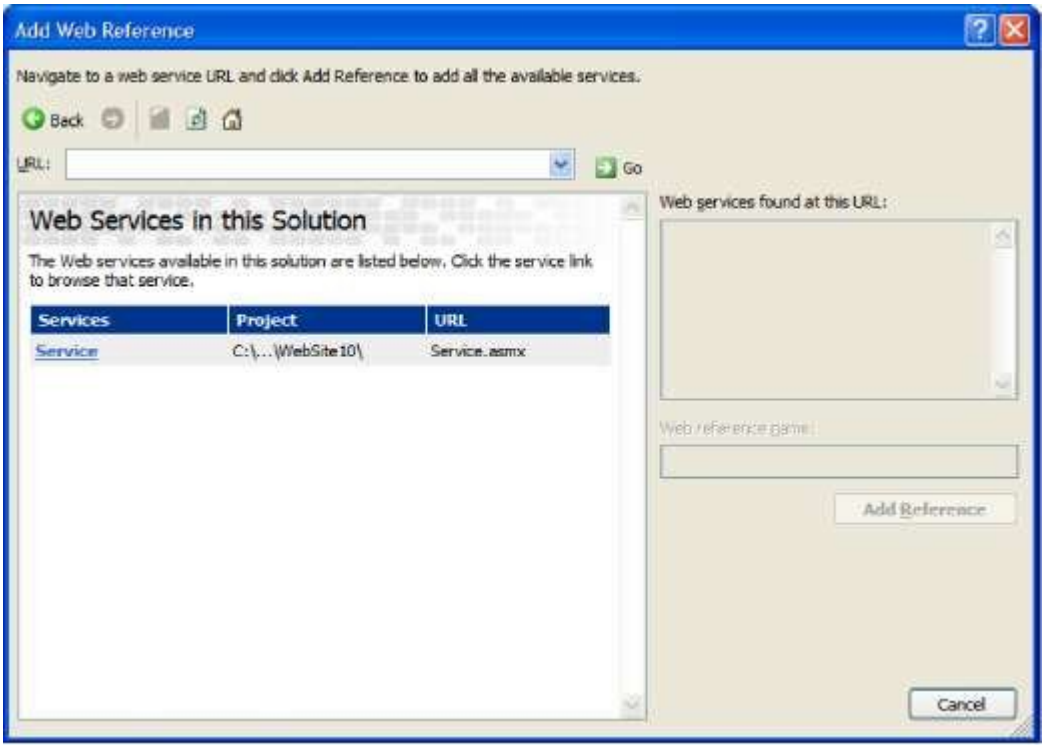
Observe que temos um método chamado Consulta. Se clicarmos no link seremos direcionados a uma outra página com um campo de texto e um botão. Este campo existe porque o método espera um parâmetro. Depois de clicar no botão seremos direcionado a uma outra página com o retorno do método Consulta.



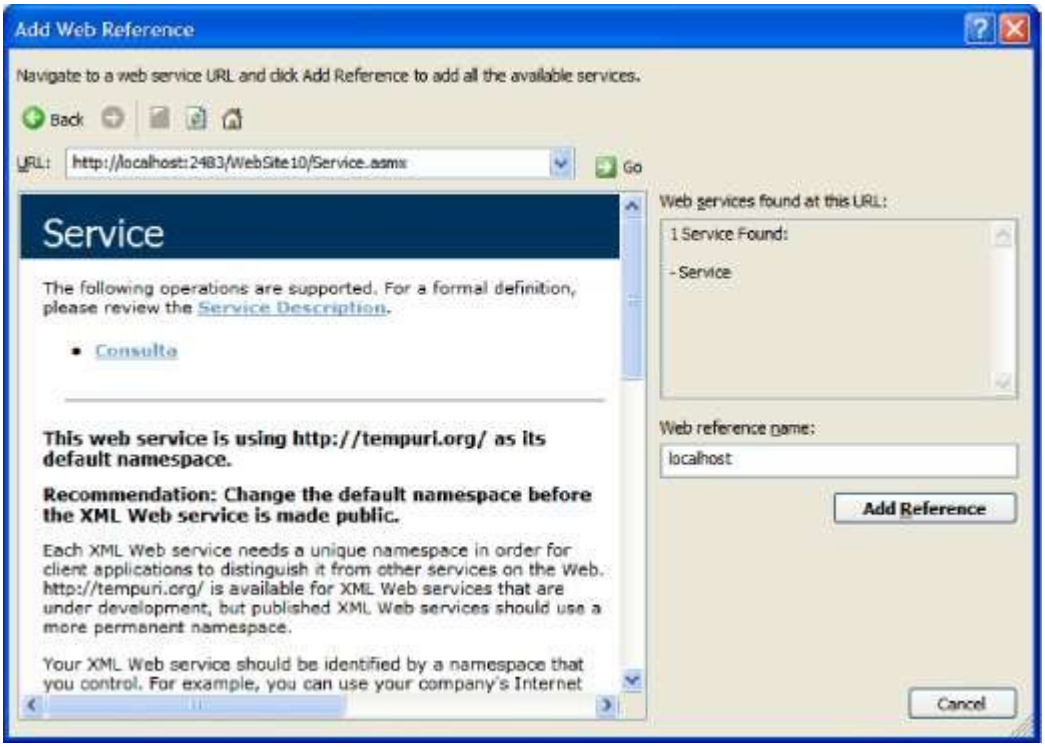
Consumindo um WebService: Para consumir este WebService, ou seja, usar os métodos dentro de uma aplicação devemos fazer primeiro uma referência ao WebService. Para isso devemos ir na SolutionExplorer e clicar com o botão direito do mouse sobre uma aplicação Asp.Net e depois com o botão esquerdo do mouse em Add Web Reference...



Podemos digitar uma url ou buscar um WebService em nossa solução.



O WebService que acabamos de criar será listado na tela. Ao clicarmos no link Service o Visual Studio fará uma busca de métodos no WebService e exibirá uma tela semelhante à tela vista no browser anteriormente. Depois podemos definir um nome de referência em Web reference name e clicar em Add Reference.



Agora basta inserir o código abaixo em nossa aplicação Asp.Net:

```
using localhost;
...
protected void Page_Load(object sender, EventArgs e)
{
    Service Con = new Service();
    DataSet ds = Con.Consulta("SELECT * FROM Tabela") ;
    Gridviewl.DataSource = ds;
    GridViewl.DataBind();
}
```