

# Capítulo 11

## WebSnap

O *WebSnap*, presente desde a versão 6 do Delphi, e a versão 2 do Kylix, tem como proposta e objetivo principal o desenvolvimento de aplicações para Internet com o conceito RAD (Rapid Application Development), ou desenvolvimento rápido de aplicações. Na realidade não foi incorporada para substituir o *WebBroker*, e sim complementar e estender suas funcionalidades.

Uma das coisas mais interessantes nesta tecnologia é a possibilidade de utilizar inúmeros formulários para uma mesma aplicação. Em *WebBroker* não era possível, pelo menos não de maneira organizada e documentada. Na tecnologia *WebBroker* as equipes de desenvolvimento precisavam criar projetos diferentes para produzir de maneira adequada. Com o *WebSnap* essa barreira foi quebrada, permitindo compartilhar diversas *units* entre as equipes.

A tecnologia *WebSnap* também permitiu o desenvolvimento de outros *frameworks* bastante produtivos e que estão fazendo o maior sucesso entre os desenvolvedores. Prova disso foi a incorporação de um desses *frameworks* nesta versão do Delphi: o IntraWeb, que veremos no próximo capítulo.

### Algumas curiosidades



O modelo de desenvolvimento *Web* no Delphi vem crescendo a cada versão, provando a forte tendência deste modelo. Grandes empresas estão partindo para este modelo de desenvolvimento, justamente pela portabilidade, economia e praticidade. Sinceramente, sugiro que todos explorem o máximo este modelo de desenvolvimento, pois o mercado está indo para este lado.

## Conhecendo os componentes

Embora a melhor maneira de conhecer os componentes do *WebSnap* seja praticando alguns exemplos, darei um breve descritivo de cada um deles.

A figura 11.1 ilustra a paleta de componentes do WebSnap.

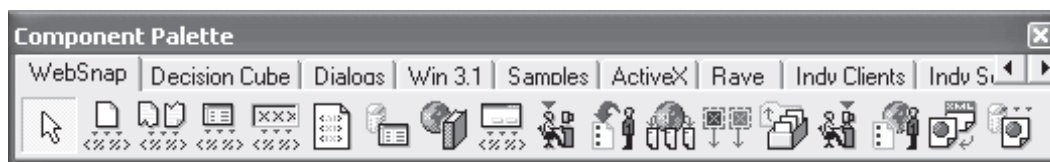



















Figura 11.1 Componentes WebSnap

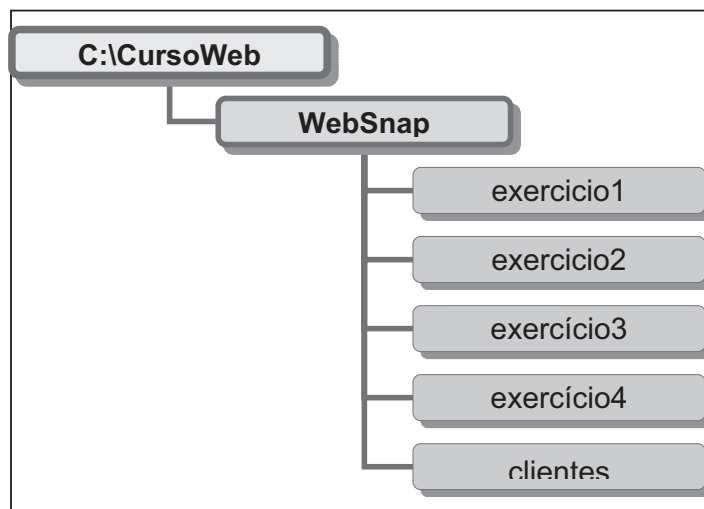
Componente	Descrição
 TAdapter	Fornecer uma interface de scripts e execução de comandos
 TPagedAdapter	Fornecer uma camada lógica para divisão de páginas de dados. Tecnologia comumente encontrada em sites de busca.
 TDataSetAdapter	Fornecer soluções de controles e comandos para DataSets. Adiciona funções semelhantes aos Adapters, relacionando com os DataSets.
 TLoginFormAdapter	Utilizado para criar formulários de Login.
 TStringsValuesList	Fornecer aos <i>Adapters</i> uma lista de nomes e valores. Exemplo: Nome e Senha, Código e Razão Social, e assim por diante
 TDataSetValuesList	Semelhante ao TStringsValuesList, mas vinculado a um DataSet.
 TWebAppComponents	Principal componente da tecnologia <i>WebSnap</i> . Fornece toda a interface para o desenvolvimento de aplicações <i>WebSnap</i> .
 TApplicationAdapter	Fornecer interface de scripts relacionados à informação da aplicação.
 TEndUserAdapter	Controle de informação sobre o usuário, como Nome, ID, sessão.
 TEndUserSessionAdapter	Controle completo sobre a informação de sessão do usuário. Normalmente utilizado com TSessionsService e TEndUserAdapter.

Componente	Descrição
 TPageDispatcher	Responsável pela interface de <i>request</i> HTTP de aplicações <i>WebSnap</i> . É necessário apenas um por aplicação.
 TAdapterDispatcher	Fornecer interface de tratamento de submissões de formulários HTML e imagens. É necessário para que a aplicação trabalhe com os adapter actions, além de imagens provenientes de adapter fields.
 TLocateFileService	Fornecer controle sobre arquivos de template e arquivos script-server-side. Facilita a manipulação destes arquivos.
 TSessionService	Armazena informações de sessões temporariamente. Possui TimeOut, número máximo de sessões, entre outras funcionalidades.
 TWebUserList	Controla uma lista predefinida de usuários, fazendo validação de login e controle de acesso.
 TXSLPageProducer	Faz um <i>parse</i> em documentos XML, baseados no template XSL.
 TAdapterPageProducer	Muito utilizado na tecnologia <i>Web</i> , cria um documento HTML com diversas funcionalidades e controles.

## Preparando o ambiente para os exemplos

Neste ponto vamos preparar nosso ambiente para o desenvolvimento das aplicações baseadas na tecnologia *WebSnap*.

Crie a seguinte estrutura de diretórios (*diagrama 11.1*)

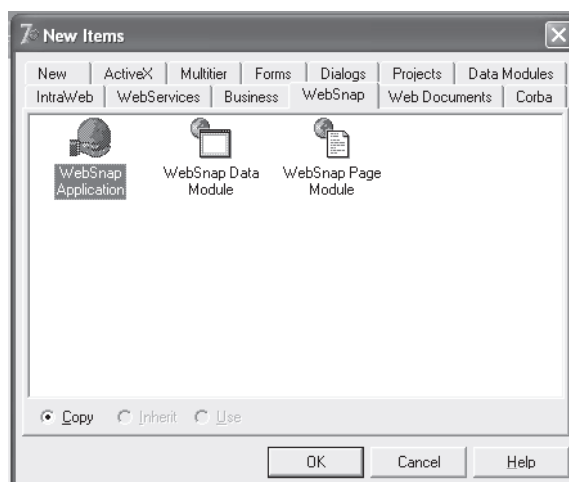


*Diagrama 11.1*

## Primeiro Exemplo

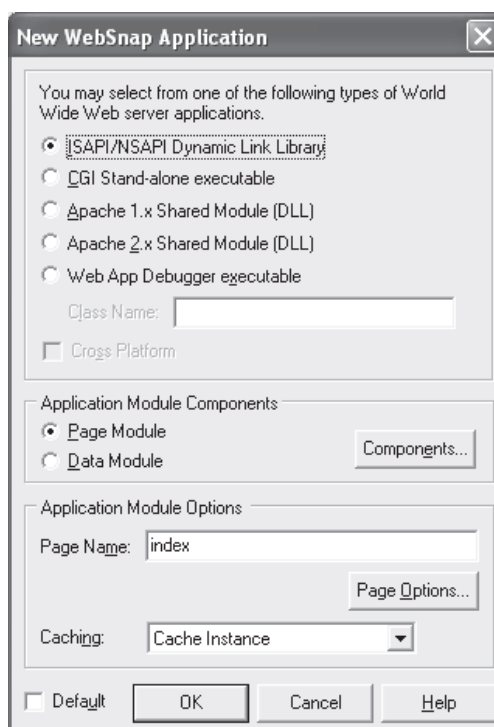
Neste primeiro exemplo, faremos uma simples demonstração da tecnologia *WebSnap*. O mais importante neste tópico é compreender toda a estrutura da tecnologia. Todos os passos serão analisados para que você compreenda de maneira satisfatória cada detalhe.

Através das opções *File/New/Other...*, selecione a seção *WebSnap* e escolha a opções *WebSnap Application* (figura 11.2).



**Figura 11.2** Selecionando o modelo da aplicação

Em seguida teremos uma janela assistente (figura 11.3) com diversas opções de configuração da aplicação.



**Figura 11.3** Opções da aplicação WebSnap

Vamos analisar as opções:

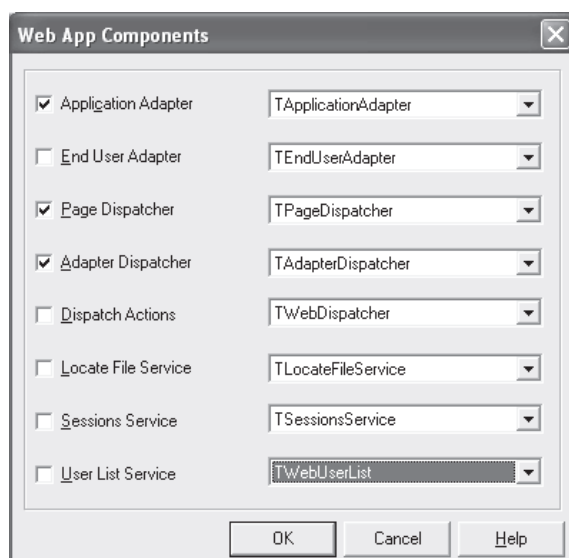
### **Tipo do servidor**

Primeiro, devemos escolher o tipo do servidor (*CGI*, *ISAPI*, *Apache* ou *Web App Debugger*). Escolha a opção *Web App Debugger* e informe o nome da classe (necessário apenas para esta opção). Para este exemplo vamos nomear como *classExemplo1*.

O *Web App Debugger* na realidade é um servidor HTTP que oferece um ótimo nível de depuração para a nossa aplicação. Escolhendo esta opção é gerado um modelo de servidor COM, para interagir com o *Web App Debugger*. Esta opção é utilizada única e exclusivamente para o desenvolvimento e nunca para a distribuição.

### **Componentes da Aplicação**

Em seguida temos a seção *Application Module Components*, responsável pela configuração dos módulos da aplicação. Clicando no botão *Components*, você terá acesso à outra caixa de diálogo (figura 11.4) com opções de serviço.

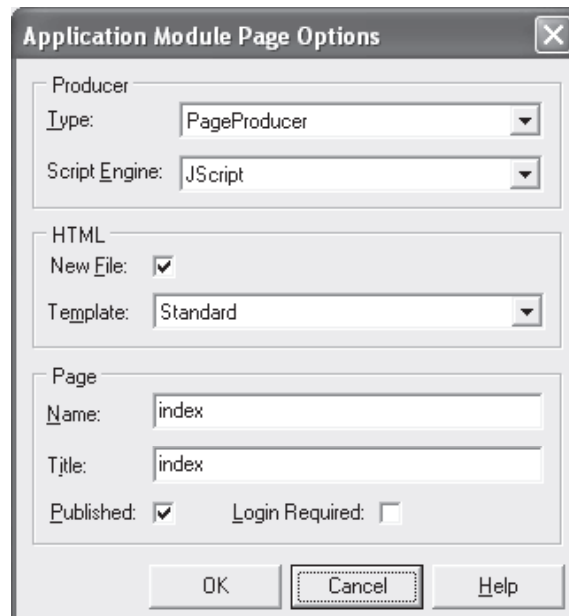


**Figura 11.4** Opções de serviço

Nesta caixa de diálogo escolhemos os serviços básicos de nossa aplicação. Na realidade cada serviço poderá ser adicionado à sua aplicação conforme a necessidade. Para este primeiro exemplo, mantenha as opções padrão e clique no botão OK.

### **Opções da Aplicação**

Em seguida temos a seção *Application Module Options*, responsável pela configuração das opções da aplicação. Clicando no botão **Page Options**, você terá acesso à caixa de diálogo (figura 11.5) com opções da aplicação.



**Figura 11.5** Opções da aplicação

Aqui temos um nível bastante interessante de configuração. Na primeira seção, temos a configuração do nosso *Producer*, com os seguintes tipos:

<i>AdapterPageProducer</i>
<i>DataSetPageProducer</i>
<i>InetXPageProducer</i>
<i>PageProducer</i>
<i>XLSPageProducer</i>

Selecione a opção *PageProducer*. Dependendo do tipo selecionado, temos as seguintes opções para o *script engine*:

<i>JScript</i>
<i>VBScript</i>

Mantenha a seleção em *JScript*.

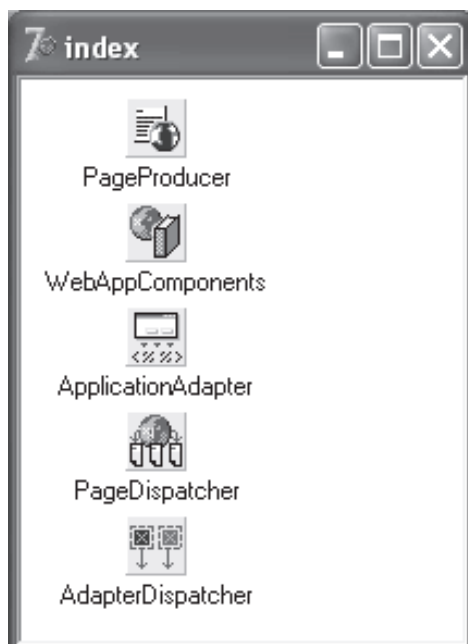
Em seguida temos a seção *HTML*. Esta seção permite ao desenvolvedor criar uma página automaticamente, seja a partir de um modelo (*Standard*) ou em branco (*Blank*). Selecione a opção *Standard* e deixe marcada a opção *New File*.

E, por fim, temos as informações da nossa página HTML.

<i>Name</i>	<i>Utilizado como referência para requisições HTTP e lógica da sua aplicação.</i>
<i>Title</i>	<i>Título que será apresentado na janela do browser.</i>
<i>Published</i>	<i>habilita o recurso de resposta as requisições HTTP.</i>
<i>Login Required</i>	<i>Verifica se existe a necessidade do usuário fazer Login para entrar nesta página.</i>

Mantenha as opções padrão e pressione o botão OK.

Retornando à caixa de diálogo inicial, pressione OK para concluir o assistente e gerar nossa primeira aplicação. A *figura 11.6* ilustra o *PageModule* criado.



**Figura 11.6 Page Module**

Agora vamos gravar nosso primeiro projeto no diretório \WebSnap\Exercicio1, como segue.

Unit PageModule	un_ex1.PAS
Unit Formulário	un_form.PAS
Projeto	wsnap1.DPR

Após a gravação do nosso projeto, repare que foi gerado automaticamente um arquivo **un\_ex1.HTML**, com o seguinte conteúdo.

```
<html>
<head>
<title>
<%= Page.Title %>
</title>
</head>
<body>
<h1><%= Application.Title %></h1>
<% if (EndUser.Logout != null) { %>
<%   if (EndUser.DisplayName != '') { %>
<%     <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   } %>
<%   if (EndUser.Logout.Enabled) { %>
<%     <a href="<%=EndUser.Logout.ASHREF%">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
<%     <a href="<%=EndUser.LoginForm.ASHREF%">Login</a>
<%   } %>
<% } %>

<h2><%= Page.Title %></h2>
<table cellpadding="0" cellspacing="0">
<td>
```

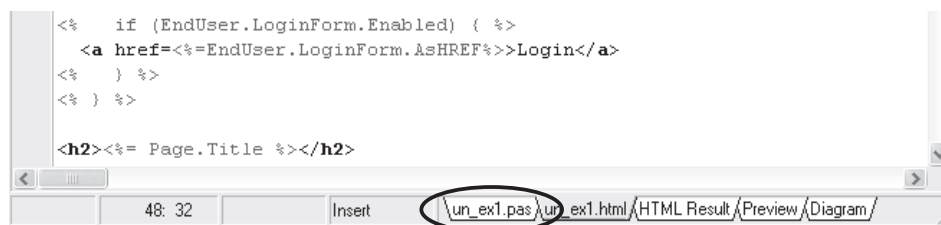
```

<% e = new Enumerator(Pages)
    s = ''
    c = 0
    for (; !e.atEnd(); e.MoveNext())
    {
        if (e.item().Published)
        {
            if (c>0) s += '&nbsp;&nbsp;&nbsp;|&nbsp;&nbsp;&nbsp;';
            if (Page.Name != e.item().Name)
                s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
            else
                s += e.item().Title
            c++
        }
    }
    if (c>1) Response.Write(s)
%>
</td>
</table>

</body>
</html>

```

Podemos visualizar o documento através das *Tabs* do nosso editor de código (figura 11.7).



**Figura 11.7** Tabs do editor de código

Antes de executar nossa primeira aplicação, insira uma frase qualquer no corpo do documento *un\_ex1.HTML*, como segue.

#### Nossa primeira aplicação

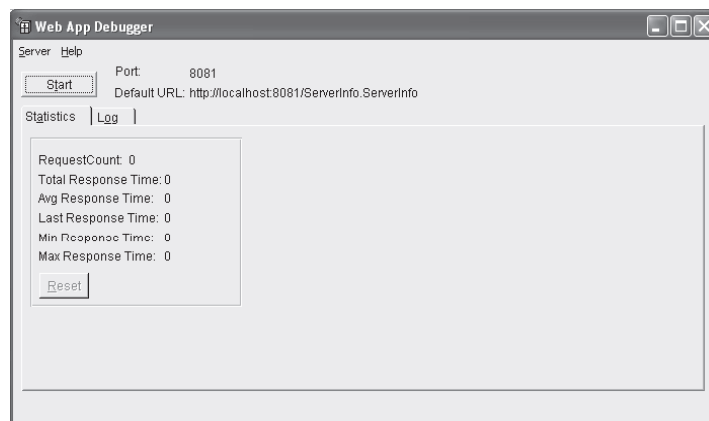
```

</body>
</html>

```

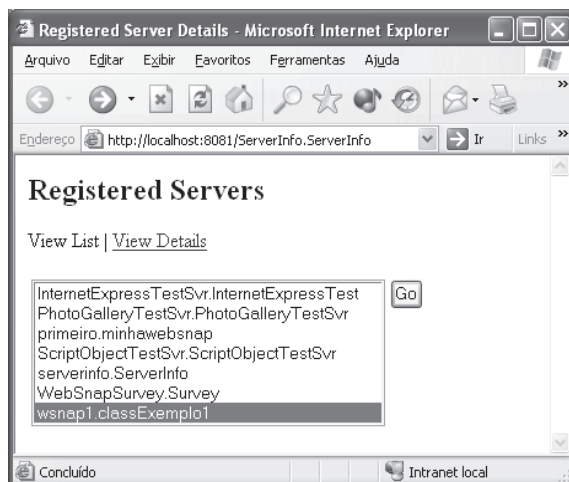
Grave o documento para que possamos executar a aplicação. Execute normalmente a aplicação através da tecla F9, ou da opção Run/Run... do menu. Perceba que o formulário em branco, é apresentado na tela. Através das opções *Tools/WebAppDebugger* vamos executar o nosso servidor e depurador de aplicações. A figura 11.8 ilustra o *WebAppDebugger* em execução. Clique no botão *Start* para iniciar o servidor e, em seguida, no link *Default URL*.





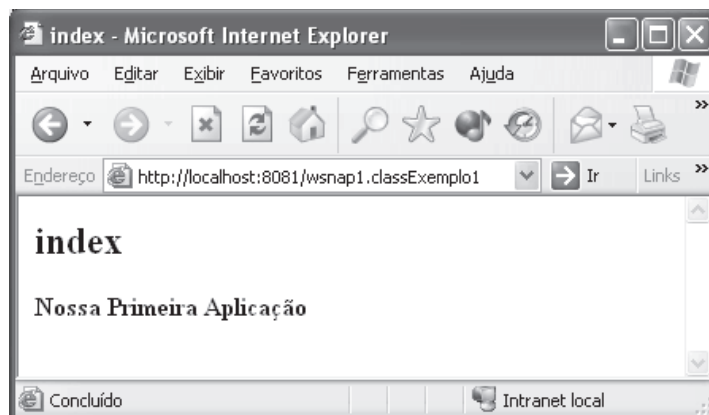
**Figura 11.8 Web App Debugger**

Ao clicar no link *Default*, será apresentado um documento HTML (figura 11.9) com todos os serviços registrados. Selecione o nosso exemplo e clique no botão *GO*.



**Figura 11.9 Serviços registrados**

A figura 11.10 ilustra o resultado da nossa primeira aplicação *WebSnap*.



**Figura 11.10 Primeira aplicação WebSnap**

Com isso concluímos nossa primeira aplicação.

**Listagem 11.1 un\_ex1.pas**

```

unit un_ex1;

interface

uses
  SysUtils, Classes, HTTPApp, WebModu, HTTPProd, ReqMulti, WebDisp,
  WebAdapt, WebComp;

type
  Tindex = class(TWebAppPageModule)
    PageProducer: TPageProducer;
    WebAppComponents: TWebAppComponents;
    ApplicationAdapter: TApplicationAdapter;
    PageDispatcher: TPageDispatcher;
    AdapterDispatcher: TAdapterDispatcher;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  function index: Tindex;

implementation

{$R *.dfm}  {$*.html}

uses WebReq, WebCntxt, WebFact, Variants;

function index: Tindex;
begin
  Result := Tindex(WebContext.FindModuleClass(Tindex));
end;

initialization
  if WebRequestHandler <> nil then
    WebRequestHandler.AddWebModuleFactory(TWebAppPageModuleFactory.Create(Tindex,
    TWebPageInfo.Create([wpPublished {, wpLoginRequired}], '.html'), caCache)
  );
end.

```

**Segundo Exemplo**

No segundo exemplo vamos conhecer os conceitos de *Server Scripts* e dos *Adapters*.

Os *Server Scripts* nos auxiliam em diversas tarefas, seja para acessar valores em objetos criados através do Delphi, ou até mesmo rotinas dinâmicas geradas no servidor. A grande vantagem de utilizar *Server Scripts* é justamente a facilidade da manutenção do projeto, onde muitas vezes, apenas o arquivo HTML sofre mudanças. Com isso não há necessidade de uma nova compilação.

Já os *Adapters*, como você irá perceber ao longo deste capítulo, serão nossos grandes aliados. Na realidade os *Adapters* fornecem uma *interface de scripts* para a sua aplicação servidora. Vamos aprender na prática esses conceitos.

Através das opções *File/New/Other...*, selecione a seção *WebSnap* e escolha a opções *WebSnap Application* (figura 11.11).

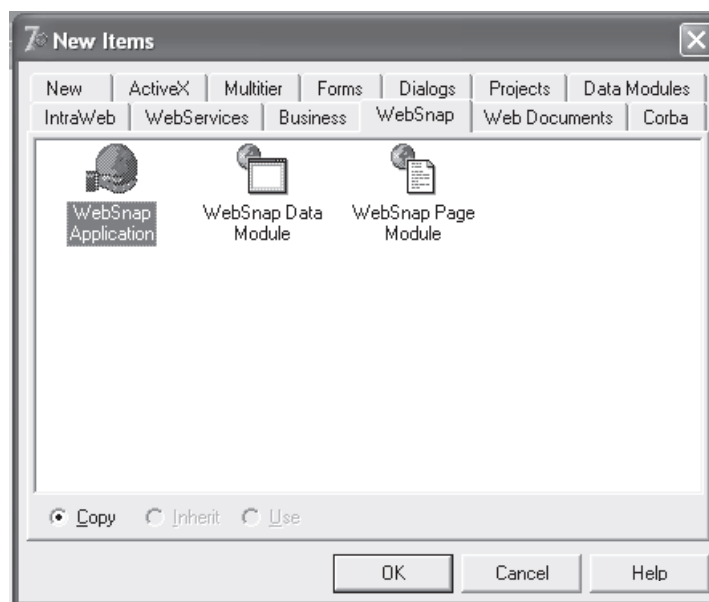


Figura 11.11 Selecionando o modelo da aplicação

Em seguida (figura 11.12) selecione a opção *Web App Debugger*, e informe o nome da classe (necessário apenas para esta opção). Para este exemplo vamos nomear como *classExemplo2*.

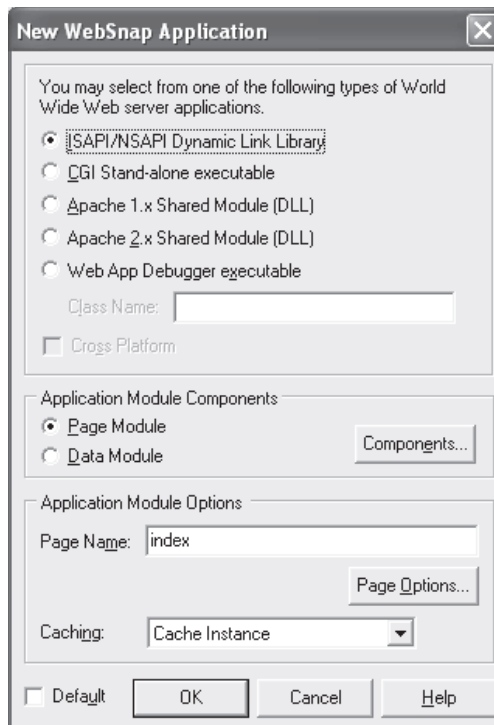


Figura 11.12 Opções da aplicação WebSnap

Em seguida temos a seção *Application Module Components*, responsável pela configuração dos módulos da aplicação. Clicando no botão *Components*, você terá acesso à outra caixa de diálogo (figura 11.13) com opções de serviço.

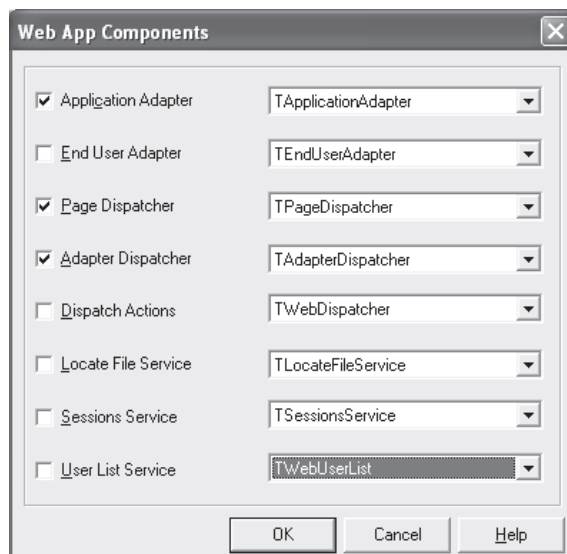


Figura 11.13 Opções de serviço

Assim como no primeiro exemplo, mantenha as opções padrão e clique no botão OK. Em seguida temos a seção *Application Module Options*, responsável pela configuração das opções da aplicação. Clicando no botão *Page Options*, você terá acesso à caixa de diálogo (figura 11.14) com opções da aplicação.

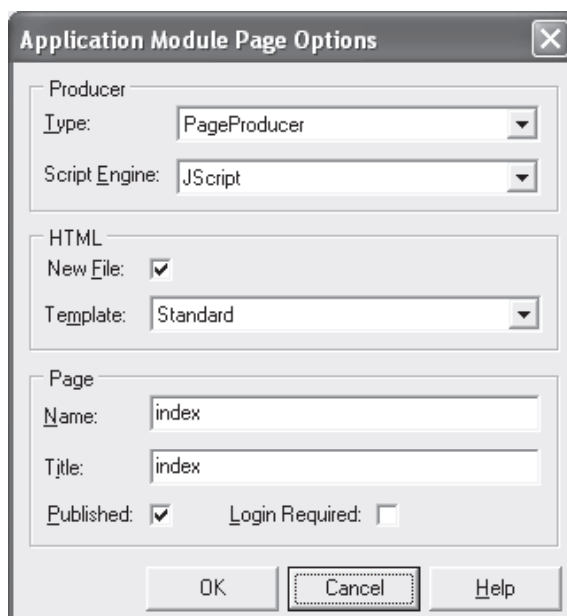
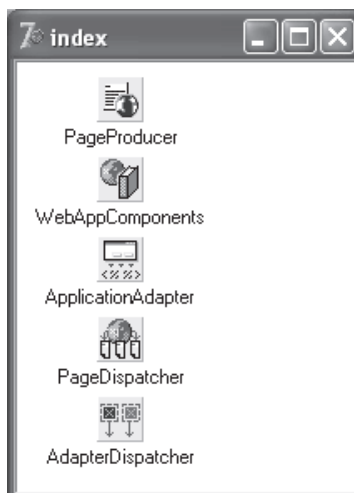


Figura 11.14 Opções da aplicação

Selecione *Page Producer* para o *Producer Type* e mantenha a seleção *JScript* no *Script Engine*. Em seguida temos a seção *HTML*. Selecione a opção *Standard* e deixe marcada a opção *New File*. Retornando à caixa de diálogo inicial, pressione OK para concluir o assistente e gerar nossa segunda aplicação. A figura 11.15 ilustra o *PageModule* criado.

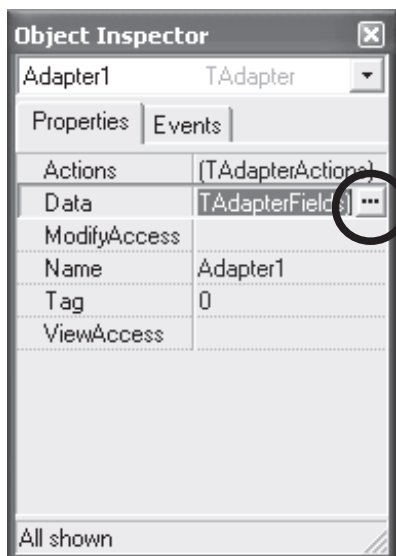


**Figura 11.15 Page Module**

Agora vamos gravar nosso segundo projeto no diretório \WebSnap\Exercicio2, como segue.

Unit PageModule	un_ex2.PAS
Unit Formulário	un_form.PAS
Projeto	wsnap2.DPR

Agora vamos inserir um objeto do tipo *TAdapter* e, em seguida, acessar a *propriedade Data* deste objeto, clicando no botão relacionado (figura 11.16).



**Figura 11.16 Propriedade Data**

Em seguida temos o editor de campos para o objeto *Adapter* (figura 11.17).



Figura 11.17 Editor de campos

Clique com o botão direito do mouse sob o editor e selecione a opção *New Component* ou clique no primeiro botão, como ilustra a *figura 11.17*. Em seguida teremos uma lista com os possíveis componentes do nosso objeto *Adapter1*.

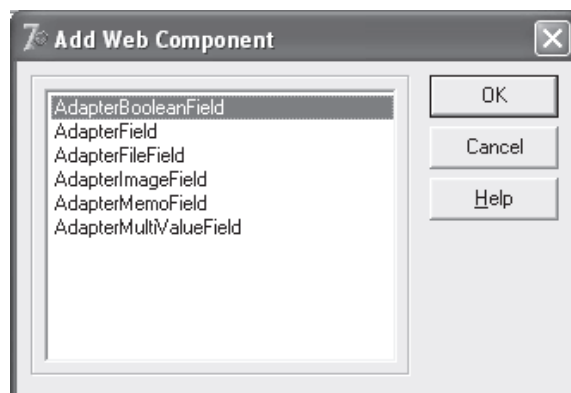


Figura 11.18 Componentes do Adapter

Selecione o componente *AdapterField* e clique no botão OK. Modifique a propriedade *Name* do componente criado para *MeuFieldAdapter*. A *figura 11.19* ilustra os eventos do nosso objeto *MeuFieldAdapter*.

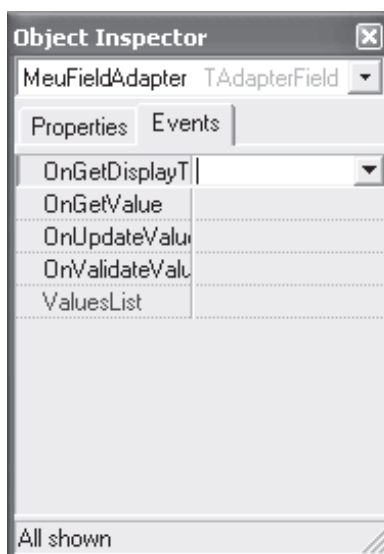


Figura 11.19 Eventos do MeuFieldAdapter

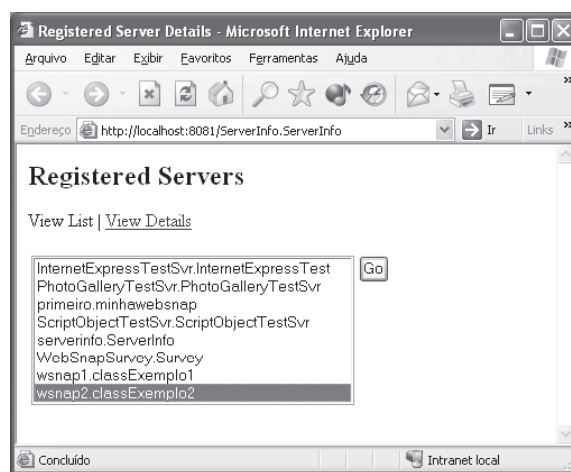
Prosseguindo com nosso exemplo, vamos atribuir um valor dinâmico ao nosso objeto *MeuFieldAdapter*. No evento *OnGetValue* insira o código que segue:

```
procedure Tindex.MeuFieldAdapterGetValue(Sender: TObject;
  var Value: Variant);
begin
  Value:=TimetoStr(Time);
end;
```

Agora vamos acessar o valor do nosso objeto, no *arquivo index.HTML* criado pelo assistente. Insira o comando que segue, na seção <BODY> do documento.

```
<B><%Response.Write(Adapter1.MeuFieldAdapter.value)%></B>
```

Grave o nosso projeto e execute normalmente a aplicação através da tecla F9, ou da opção Run/Run... do menu. Através das opções *Tools/WebAppDebugger* vamos executar o nosso servidor e depurador de aplicações. Clique no botão *Start* para iniciar o servidor, e em seguida no link *Default URL*. Ao clicar no link *Default* será apresentado um documento HTML (*figura 11.20*) com todos os serviços registrados. Selecione o nosso exemplo e clique no botão *GO*.



**Figura 11.20 Serviços registrados**

Amigos, obviamente o exemplo é bastante simples, mas imaginem que através dos objetos *Adapters* poderemos acessar valores de cálculos, resultados de instruções SQL, imagens dinâmicas, entre outras funcionalidades.

### Listagem 11.2 un\_ex2.pas

```
unit un_ex2;

interface

uses
  SysUtils, Classes, HTTPApp, WebModu, HTTPProd, ReqMulti, WebDisp,
  WebAdapt, WebComp;

type
  Tindex = class(TWebAppPageModule)
    PageProducer: TPageProducer;
    WebAppComponents: TWebAppComponents;
    ApplicationAdapter: TApplicationAdapter;
    PageDispatcher: TPageDispatcher;
    AdapterDispatcher: TAdapterDispatcher;
    Adapter1: TAdapter;
```

```

    MeuFieldAdapter: TAdapterField;
    procedure MeuFieldAdapterGetValue(Sender: TObject; var Value: Variant);
private
    { Private declarations }
public
    { Public declarations }
end;

function index: Tindex;

implementation

{$R *.dfm}    {*.html}

uses WebReq, WebCntxt, WebFact, Variants;

function index: Tindex;
begin
    Result := Tindex(WebContext.FindModuleClass(Tindex));
end;

procedure Tindex.MeuFieldAdapterGetValue(Sender: TObject;
    var Value: Variant);
begin
    Value:=TimetoStr(Time);
end;

initialization
    if WebRequestHandler <> nil then
        WebRequestHandler.AddWebModuleFactory(TWebAppPageModuleFactory.Create(Tindex,
TWebPageInfo.Create([wpPublished {, wpLoginRequired}], '.html'), caCache)
);

end.

```

### Listagem 11.3 un\_ex2.HTML

```

<html>
<head>
<title>
<%= Page.Title %>
</title>
</head>
<body>
<h1><%= Application.Title %></h1>
<% if (EndUser.Logout != null) { %>
<%   if (EndUser.DisplayName != '') { %>
    <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   } %>
<%   if (EndUser.Logout.Enabled) { %>
    <a href="<%=EndUser.Logout.ASHREF%">">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
    <a href="<%=EndUser.LoginForm.ASHREF%">">Login</a>
<%   } %>
<% } %>

```



```

<h2><%= Page.Title %></h2>
<table cellpadding="0" cellspacing="0">
<td>
<% e = new Enumerator(Pages)
    s = ''
    c = 0
    for (; !e.atEnd(); e.MoveNext())
    {
        if (e.item().Published)
        {
            if (c>0) s += '&nbsp;&nbsp;&nbsp;|&nbsp;&nbsp;&nbsp;';
            if (Page.Name != e.item().Name)
                s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
            else
                s += e.item().Title
            c++
        }
    }
    if (c>1) Response.Write(s)
%>
</td>
</table>

<B><% Response.Write(Adapter1.MeuFieldAdapter.value) %></B>

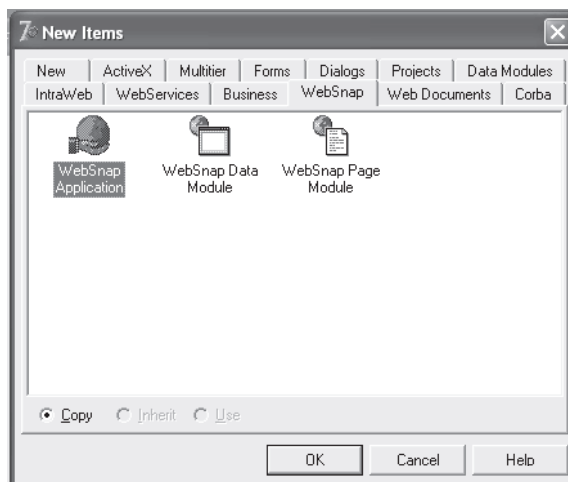
</body>
</html>

```

## Terceiro Exemplo

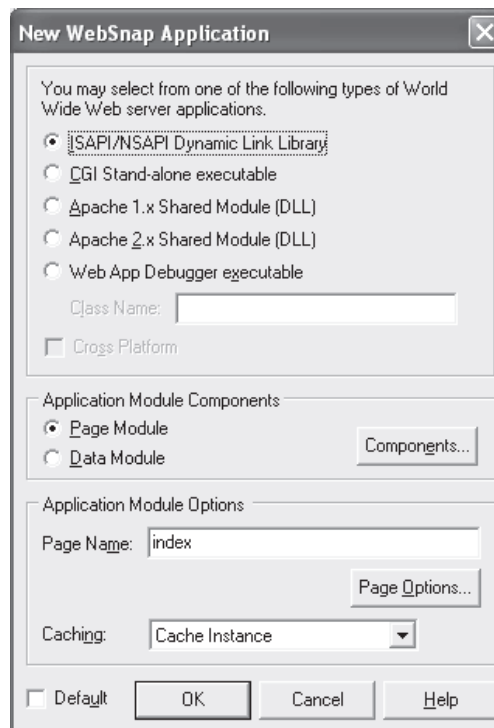
No terceiro exemplo vamos trabalhar com múltiplos *Page Modules*, além de dispensar o assistente para criação automática de páginas.

Através das opções *File/New/Other...*, selecione a seção *WebSnap* e escolha a opções *WebSnap Application* (figura 11.21).



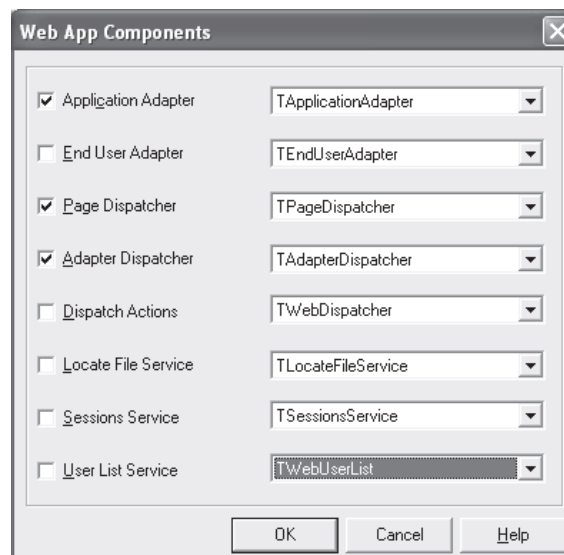
**Figura 11.21** Selecionando o modelo da aplicação

Em seguida, (figura 11.22) selecione a opção *Web App Debugger* e informe o nome da classe. Para este exemplo vamos nomear como *classExemplo3*.



**Figura 11.22** Opções da aplicação WebSnap

Em seguida temos a seção *Application Module Components*, responsável pela configuração dos módulos da aplicação. Clicando no botão *Components*, você terá acesso à outra caixa de diálogo (figura 11.23) com opções de serviço.

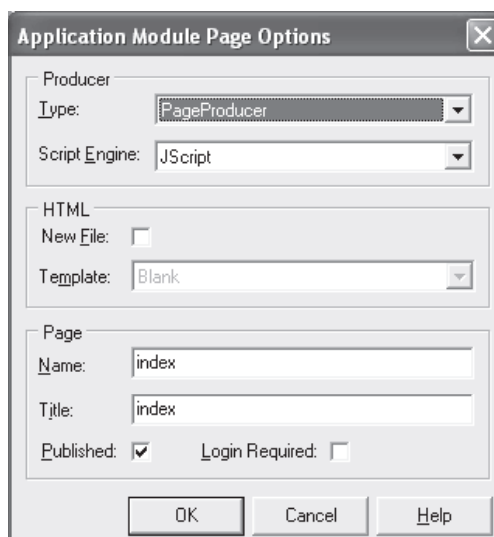


**Figura 11.23** Opções de serviço

Assim como nos exemplos anteriores, mantenha as opções padrão e clique no botão OK.

Em seguida temos a seção *Application Module Options*, responsável pela configuração das opções da aplicação. Clicando no botão *Page Options*, você terá acesso à caixa de diálogo (figura 11.24) com opções da aplicação. Selecione *Page Producer* para o *Producer Type* e mantenha a seleção *JScript* no *Script Engine*.

Em seguida temos a seção *HTML*. Desmarque a opção *New File* e clique no botão OK.

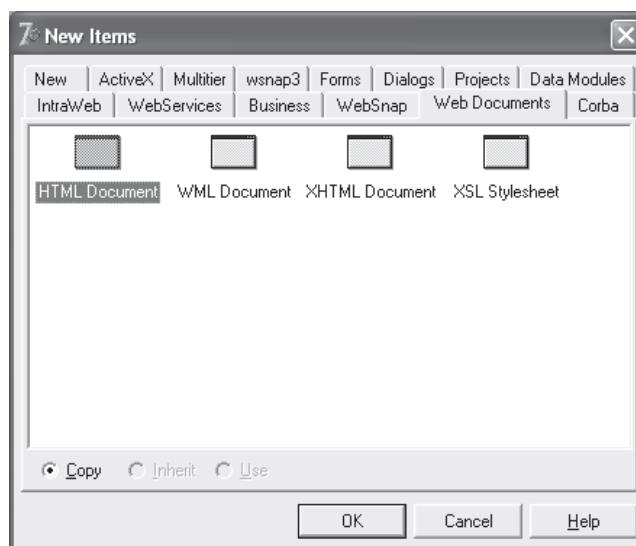


**Figura 11.24** Opções da aplicação

Agora vamos gravar nosso terceiro projeto no diretório \WebSnap\Exercicio3, como segue.

Unit PageModule	un_ex3.PAS
Unit Formulário	un_form.PAS
Projeto	wsnap3.DPR

Como dispensamos a criação automática do arquivo HTML, vamos utilizar os novos assistentes do Delphi, para criá-lo. Através das opções *File/New/Other...*, acesse a seção *Web Documents* como ilustra a figura 11.25 e selecione a opção *HTML*.



**Figura 11.25** Novo documento HTML

Veja que o seguinte documento foi criado.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE> Untitled1.html </TITLE>
  </HEAD>
  <BODY>

  </BODY>
</HTML>
```

No editor do Delphi, perceba que existe um *assistente code completion* para HTML (figura 11.26).

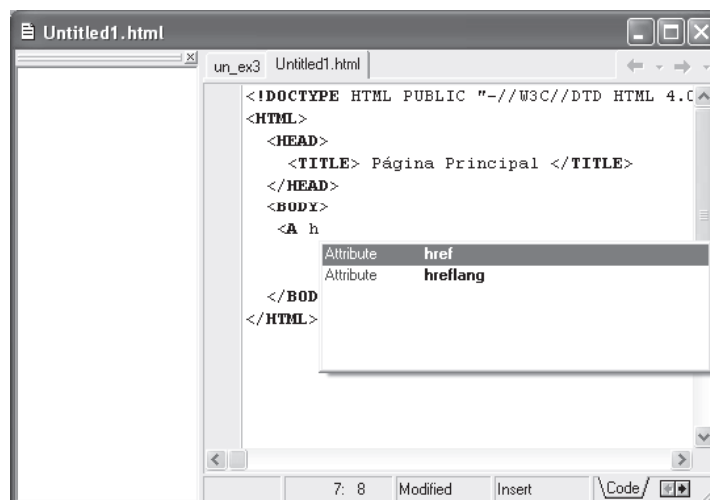


Figura 11.26 Code completion HTML

Faça as modificações (em negrito) como segue:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE> Página Principal </TITLE>
  </HEAD>
  <BODY>

  <% e = new Enumerator(Pages)
    s = ''
    c = 0
    for (; !e.atEnd(); e.moveNext())
    {
      if (e.item().Published)
      {
        if (c>0) s += '&nbsp;|&nbsp;';
        if (Page.Name != e.item().Name)
          s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
        else
          s += e.item().Title
        c++
      }
    }
  %>
```

```

    }
  }
  if (c>1) Response.Write(s)
%>
</BODY>
</HTML>

```

Vamos analisar o código *script*, normalmente utilizado pelo padrão *Standard* do *Page Module*:

No primeiro bloco, estamos criando um objeto do tipo *Enumerator* e iniciando as variáveis *s* e *c*. A variável *s* será a nossa *string* para montagem dos links que serão apresentados e a variável *c* uma auxiliar para contagem de números de páginas da aplicação.

```

<% e = new Enumerator(Pages)
    s = ''
    c = 0

```

No bloco seguinte estamos iniciando um laço baseado no objeto *e* criado anteriormente e movendo uma posição no objeto a cada ciclo. Perceba que o objeto *e* cria uma matriz de *Page Modules* da aplicação.

```

for (; !e.atEnd(); e.moveNext())

```

No bloco que segue, estamos verificando se o item atual (*Page Module*) do objeto *e* possui o método de publicação.

```

if (e.item().Published)

```

Em caso afirmativo, acrescenta-se à variável *s*, “*espaço | espaço*”, onde *&nbsp;* é igual a espaço na linguagem HTML.

```

if (c>0) s += '&nbsp;|&nbsp;';

```

Seguindo com o nosso *script*, este próximo bloco verifica se o *item* em questão é diferente do que solicita o *script*, ou seja, verifica se é diferente dele mesmo. Em caso afirmativo cria o link da página baseado no item atual, senão apenas apresenta o título da página.

```

if (Page.Name != e.item().Name)
  s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
else
  s += e.item().Title

```

Em seguida apenas incrementamos a variável *c*.

```

c++

```

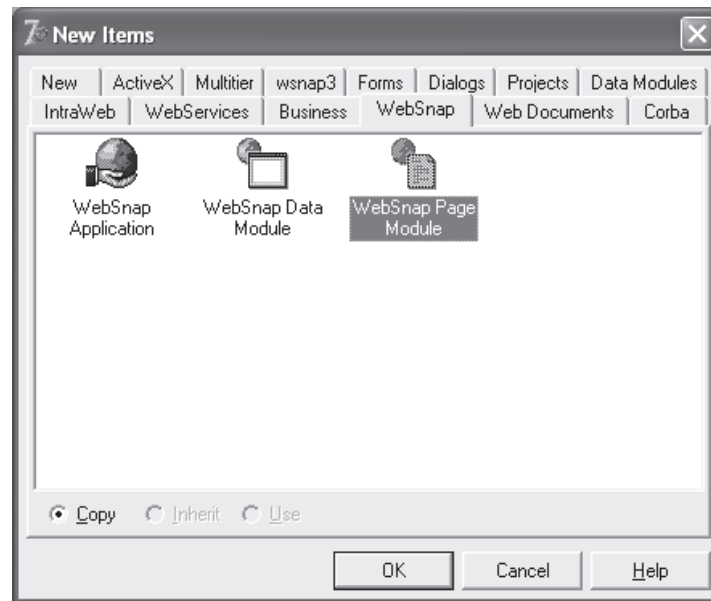
E por fim, escrevemos no HTML todo o conteúdo da variável *s* caso a variável *c* seja maior que *1*, isto é, apenas se nossa aplicação contiver 2 ou mais *PageModules*.

```

if (c>1) Response.Write(s)
%>

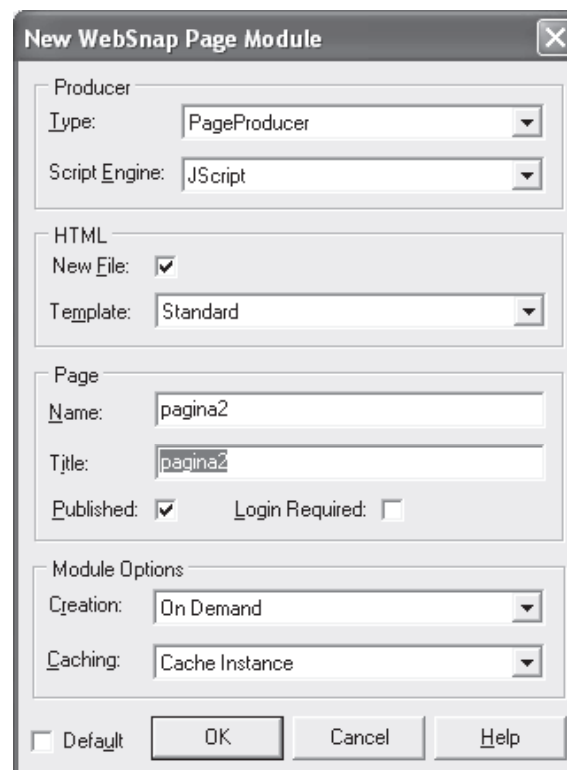
```

Agora vamos adicionar *Page Modules* ao nosso projeto. Através das opções *File/New..Other...*, seção *WebSnap*, selecione o assistente *WebSnap PageModule* (figura 11.27).



**Figura 11.27 Novo PageModule**

Em seguida, configure as opções como ilustra a *figura 11.28*.

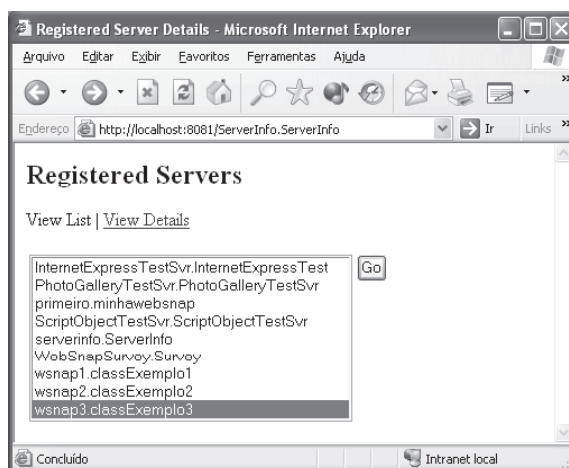


**Figura 11.28 Configuração Page Module**

Grave a *unit* como *un\_ex3\_pag2.pas*. Insira o código que segue no HTML associado a *unit un\_ex3\_pag2*, em nosso caso *un\_ex3\_pag2.HTML*.

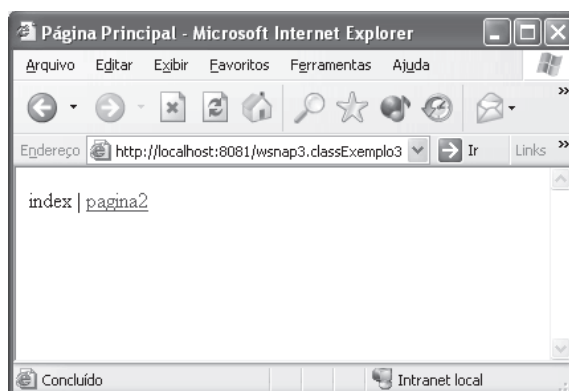
```
<B> Estamos na Página 2 </B>
```

Grave o nosso projeto e execute normalmente a aplicação através da tecla *F9*, ou da opção *Run/Run...* do menu. Através das opções *Tools/WebAppDebugger* vamos executar o nosso servidor e depurador de aplicações. Clique no botão *Start* para iniciar o servidor, e em seguida no link *Default URL*. Ao clicar no link *Default* será apresentado um documento HTML (figura 11.29) com todos os serviços registrados. Selecione o nosso exemplo e clique no botão *GO*.

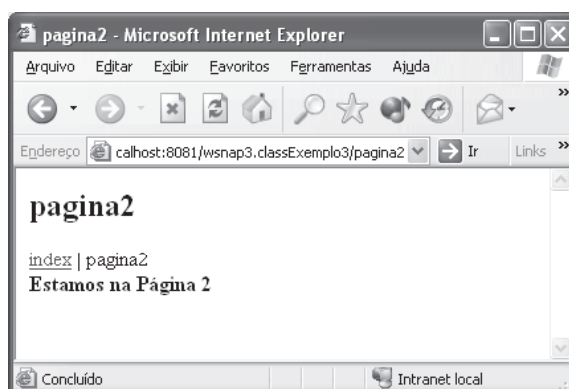


**Figura 11.29** Aplicações registradas

As figuras 11.30 e 11.31 ilustram o resultado de nossa aplicação.



**Figura 11.30** Terceiro exemplo página principal



**Figura 11.31** Terceiro exemplo página 2

### Listagem 11.4 principal.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE> Página Principal </TITLE>
  </HEAD>
  <BODY>
    <% e = new Enumerator(Pages)
      s = ''
      c = 0
      for (; !e.atEnd(); e.moveNext())
      {
        if (e.item().Published)
        {
          if (c>0) s += '&nbsp;|&nbsp;';
          if (Page.Name != e.item().Name)
            s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
          else
            s += e.item().Title
          c++
        }
      }
      if (c>1) Response.Write(s)
    %>

  </BODY>
</HTML>
```

## Quarto Exemplo

No quarto exemplo vamos trabalhar com a interatividade do usuário.

Através das opções *File/New/Other...*, selecione a seção *WebSnap* e escolha a opções *WebSnap Application* (figura 11.32).

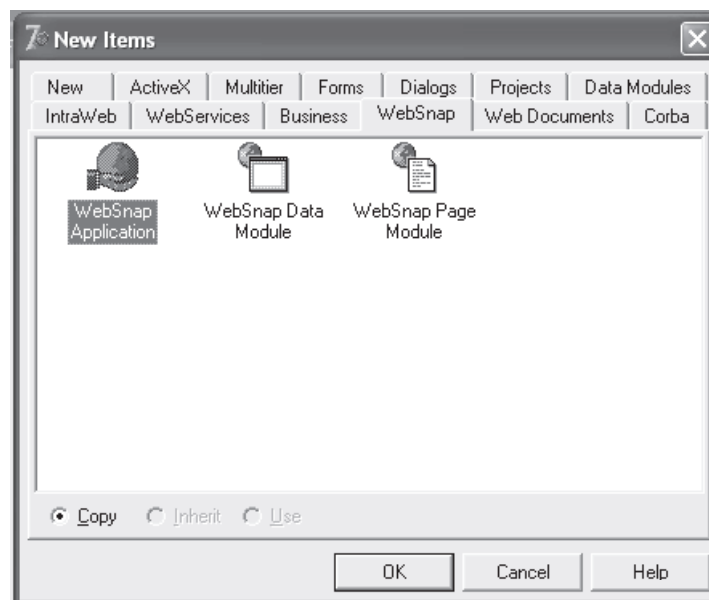


Figura 11.32 Selecionando o modelo da aplicação



Em seguida (figura 11.33) selecione a opção *Web App Debugger* e informe o nome da classe. Para este exemplo vamos nomear como *classExemplo4*.

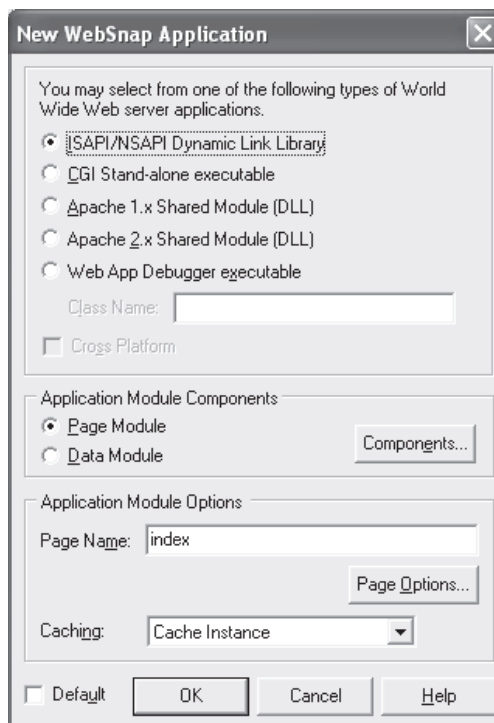


Figura 11.33 Opções da aplicação WebSnap

Em seguida temos a seção *Application Module Components*, responsável pela configuração dos módulos da aplicação. Clicando no botão *Components*, você terá acesso à outra caixa de diálogo (figura 11.34) com opções de serviço.

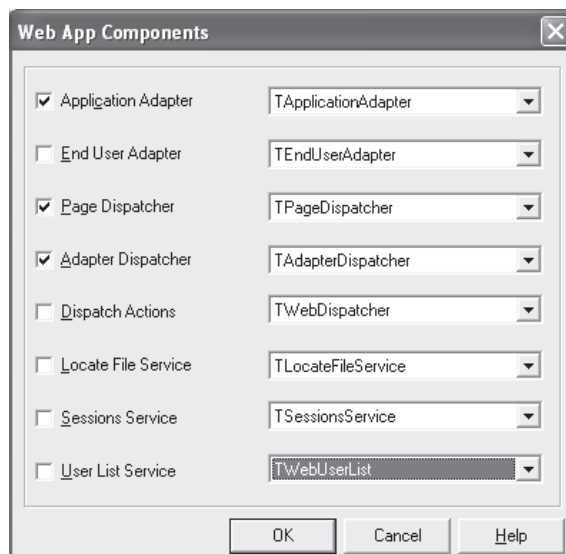
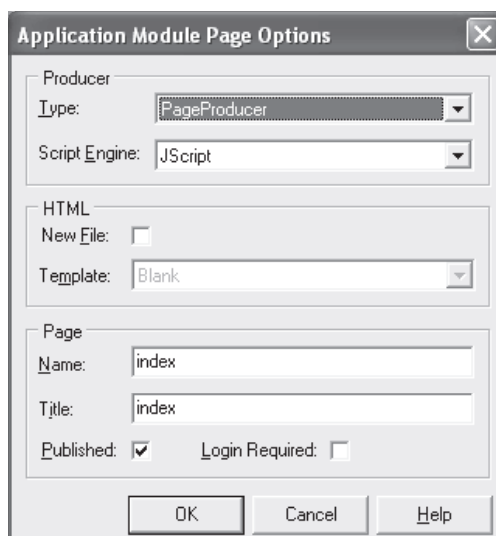


Figura 11.34 Opções de serviço

Assim como nos exemplos anteriores, mantenha as opções padrão, e clique no botão OK.

Em seguida temos a seção *Application Module Options*, responsável pela configuração das opções da aplicação. Clicando no botão *Page Options*, você terá acesso à caixa de diálogo (figura 11.35) com opções da aplicação. Selecione *Page Producer* para o *Producer Type* e mantenha a seleção *JScript* no *Script Engine*. Em seguida temos a seção *HTML*. Desmarque a opção *New File* e clique no botão OK.

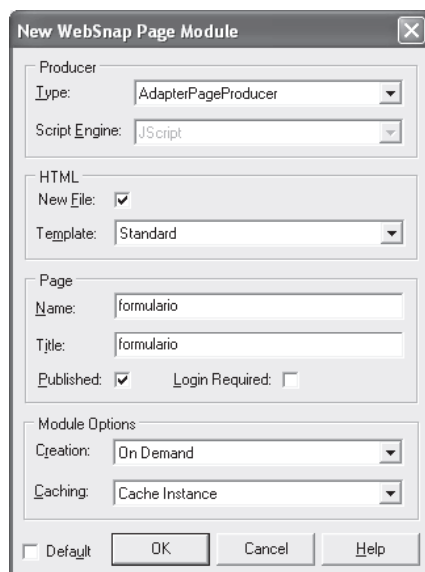


**Figura 11.35** Opções da aplicação

Agora vamos gravar nosso quarto projeto no diretório `\WebSnap\Exercicio4`, como segue.

Unit PageModule	un_ex4.PAS
Unit Formulário	un_form.PAS
Projeto	wsnap4.DPR

Agora vamos criar um novo *Web Page Module* através das opções *File/New...Other...* e na seção *WebSnap*, selecione *WebSnap Page Module*. Configure de acordo com a figura 11.36.

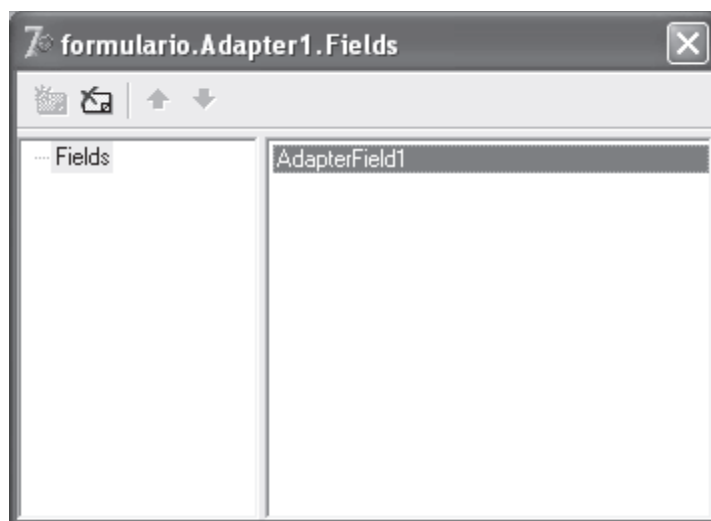


**Figura 11.36** Novo Web Page Module

Pressione OK, e grave a *unit* como **un\_formulario.pás**. Repare que selecionamos *AdapterPageProducer*, ao invés do tradicional *Page Producer*. Vamos criar três variáveis na seção **private**, com o objetivo de executar um cálculo futuro.

```
private
  v1:single;
  v2:single;
  vResultado:single;
```

Agora insira um objeto do tipo *TAdapter* para que possamos trabalhar com objetos *Fields*. Inclua um *Adapter Field*, através da propriedade *Data* do objeto *Adapter1*, como ilustra a *figura 11.37*.



**Figura 11.37 Adapter Field**

Altere a propriedade *Name* para *Valor1* e insira o código que segue no evento *OnGetValue*:

```
procedure TFormulario.Valor1GetValue(Sender: TObject; var Value: Variant);
begin
  Value:=v1;
end;
```

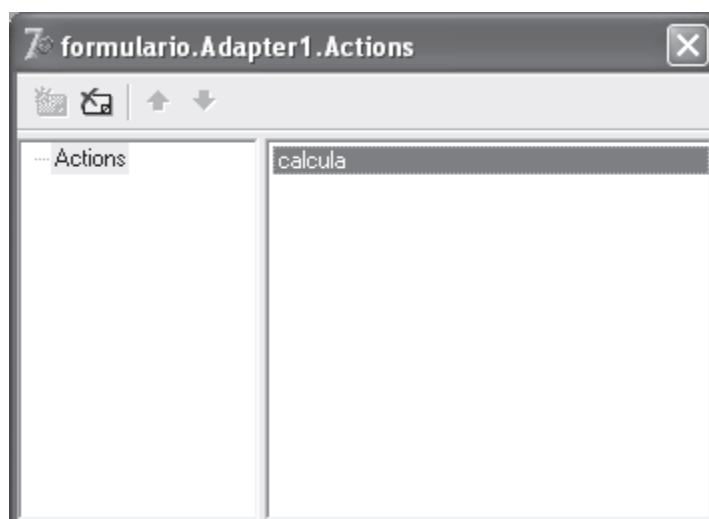
Agora vamos criar outro *AdapterField* com o nome *Valor2* e insira o seguinte código no evento *OnGetValue*:

```
procedure TFormulario.Valor2GetValue(Sender: TObject; var Value: Variant);
begin
  Value:=v2;
end;
```

Agora vamos incluir um *AdapterField* para exibir o resultado da operação, com o nome *Resultado*. Insira o código que segue no evento *OnGetValue*:

```
Value:= vResultado;
```

Antes de definir o *layout* da aplicação vamos definir a ação que irá alimentar nossas variáveis. Selecione o objeto *Adapter1* e através da propriedade *Actions* (*figura 11.38*) crie uma nova *Action* para o nosso objeto. Altere o nome da *Action* para *calcula*.



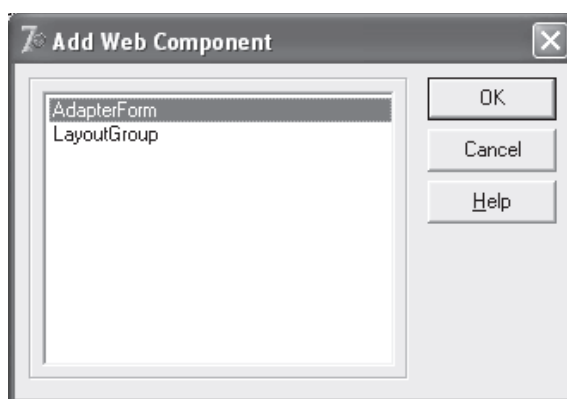
**Figura 11.38 Actions**

No evento *OnExecute* da Action *calcula*, insira o código a seguir.

```
procedure TFormulario.calculaExecute(Sender: TObject; Params: TStrings);
begin
    v1:= Valor1.ActionValue.Values[0];
    v2:= Valor2.ActionValue.Values[0];
    vResultado:=v1+v2;
end;
```

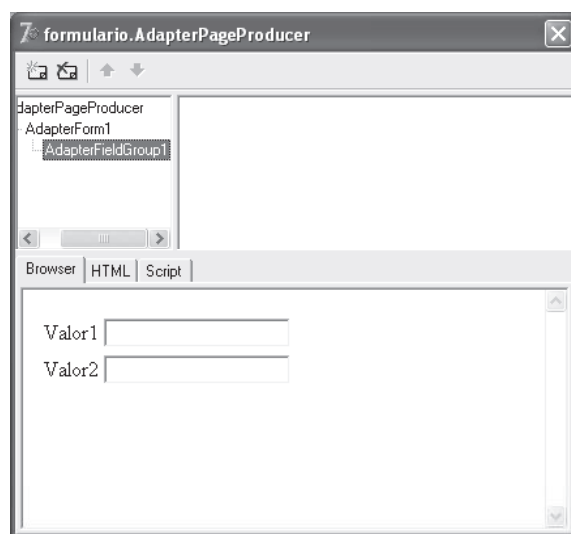
O código é bastante simples, onde estamos atribuindo às variáveis *v1*, *v2* e *vResultado*, os valores resgatados através de um formulário HTML que iremos criar. Na realidade, a variável *vResultado* está sendo “alimentada” com o resultado da operação (*v1+v2*).

Agora vamos criar o formulário HTML utilizando o nosso *AdapterPageProducer*. Através do duplo-clique no objeto *AdapterPageProducer1*, adicione um novo componente do tipo *AdapterForm*



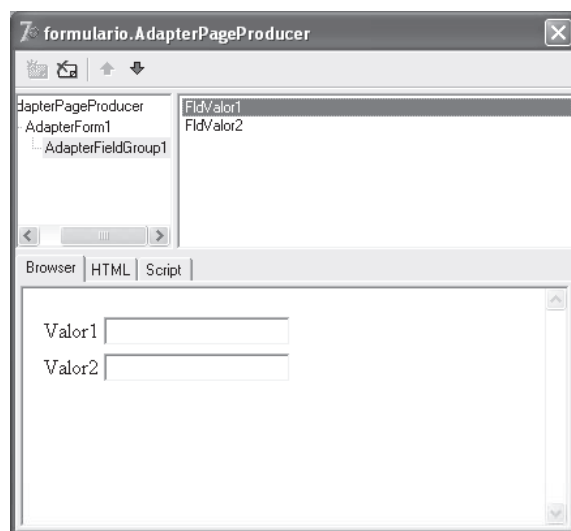
**Figura 11.39 Web Component**

Agora clique no *AdapterForm1* e insira um objeto do tipo *AdapterFieldGroup*. Devemos vincular o *AdapterFieldGroup* ao objeto *Adapter1*, através da propriedade *Adapter*. A figura 11.40 ilustra nosso formulário após o vínculo.



**Figura 11.40** Formulário criado através do Adapter

Em nosso caso desejamos configurar manualmente as propriedades de cada campo. Para realizar tal operação, devemos adicionar os *Fields*, clicando com o botão direito do mouse em *AdapterFieldsGroup* e selecionar a opção *Add All Fields*. A figura 11.41 ilustra nossa operação.



**Figura 11.41** Fields

Altere as propriedades dos objetos criados como segue:

Objeto FldValor1	
Propriedade	Valor
Caption	Informe o primeiro valor

Objeto FldValor2	
Propriedade	Valor
Caption	Informe o segundo valor

Agora vamos inserir o botão com a nossa ação.

Selecione o *AdapterForm1* e insira um novo componente do tipo *AdapterCommandGroup*. No objeto criado, faça o vínculo da propriedade *DisplayComponent* com o objeto *AdapterFieldGroup1*. Na realidade estamos colocando o nosso botão dentro do *AdapterFieldGroup1*. A figura 11.42 ilustra nosso formulário atual.

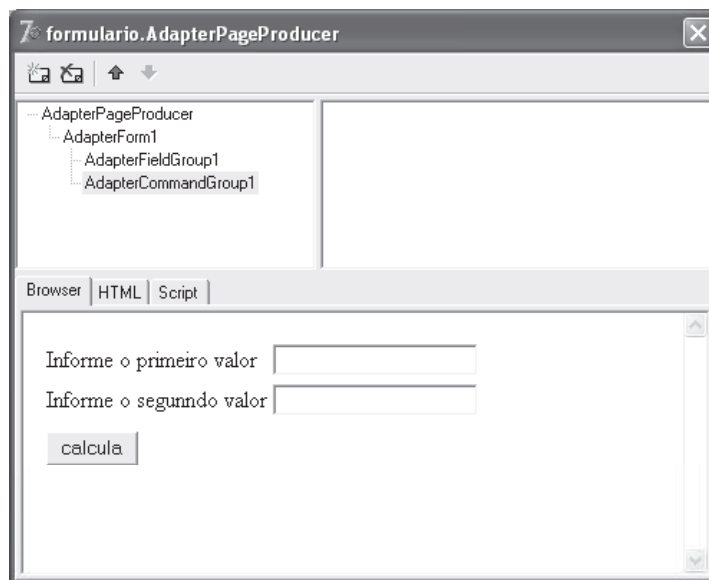


Figura 11.42 Formulário

Agora vamos inserir o campo *Resultado*. Selecione o objeto *AdapterForm1* e insira um novo componente do tipo *AdapterFieldGroup*. Será criado o *AdapterFieldGroup2*. Faça o vínculo com o objeto *Adapter1*, através da propriedade *Adapter*. Ainda no *AdapterFieldGroup2*, adicione o field *fldResultado* através da opção *Add Field*. Altere a propriedade *ViewMode* do field adicionado para **vwDisplay**. A figura 11.43 ilustra o resultado da nossa operação.

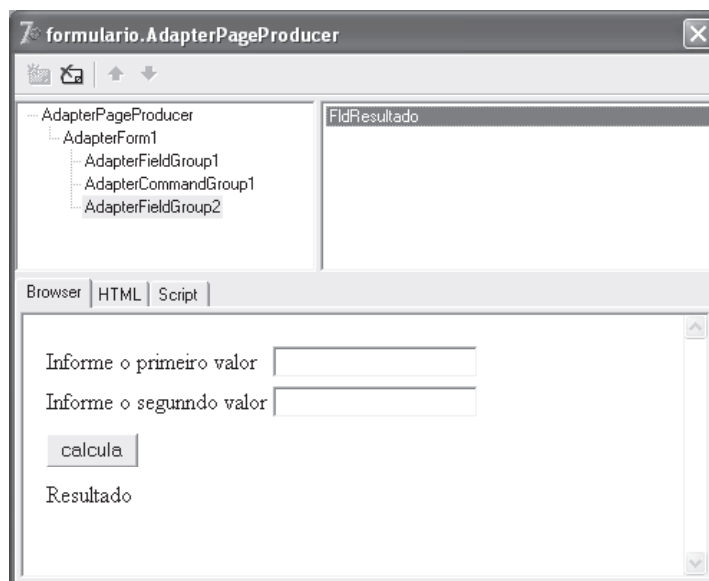
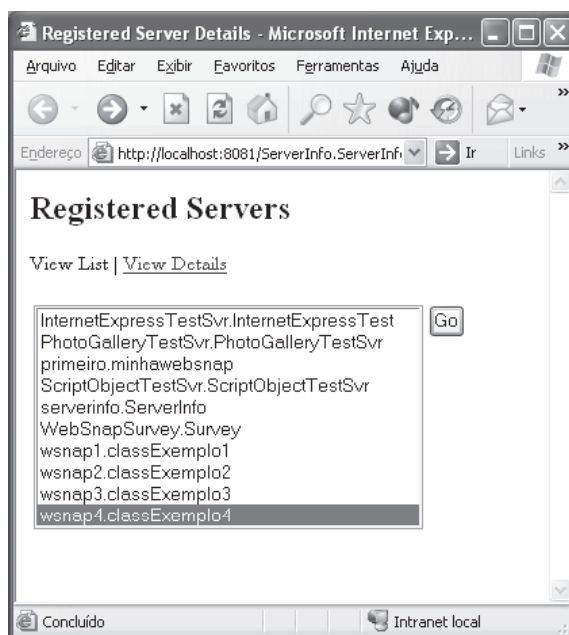


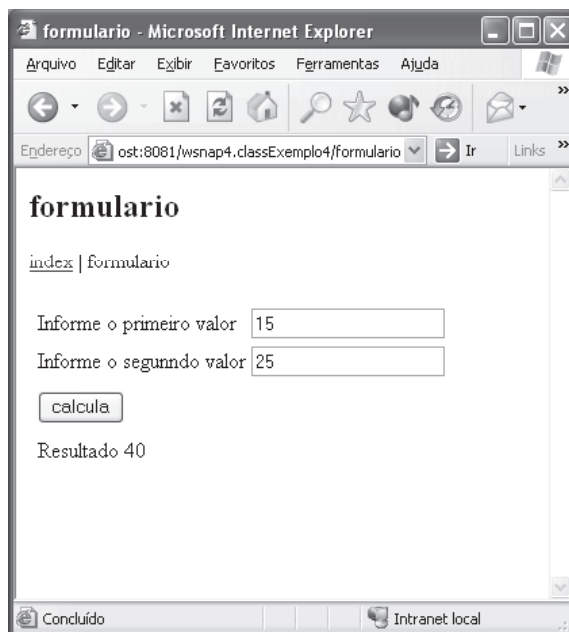
Figura 11.43 Campo resultado

Grave o nosso projeto e execute normalmente a aplicação através da tecla *F9*, ou da opção *Run/Run...* do menu. Através das opções *Tools/WebAppDebugger* vamos executar o nosso servidor e depurador de aplicações. Clique no botão *Start* para iniciar o servidor e, em seguida, no link *Default URL*. Ao clicar no link *Default* será apresentado um documento HTML (figura 11.44) com todos os serviços registrados. Selecione o nosso exemplo e clique no botão *GO*.



**Figura 11.44** Serviços registrados

A figura 11.45 ilustra o resultado de nossa aplicação.



**Figura 11.45** Quarto exemplo em execução

**Listagem 11.5 un\_formulario.pas**

```

unit un_formulario;

interface

uses
  SysUtils, Classes, HTTPApp, WebModu, HTTPProd, CompProd, PagItems,
  SiteProd, WebComp, WebAdapt, MidItems, WebForm;

type
  Tformulario = class(TWebPageModule)
    AdapterPageProducer: TAdapterPageProducer;
    Adapter1: TAdapter;
    Valor1: TAdapterField;
    Valor2: TAdapterField;
    calcula: TAdapterAction;
    AdapterForm1: TAdapterForm;
    AdapterFieldGroup1: TAdapterFieldGroup;
    FldValor1: TAdapterDisplayField;
    FldValor2: TAdapterDisplayField;
    AdapterCommandGroup1: TAdapterCommandGroup;
    Resultado: TAdapterField;
    AdapterFieldGroup2: TAdapterFieldGroup;
    FldResultado: TAdapterDisplayField;
    procedure Valor1GetValue(Sender: TObject; var Value: Variant);
    procedure Valor2GetValue(Sender: TObject; var Value: Variant);
    procedure calculaExecute(Sender: TObject; Params: TStrings);
    procedure ResultadoGetValue(Sender: TObject; var Value: Variant);
  private
    v1:single;
    v2:single;
    vResultado:single;
    { Private declarations }
  public
    { Public declarations }
  end;

  function formulario: Tformulario;

implementation

{$R *.dfm}  {$*.html}

uses WebReq, WebCntxt, WebFact, Variants;

function formulario: Tformulario;
begin
  Result := Tformulario(WebContext.FindModuleClass(Tformulario));
end;

procedure Tformulario.Valor1GetValue(Sender: TObject; var Value: Variant);
begin
  Value:=v1;
end;

```

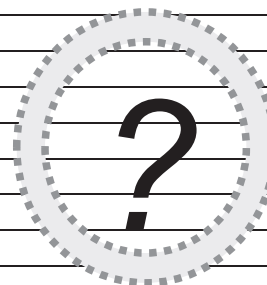
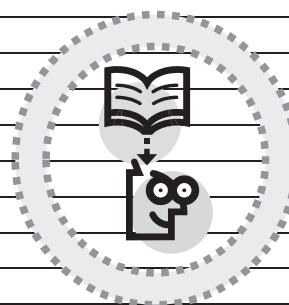


```
procedure TFormulario.Valor2GetValue(Sender: TObject; var Value: Variant);
begin
    Value:=v2;
end;

procedure TFormulario.calculaExecute(Sender: TObject; Params: TStrings);
begin
    v1:= Valor1.ActionValue.Values[0];
    v2:= Valor2.ActionValue.Values[0];
    vResultado:=v1+v2;
end;

procedure TFormulario.ResultadoGetValue(Sender: TObject;
    var Value: Variant);
begin
    Value:=vResultado;
end;

initialization
    if WebRequestHandler <> nil then
        WebRequestHandler.AddWebModuleFactory(TWebPageModuleFactory.Create(Tformulario,
            TWebPageInfo.Create([wpPublished {, wpLoginRequired}], '.html'), crOnDemand, caCache)
        );
end.
```

**Anotações de Dúvidas****Preciso Revisar****Anotações Gerais**