

Capítulo 10

Desenvolvendo Aplicações para Celulares

Visão geral

Amigos, neste capítulo iremos aprender a desenvolver aplicações WAP para celulares.

Mas Facunte, o WAP não foi um fracasso no Brasil?

Depende muito do ponto de vista de cada empresa, de cada brasileiro. O WAP é uma realidade para muitas empresas, que mesmo após diversas resistências, adotaram a tecnologia em suas culturas empresariais. Cultura? É isso mesmo! Cultura. O grande problema do WAP é que nós brasileiros demoramos para nos acostumar com novidades, além, é claro, da barreira de resistência a tornar um hábito certas coisas.

É verdade que o WAP não atingiu o que as “teles” previam, onde todos os seus assinantes utilizariam o WAP para algum tipo de serviço. Isso mesmo, serviço. Aí é que está o pecado da coisa. Como pagar por uma coisa que nem sabemos utilizar? Faltou incentivo cultural. Mas, hoje em dia, muitas empresas aproveitam da tecnologia para força de vendas, troca de mensagens em grupo, pesquisa de campo, entre outras aplicações que veremos ao longo do capítulo.

WAP no Brasil

Continuando com a nossa discussão, será que temos oportunidades de lucro no Brasil? A oportunidade existe, mas poucas empresas estão investindo nesta área, certamente lucrativa. Será que sou um romântico sonhador? Sonhando com o lucro em tempos de vacas magras? Amigos, posso afirmar que não, pois em 2000, quando lancei meu livro de WAP (Wap Guia de Tecnologia), o mercado estava muito aquecido, devido às fortes propagandas de empresas de telefonia em torno do WAP, mas na época, a barreira cultural de nosso país quase levou o sonho destas empresas por água abaixo. Sabem o que elas fizeram? Deixaram o WAP de lado e continuaram a investir em aparelhos móveis, mas destacando apenas sua beleza e algumas funcionalidades.

Facunte, então o sonho do lucro ficou ainda mais distante? Amigos, aí é que entra o consultor, para explicar aos gerentes de tecnologia, CEOs, que a relação custo benefício com o WAP é muito boa, e que o desenvolvimento é quase transparente, pois podemos criar aplicações WAP na maioria das linguagens, como: Delphi, .NET (VB, C#, ASP), PHP, Java (JSP, EJB), entre outras.

Modelos de Aparelhos

A disponibilidade de aparelhos no Brasil é muito grande, e o preço é um forte atrativo, variando entre R\$ 199,00 (modelo LG DM 160), até R\$ 1.999,00 (Nokia 9210). Imaginem uma equipe de vendas, ou até mesmo uma equipe de pesquisadores, com modelos mais baratos, em torno de R\$ 199,00, fechando negócios, elaborando pesquisas, consultando clientes, disponibilidade de estoque, entre outros.

			
Motorola v.8160	LG DM 160	Startac 7860	Ericsson T60
			
LG TM 520	LG DM 515	Siemens SX45	Nokia 9210

Tabela 10.1 Celulares com suporte WAP

Cases de Sucesso

Para provar a força do WAP, vamos conhecer alguns *cases de sucesso*.

Caixa Econômica Federal

A *Caixa Econômica Federal* disponibiliza a todos a informação sobre o saldo do FGTS através da tecnologia WAP. Para ter uma idéia da transparência neste *case*, a mesma aplicação disponibiliza informação via telefone (fixo), Internet e celular (WAP). Totalmente desenvolvida em JAVA, a solução teve seu custo reduzido devido à leve infra-estrutura necessária para abrigar toda a aplicação.

Embora tenha exemplificado o *case* da *Caixa Econômica Federal*, é importante ressaltar que a maioria dos bancos nacionais oferecem variados recursos com a tecnologia WAP.

InvestShop.COM.BR

A instituição financeira InvestShop desenvolveu em conjunto com a EverSystem (talvez a maior empresa de desenvolvimento deste setor) uma solução bastante robusta oferecendo ao usuário um controle total de suas aplicações. Só para ter uma idéia, a aplicação disponibiliza até gráficos de ações em celulares WAP. É realmente incrível.

DETRAN-SP

Um dos pioneiros na tecnologia WAP, o DETRAN de São Paulo disponibiliza informações de Multas, Pontuações na Carteira, entre outros.

Viram como o poder do WAP é grande? Basta uma idéia para transformá-lo em lucro!

Algumas idéias para ganhar dinheiro

Bem, aqui vão algumas sugestões bastante interessantes para ganhar dinheiro com aplicações WAP.

Força de Vendas	Aplicação de auxílio a vendedores externos, com opção de consulta a estoque, faturas de clientes e fechamento de pedidos.
Pesquisas de Campo	Aplicação que auxilia pesquisadores das mais variadas áreas: IBOPE, CENSO, Opinião, Produtos, entre outras.
TimeSheet	Aplicação para TimeSheet de consultores, advogados, técnicos, entre outros.
Processos	Gerenciamento de Processos Jurídicos, onde o advogado ou cliente poderá consultar informações sobre os andamentos dos processos.
m-Ticket	(m=mobile) Vendas de ingressos, passagens aéreas, cinemas, teatros, etc .
m-Finance	Aplicações para o mercado financeiro.
m-Commerce	Comércio em geral através de aplicações WAP.

Entendendo o funcionamento das aplicações WAP

Para entender um pouco como funciona uma aplicação WAP, precisamos conhecer o funcionamento de uma aplicação servidora. Em nosso caso teremos apenas o módulo servidor, pois o cliente da nossa aplicação é um browser com suporte à tecnologia WAP, ou seja, a grande maioria dos celulares comercializados no mercado nacional. Para facilitar a compreensão, a *figura 10.1* ilustra bem o nosso caso:

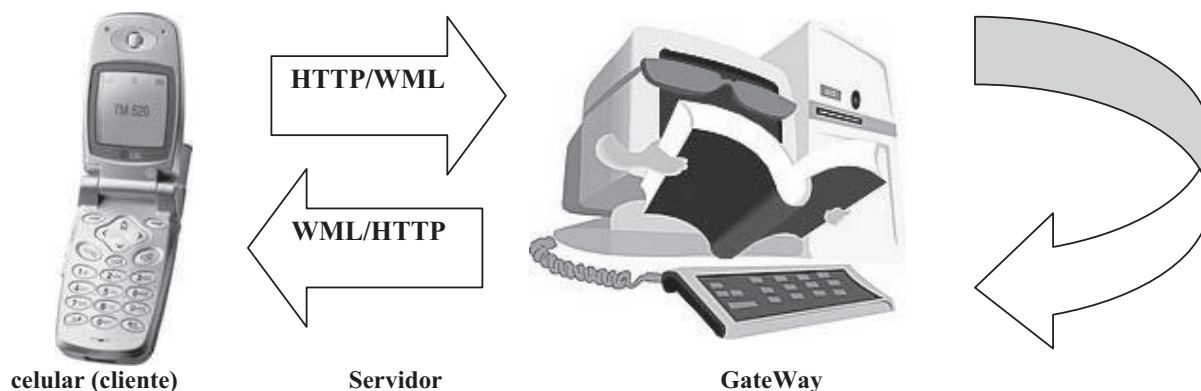


Figura 10.1 Esquema de funcionamento WAP

Na ilustração, temos um **celular**, que funciona como o nosso **cliente (terminal)**, onde solicita os serviços ao servidor. Os **dados (WML)** trafegam através do protocolo **HTTP**. Quando os dados chegam no servidor, o mesmo tem que interpretar a requisição, e esse serviço é feito pelo **GateWay**. O **GateWay** também tem como função empacotar o resultado da requisição, para que o servidor retransmita ao celular.

Em resumo, necessitamos de duas aplicações básicas no servidor:

- Servidor HTTP (recomendo o Apache Server)
- Nossa aplicação servidora.

Um detalhe bastante interessante é que o **Apache Server** traz consigo embutido um excelente **GateWay WAP**. Embora esteja documentado que o **Apache** apenas suporta aplicações **WAP**, ele possui toda a implementação necessária para um perfeito funcionamento.

Para que possamos prosseguir no desenvolvimento de nossas aplicações, precisamos também de um simulador **WAP**. Existem vários no mercado, mas particularmente prefiro o **Deck-it da PyWeb**. A *figura 10.2* ilustra a interface do **Deck-it**.



Figura 10.2 – Deck-it

O **Deck-it** está disponível no site da **PyWeb** (www.pyweb.com) e, assim como o **Apache**, é totalmente freeware.

Mime-Types WAP

Outra coisa importante a saber são os *MimeTypes WAP*. Mas o que são *MimeTypes*? Para que os servidores HTTP reconheçam um tipo de requisição, ou então, um método de envio, é necessária a definição dos *MimeTypes*. Só para clarear um pouco a informação, uma imagem do tipo JPG é definida como *image/JPG*. Com isso o servidor saberá qual o melhor método de transmissão para este tipo de arquivo.

Mime Types do WAP

Tipo de Arquivo	Extensão	MIME Type
Código-fonte WML	.wml	text/vnd.wap.wml
WML compilado	.wmlc	application/vnd.wap.wmlc
Código-fonte WMLScript	.wmls	text/vnd.wap.wmlscript
WMLScript compilado	.wmlcs	application/vnd.wap.wmlscriptc
Imagem bitmap	.wbmp	image/vnd.wap.wbmp

Tabela 10.2 – mime types WAP

Vamos adicionar os Mime-Types ao Apache, embora a partir da versão 1.3.1 já estejam implementados todos os tipos necessários.

No arquivo *httpd.conf* adicione as seguintes linhas:

```
AddType text/vnd.wap.wml .wml
AddType image/vnd.wap.wbmp .wbmp
AddType text/vnd.wap.wmlscript .wmls
AddType application/vnd.wap.wmlscriptc .wlsc
```

Grave o arquivo e reinicie o Apache.

WML

O WML, Wireless Markup Language, é a linguagem de programação para o desenvolvimento de documentos WAP. Muito parecida com o HTML e também com alguns conceitos do XML. Uma das características desta linguagem é a adoção do *case sensitive*, ou seja, *deck* é diferente de *Deck* ou *DECK*. Fique atento.

Normalmente utilizamos letras minúsculas para trabalhar com as *Tags* WML.



Minha opinião

O WML evoluiu muito durante os tempos e o consórcio formado para a criação deste padrão continua investindo na tecnologia. Muitas pessoas comentam que o WAP é ruim, que é isso, que é aquilo, mas como disse anteriormente, a oportunidade está aí, batendo na sua porta. Você vai ficar dando “ouvidos” a quem nunca ao menos tentou fazer a coisa certa?.

TAGS WML

<a>

Cria um hiperlink ligando uma URL, ou ID de outra *Tag*. É recomendado o uso desta *Tag* ao invés da *Tag* **<anchor>**.

Exemplo

```
<a href = "consulta" consulta>
```

Atributos

Nome	Tipo	Obrigatório
id	String	Não
class	String	Não
xml:lang	String	Não
href	href	Sim
title	String	Não

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>xml:lang</i>	<i>Definição da linguagem XML</i>
<i>href</i>	<i>Link de referência</i>
<i>title</i>	<i>Texto, descrevendo o link</i>

<access>

A Tag <access> define o controle de acesso a um deck. Na realidade controla acessos preestabelecidos de acordo com a combinação domain+path, ou seja, você poderá definir se determinado deck poderá ser acessado a partir de um determinado domínio ou path (caminho).

Exemplo:

```
<access domain = "facunte.com.br" path="/VIP">
```

Baseado neste exemplo, vejamos o que é permitido:

permitido	não permitido
www.facunte.com.br/VIP/x.wml	www.facunte.com.br/livros.wml
www.facunte.com.br/VIP/index.wml	www.facunte.com.br/x.wml

Neste exemplo determinamos que o *deck* atual só poderá ser acessado, a partir de uma requisição da domínio **facunte.com.br**, e do caminho **/VIP**, ou seja, somente quem estiver no diretório **/VIP** poderá acessar as informações do *deck* em questão.

Atributos

Nome	Tipo	Obrigatório
id	String	Não
class	String	Não
domain	String	Não
path	String	Não

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>domain</i>	<i>Domínio que poderá acessar o elemento</i>
<i>path</i>	<i>Caminho dentro do domain (domínio) que poderá acessar o elemento. Se não for especificado, todos os caminhos do domain poderão acessar o elemento</i>

<anchor>

Assim como a Tag <a>, cria um hiperlink de um texto. Exemplo:

```
<anchor href="" consulta"> consulta</anchor>
```

Tag utilizada para formatar textos em negrito. É recomendado o uso da Tag **** ao invés de ****, **<i>** ou **<u>**. *Exemplo:*

```
<b> Seja bem-vindo </b>
```

<big>

Utilizada para formatar o texto de maneira enfatizada, grande. *Exemplo:*

```
<big> Seja bem-vindo </big>
```

**
**

Utilizada para concluir a linha atual e iniciar uma nova linha. Pode ser utilizada dentro de tabelas. *Exemplo:*

```
<br> Linha 1
<br> Linha 2
<br> Linha 3
```

<card>

A Tag **<card>** é utilizada para delimitar um conjunto de elementos. A estrutura da linguagem WML pode conter uma coleção de cards, formando um **deck**. *Exemplos:*

```
<card id ="menu">
...
</card>

<card id ="opcao1">
..
</card>

<card id ="opcao2">
..
</card>

<card id ="opcao3">
..
</card>
```

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	
title	String	Não	
newcontext	Boolean	Não	Não
ordered	Boolean	Não	Sim

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>title</i>	<i>Identificação do cartão</i>
<i>newcontext</i>	<i>Indica ao browser para trazer o card em questão atualizado</i>
<i>ordered</i>	<i>Organiza o card</i>

Eventos do elemento Card

Evento	Descrição
<i>onenterforward</i>	<i>Utilizado em conjunto com o elemento GO (veja nas próximas páginas)</i>
<i>onenterbackward</i>	<i>Utilizado em conjunto com o elemento PREV (veja nas próximas páginas)</i>
<i>ontimer</i>	<i>Utilizado para disparar uma referência, que pode ser um CARD ou outra URL, quando o tempo determinado expirar</i>

<do>

Esta Tag é utilizada para acessar informações acima do **card** atual. *Exemplos:*

```
<do type="accept" label="Segundo">
```

```
<do type="accept" label="Primeiro">
```

Atributos

Nome	Tipo	Obrigatório
id	String	Não
class	String	Não
xml:lang	String	Não
type	String	Sim
label	String	Não
name	String	Não
optional	Boolean	Não

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>type</i>	<i>Tipos definidos do elemento (veja tabela a seguir)</i>
<i>label</i>	<i>Texto descritivo do elemento, visível para o usuário</i>
<i>name</i>	<i>Identifica o nome de ligação</i>
<i>optional</i>	<i>Define o elemento como opcional</i>

Complementos do atributo Type

tipo	Descrição
<i>accept</i>	<i>Reconhecimento positivo</i>
<i>prev</i>	<i>Navegação de retorno</i>
<i>help</i>	<i>Ajuda</i>
<i>reset</i>	<i>Limpar, ou resetar</i>
<i>options</i>	<i>Solicitação para opções ou operações adicionais</i>
<i>delete</i>	<i>Apagar item</i>
<i>unknown</i>	<i>De uso genérico</i>

Utilizada para formatar o texto de maneira enfatizada, parecida com a Tag <big>. *Exemplo:*

```
<em> Seja bem-vindo </em>
```

<fieldset>

O elemento *fieldset* permite o agrupamento de campos relacionados ao texto. Este agrupamento permite otimizar o layout e navegação.

Atributos

Nome	Tipo	Obrigatório
id	String	Não
class	String	Não
xml:lang	String	Não
title	String	Não

Exemplo:

```
<fieldset title="endereco">
  Endereco:
  <input type="text" name="fendereco" maxlength="50"/>
  <br/>
  Cidade:
  <input type="text" name="fcidade" maxlength="30"/>
  <br/>
  Estado:
  <input type="text" name="festado" maxlength="2"/>
  <br/>
  Cep:
  <input type="text" name="fcep" maxlength="8"/>
  <br/>
</fieldset>
<fieldset title="telefones">
  Fone_Residencial:
  <input type="text" name="fresidencial" maxlength="20"/>
  <br/>
  Fone_Comercial:
  <input type="text" name="fcomercial" maxlength="20"/>
```

```

<br/>
    Fone_Celular:
<input type="text" name="fcomercial" maxlength="20"/>
<br/>
</fieldset>

```

<go>

A tag <go> permite a navegação entre *cards* ou *decks*, e é utilizado em conjunto com o elemento <do>.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
href	href	Não	
sendreferer	Boolean	Não	false
method	Method	Não	get
accept-charset	String	Não	unknown

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>href</i>	<i>Endereço URL (caminho)</i>
<i>sendreferer</i>	<i>Em caso verdadeiro(True), o browser do usuário deverá especificar o URL do deck contendo esta tarefa</i>
<i>method</i>	<i>Método de envio de dados (POST ou GET)</i>
<i>optional</i>	<i>Define o elemento como opcional</i>

Exemplo:

```

...

<card id="inicio" title="teste" >
  <do type="accept" label="envia">
    <go href="#recebe">
  </do>
  <p>clique em envia para ir ao card recebe</p>
</card>

<card id="recebe" title="teste" >
  <do type="accept" label="volta">
    <go href="#inicio">
  </do>
  <p>clique em volta para retornar</p>

```

<head>

A tag **<head>** contém informações sobre o documento, como os metadados e as tags de controle de acesso.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>

<i>

Utilizada para formatar o texto em itálico. *Exemplo:*

```
<i> Seja bem-vindo </i>
```

A tag **** é utilizada para incluir uma imagem no documento. As imagens devem seguir o padrão WBMP. Veremos um tópico abordando este assunto.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	
alt	String	Não	
scr	URL	Sim	
localscr	String	Não	
vspace	Length	Não	Small, non-zero
hspace	Length	Não	Small, non-zero
align	String	Não	bottom
height	Length	Não	
width	Length	Não	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>alt</i>	<i>Texto a ser exibido enquanto a imagem é carregada, ou numa possível falha em seu carregamento</i>

Atributo	Descrição
<i>scr</i>	<i>Especifica o caminho da imagem</i>
<i>localscr</i>	<i>Alternativa à imagem</i>
<i>vspace</i>	<i>Espaço em branco vertical</i>
<i>hspace</i>	<i>Espaço em branco horizontal</i>
<i>align</i>	<i>Alinhamento da imagem em relação à página</i> <i>top – topo da página</i> <i>middle – meio</i> <i>bottom – alinhada abaixo da página</i>
<i>height</i>	<i>Tamanho vertical da imagem</i>
<i>width</i>	<i>Tamanho horizontal da imagem</i>

Exemplos:

```











```

<input>

A tag **<input>** é utilizada para definir a entrada de dados para o usuário.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	
name	String	Sim	
type	Input_Type	Não	text
value	String	Não	
format	String	Não	*M
emptyok	Boolean	Não	false
size	Number	Não	
maxlength	Number	Não	Infinite
tabindex	Number	Não	
title	String	Não	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	Identificação do elemento (nome único)
<i>class</i>	Classe a que o elemento pertence
<i>name</i>	Nome do elemento — muito importante para tratar o dado
<i>type</i>	Tipo da entrada de dados, que pode ser texto (<i>text</i>), ou senha (<i>pass</i>)
<i>value</i>	Valor padrão do campo
<i>format</i>	Máscara do campo (veja a tabela que segue (A))
<i>emptyok</i>	Permite a escolha de campos opcionais
<i>size</i>	Largura da área de entrada de dados
<i>maxlength</i>	Tamanho máximo, em caracteres, para a entrada de dados
<i>tabindex</i>	Como no Delphi — ordem de entrada
<i>title</i>	Título do campo

Máscaras (A)

Máscara	Descrição
A	Letras maiúsculas e caracteres de pontuação
a	Letras minúsculas e caracteres de pontuação
N	Somente números
X	Somente letras maiúsculas
x	Somente letras minúsculas
M	Qualquer caractere e assume maiúsculo (dependendo do browser)
m	Qualquer caractere e assume minúsculo (dependendo do browser)
*f	Somente caractere numérico
nf	Caractere numérico 0 a 9

Exemplos

```
<input name="nome" type="text" maxlength="50">
<input name="cep" type="text" maxlength="8" format="NNNNNNNN">
<input name="nome" type="text" maxlength="50">
<input name="pais" type="text" value="Brasil">
```

<meta>

A tag **<meta>** contém informações genéricas relativas ao deck do WML. Assim, como no padrão HTML, é utilizada para definir alguns dados do documento.

Atributos

Nome	Tipo	Obrigatório	Padrão
Id	String	Não	
class	String	Não	
http-equiv	String	Não	
name	String	Não	
forua	Boolean	Não	
content	String	Sim	
scheme	String	Não	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	Identificação do elemento (nome único)
<i>class</i>	Classe a que o elemento pertence
<i>http-equiv</i>	Indica que a propriedade deve ser interpretada como um cabeçalho http
<i>forua</i>	Em caso de false , o intermediário deverá remover o “elemento meta” antes do mesmo ser enviado ao cliente; senão (true), os dados são enviados ao browser do usuário
<i>content</i>	Conteúdo do metadata, como por exemplo, um tipo
<i>scheme</i>	Estrutura que deverá ser utilizada para interpretar o metadata

<noop>

A tag **<noop>** é utilizada para cancelar operações realizadas no deck.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	

<onevent>

A tag **<onevent>** associa uma tarefa a um determinado elemento.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
type	String	Sim	

<optgroup>

A tag **<optgroup>** é utilizada para agrupar elementos criados com a tag **<option>**, definindo assim uma hierarquia.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	
title	String	Não	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>title</i>	<i>Referência do grupo</i>

Exemplos:

```
<optgroup title="estado">
  <option title="São Paulo" value="SP">
  <option title="Rio de Janeiro" value="RJ">
  <option title="Bahia" value="BA">
  ....
</optgroup>
```

<option>

A tag **<option>** cria uma opção de escolha em um elemento selecionado.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	
value	String	Não	
title	String	Não	
onpick	href	Não	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>value</i>	<i>Valor atribuído à opção</i>
<i>title</i>	<i>Expressão apresentada ao usuário</i>
<i>onpick</i>	<i>Dispara a chamada de uma URL, quando a opção for selecionada</i>

Exemplos:

```
<optgroup title="sexo">
  <option title="Masculino" value="M">
  <option title="Feminino" value="F">
  ....
</optgroup>
```

```
<optgroup title="idioma">
  <option title="Português" value="P" onpick="/port.wml">
  <option title="Inglês" value="I" onpick="/ingles.wml">
  ....
</optgroup>
```

<p>

A tag <p> inicia um parágrafo com os atributos especificados.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	
align	TAlign	Não	left
mode	Wrapmode	Não	wrap

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>align</i>	<i>Alinhamento do parágrafo</i> <i>left = esquerda</i> <i>right = direita</i> <i>center = centralizado</i>
<i>mode</i>	<i>Especifica o modo de parada do texto</i>

Exemplos:

```
<p align="left">
  Estou alinhado a esquerda
</p>
```

```
<p align="right">
  Estou alinhado a direita
</p>
```

```
<p align="left">
  Estou alinhado a esquerda
</p>
```



```
<p align="center" wrap="nowrap">
  Este texto deverá ser visto sem rolagem e centralizado
</p>
```

<postfield>

A tag **<postfield>** define o nome de um campo e valor para transmissão a um servidor de origem, durante uma requisição de URL.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
name	String	Sim	
value	String	Sim	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>name</i>	<i>Nome do campo</i>
<i>value</i>	<i>Valor do campo</i>

Exemplos:

```
<postfield name="PAIS" value="BRASIL">
```

<prev>

A tag **<prev>** retorna a navegação para a URL anterior no histórico do browser.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	

Sintaxe

```
<prev>
```

<refresh>

A tag **<refresh>** atualiza a URL em atividade e o contexto do dispositivo, de acordo com os dados atuais do servidor.

Atributos

Nome	Tipo	Obrigatório	Padrão
Id	String	Não	
class	String	Não	

*Sintaxe***<refresh>****<select>**

A tag **<select>** inicia uma lista de seleção. Os itens da lista são criados através da tag **<option>**.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	
title	String	Não	
name	String	Sim	
value	String	Não	
iname	String	Não	
ivalue	String	Não	
multiple	Boolean	Não	
tabindex	Number	Não	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>title</i>	<i>Define o título a ser exibido ao usuário</i>
<i>name</i>	<i>Nome do elemento (campo)</i>
<i>value</i>	<i>Recebe o nome da opção selecionada; o nome é definido no atributo iname, e o valor no ivalue</i>
<i>iname</i>	<i>Declaração do nome do elemento</i>
<i>ivalue</i>	<i>Valor da opção selecionada na lista de opções</i>

Exemplos:

```
<select name="linguagens" iname="L" ivalue="1;2;3;4" multiple="true">
  <option value="A">Delphi</option>
  <option value="B">Java</option>
  <option value="A">C++</option>
  <option value="A">.Net</option>
</select>
```

<setvar>

A tag **<setvar>** é utilizada para definir o valor de uma variável.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
name	String	Sim	
value	String	Sim	

Exemplo:

```
<setvar name="SOBRENOME" value="FACUNTE">
```

<small>

A tag **<small>** formata o texto com a fonte pequena.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	

Exemplo:

```
<small> made in Borland </small>
```

A tag **** formata o texto com a fonte destacadas, fortes. Normalmente esta tag é utilizada no lugar das tags ****, **<i>** e **<u>**.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	

Exemplo:

```
<strong> powered by Facunte </strong>
```

<table>

A tag **<table>** é utilizada em conjuntos com as tags **<tr>** e **<td>**, para criar tabelas.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
xml:lang	String	Não	
title	String	Não	
align	String	Não	
columns	Number	Sim	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	<i>Identificação do elemento (nome único)</i>
<i>class</i>	<i>Classe a que o elemento pertence</i>
<i>title</i>	<i>Define o título da tabela</i>
<i>align</i>	<i>Alinhamento dos elementos da tabela</i>
<i>columns</i>	<i>Número de colunas</i>

Exemplo simples:

```
<table align="right" columns =2>
```

```
<td>
```

A tag `<td>` é utilizada para delimitar uma célula.

```
<tr>
```

A tag `<tr>` é utilizada para iniciar uma linha na tabela.

Exemplos de tabelas:

```
<table align="center" columns=3>
<tr>
  <td>marca</td>
  <td>ano</td>
  <td>modelo</td>
</tr>

<tr>
  <td>fiat</td>
  <td>2002</td>
  <td>palio elx</td>
</tr>

<tr>
  <td>mercedes</td>
  <td>2001</td>
  <td>E500</td>
</tr>
```

```

<tr>
  <td>honda</td>
  <td>1999</td>
  <td>civic</td>
</tr>

<tr>
  <td>GM</td>
  <td>1998</td>
  <td>blazer</td>
</tr>

</table>

```

Resultado do exemplo

marca	ano	modelo
fiat	2002	palio elx
mercedes	2001	e500
honda	1999	civic
gm	1998	blazer

<template>

A tag <template> define características comuns a todos os *cards* do dispositivo, disparando alguns eventos.

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
onenterforward	href	Não	
onenterbackward	href	Não	
ontimer	href	Não	

Descritivo dos Atributos

Atributo	Descrição
<i>id</i>	Identificação do elemento (nome único)
<i>class</i>	Classe a que o elemento pertence
<i>onenterforward</i>	Define a URL a ser disparada, caso o usuário acesse um cartão através da tag <do>
<i>onenterbackward</i>	Define a URL a ser disparada, caso o usuário acesse um cartão através da tag <prev>
<i>ontimer</i>	Define a URL a ser disparada, caso o tempo definido na tag <timer> expire

Exemplo:

```

<template
  onenterforward="/home.wml"
  onenterbackward="/delphi.wml"

```

```
ontimer="/login.wml">
</template>
```

```
<timer>
```

A tag **<timer>** é utilizada para disparar um evento ou chamada a partir de um determinado tempo, especificado na propriedade *value* (100 = 1 segundo).

Atributos

Nome	Tipo	Obrigatório	Padrão
id	String	Não	
class	String	Não	
value	Number	Sim	

Exemplo:

```
<card name="card1" ontimer="#card2">
<timer value=100/>
<p>Esta mensagem será destruída em 10 segundos</p>
</card>
<card name="card2">
<p>Mensagem Destruída</p>
</card>
```

Acentos em WML

Nome	Código	Resultado
 	 	espaço
¡	¡	¡
¢	¢	¢
£	£	£
¤t;	¤	¤
¥	¥	¥
¦	¦	¦
§	§	§
¨	¨	¨
©	©	©
¬	¬	¬
®	®	®
°	°	°
±	±	±
´	´	´
¼	¼	¼
½	½	½
¾	¾	¾
¿	¿	¿
À	À	À
Á	Á	Á
Â	Â	Â

Ã	Ã	Ã
Ä	Ä	Ä
Å	Å	Å
Æ	Æ	Æ
Ç	Ç	Ç
È	È	È
É	É	É
Ê	Ê	Ê
Ë	Ë	Ë
Ì	Ì	Ì
Í	Í	Í
Î	Î	Î
Ï	Ï	Ï
Ñ	Ñ	Ñ
Ò	Ò	Ò
Ó	Ó	Ó
Ô	Ô	Ô
Õ	Õ	Õ
Ö	Ö	Ö
×	×	×
Ø	Ø	Ø
Ù	Ù	Ù
Ú	Ú	Ú
Û	Û	Û
Ü	Ü	Ü
Ý	Ý	Ý
þ	Þ	þ
ß	ß	ß
à	à	à
á	á	á
â	â	â
ã	ã	ã
ä	ä	ä
å	å	å
æ	æ	æ
ç	ç	Ç
è	è	è
é	é	é
ê	ê	ê
ë	ë	ë
ì	ì	ì
í	í	í
î	î	î
ï	ï	ï
ñ	ñ	ñ
ò	ò	ò

´	ó	ó
ô	ô	ô
õ	õ	õ
ö	ö	ö
÷	÷	÷
ø	ø	ø
ù	ù	ù
ú	ú	ú
û	û	û
ü	ü	ü
ý	ý	ý
þ	þ	þ
ÿ	ÿ	ÿ

Desenvolvendo Aplicações WAP x Delphi

Agora vamos colocar a mão na massa e desenvolver alguns exemplos de aplicações WAP, com o nosso amigo Delphi.

Primeiro Exemplo (o famoso Hello...)

Sei que isso pode parecer bobagem, mas o nosso companheiro “Hello World” ajuda muito. No Delphi iremos criar uma aplicação servidora no padrão CGI.

A partir do Delphi, selecione as opções *File/New/Other...* e em seguida a opção *Web Server Application*, como ilustra a figura 10.3.

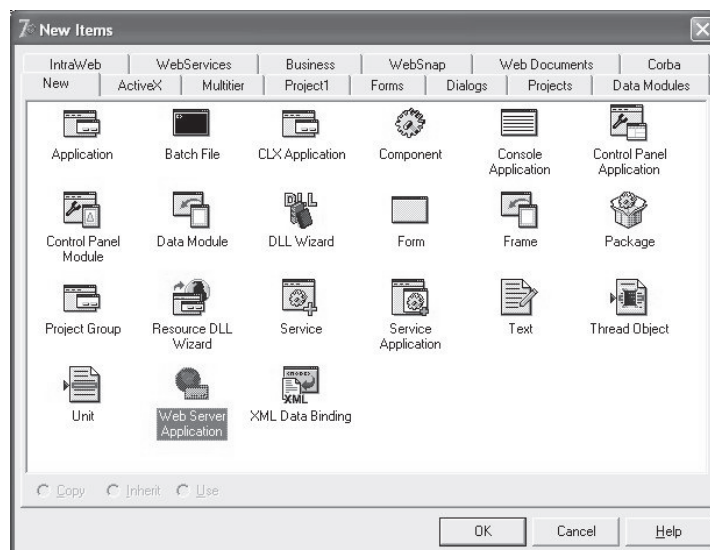


Figura 10.3 Opção Web Server Application

Na janela seguinte selecione a opção *CGI Stand-Alone executable* (figura 10.4).

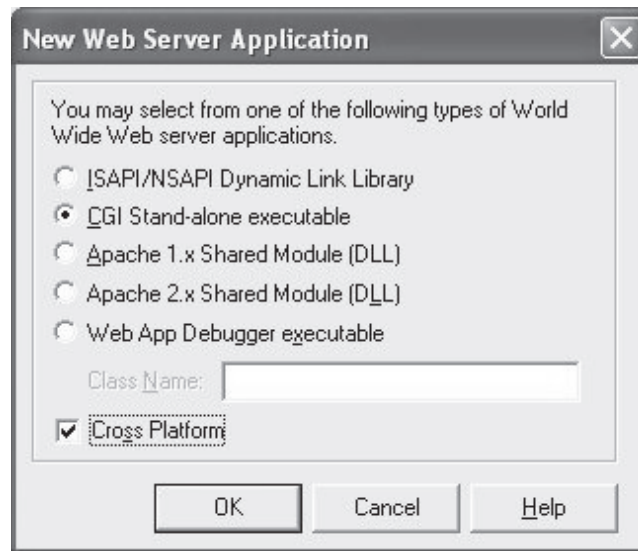


Figura 10.4 Seleção do tipo da aplicação

Em seguida teremos o nosso *WebModule* (figura 10.5),

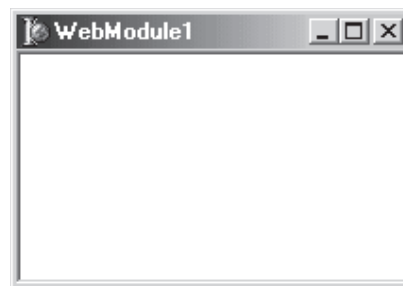


Figura 10.5 WebModule

Bem, seguindo o nosso primeiro projeto, através do duplo-clique no *WebModule*, acesse o editor de *ActionItems* (figura 10.6).

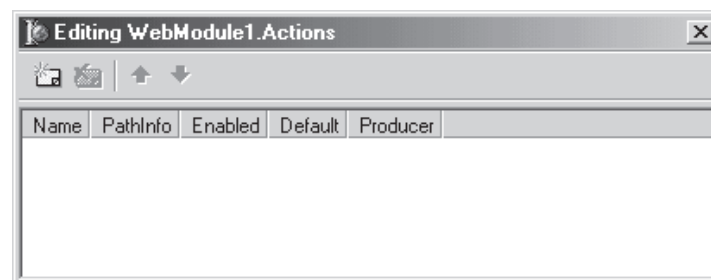


Figura 10.6 editor ActionItems

Clique no primeiro botão do editor para inserir uma nova *Action* (figura 10.7).

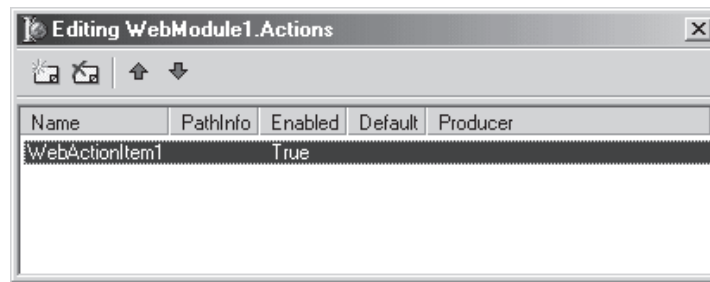



Figura 10.7 ActionItem

Em seguida altere as seguintes propriedades.

OBJETO		
	TWebActionItem	
Objeto	Propriedade	Valor
padrao	Default	True
	Name	padrao
	PathInfo	/padrao

Esta será nossa *Action padrão*, ou seja, caso o usuário não digite nada, além do nome da nossa aplicação, esta *Action* será executada.

Embora a propriedade *PathInfo* possua o mesmo valor da propriedade *Name*, é ela que executa a *Action*, ou seja, no browser o que vale é o valor da *PathInfo*. No evento *OnAction* coloque o seguinte código:

```

001 Response.ContentType:='text/vnd.wap.wml';
002 Response.Content:='<?xml version="1.0"?>'+
003   '<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1//EM"
004   "http://www.wapforum.org/DTD/wml_1.1.xml">'+
   '<wml><card id="cartaum"><p>Ola Mundo!</p></card></wml>';

```

Não se assustem com as denominações do cabeçalho “<DOCTYPE wml...”, isso é um padrão, e o restante iremos conhecendo aos poucos. Vamos analisar o código:

Na linha 001, estou dizendo ao servidor, através do método **Response.ContentType**, que o tipo de informação é no padrão WAP.

```
Response.ContentType:='text/vnd.wap.wml';
```

Em, seguida, nas linhas 002,003 e 004, estou empacotando a resposta (**Response.Content**) e enviando ao servidor, que por sua vez, envia ao celular.

```

Response.Content:='<?xml version="1.0"?>'+
  '<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1//EM"
"http://www.wapforum.org/DTD/wml_1.1.xml">'+
  '<wml><card id="cartaum"><p>Ola Mundo!</p></card></wml>';

```

Grave a *unit* do nosso projeto como *un_ola.pas* e o projeto como *ola.dpr*. Habitualmente testamos nossas aplicações servidoras num browser, como o Internet Explorer, mas neste caso iremos testá-la no nosso simulador **Deck-It**. Lembram dele?

No **Deck-It** digite o que segue na barra de endereços:

`http://localhost/cgi-bin/ola.exe`

A *figura 10.8* ilustra o resultado da nossa aplicação:

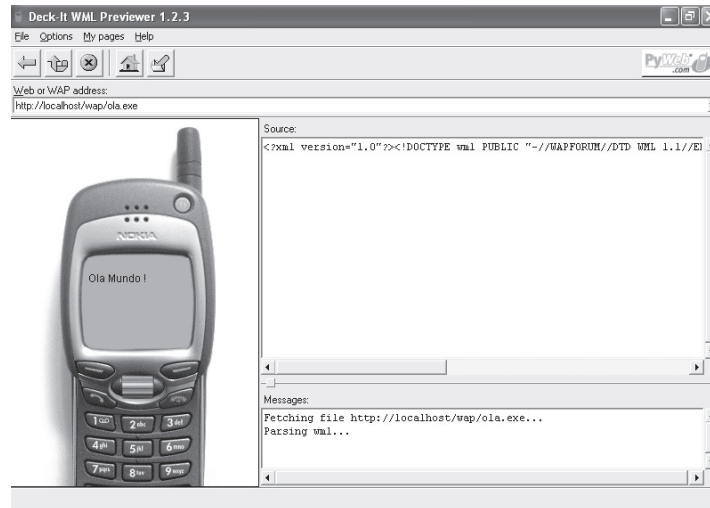


Figura 10.8 Resultado da aplicação


Só para ter uma idéia, você poderia simular em seu próprio celular WAP este exemplo. Mas Facunte, como eu faria isso?

Conecte seu computador à Internet, anote o número do IP fornecido no momento da conexão, exemplo: 200.198.12.1, e digite no seu celular (o modo de entrada para comunicação WAP varia de celular para celular, consulte o manual de instruções), o endereço:

`http://200.198.12.1/cgi-bin/ola.exe`

Repare que o número, na realidade é o seu endereço de IP atual (exemplo).

Continuando com a nossa aplicação, vamos criar mais uma *Action* com as seguintes propriedades:

OBJETO		
		
TWebActionItem		
Objeto	Propriedade	Valor
Theclub	Default	False
	Name	theclub
	PathInfo	/theclub

No evento *OnAction* coloque o seguinte código:

```

001 Response.ContentType='text/vnd.wap.wml';
002 Response.Content='<?xml version="1.0"?>'+
003 ' <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
    "http://www.wapforum.org/DTD/wml_1.1.xml">'+
004 ' <wml><card id="cartaum"><p>Sejam bem-vindos ao The
    Club</p></card></wml>';
  
```

O código é exatamente igual ao anterior, em que apenas substituímos a mensagem. Vamos testar o código? No **Deck-It** digite o que segue na barra de endereços:

http://localhost/ola.exe/theclub

Repare que estamos colocando o *pathinfo* **theclub**. A *figura 10.9* ilustra o resultado da nossa segunda *Action*.

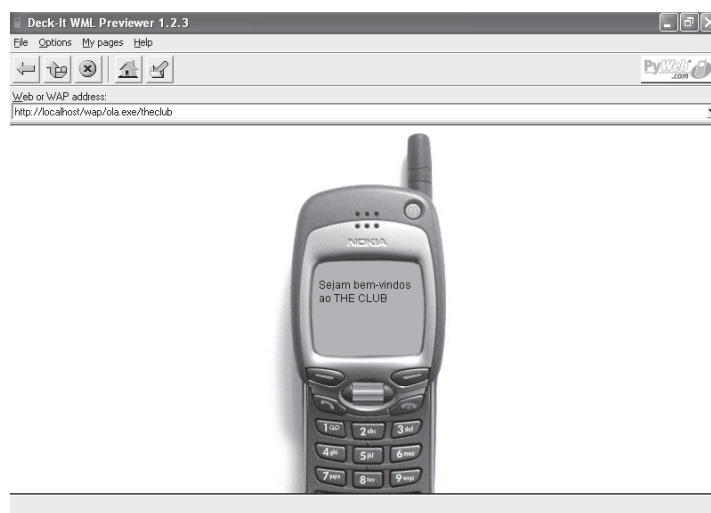


Figura 10.9 Segunda Action

É interessante, não acham? E aí, amigos, estão preparados para brincar com banco de dados e WAP?

Listagem 10.1

```
unit un_ola;

interface

uses
  SysUtils, Classes, HTTPApp;

type
  TWebModule1 = class(TWebModule)
    procedure WebModule1padraoAction(Sender: TObject; Request: TWebRequest;
      Response: TWebResponse; var Handled: Boolean);
    procedure WebModule1theclubAction(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  WebModule1: TWebModule1;

implementation
```

```
{ $R *.DFM }

procedure TWebModule1.WebModule1padraoAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.ContentType:='text/vnd.wap.wml';
  Response.Content:='<?xml version="1.0"?>'+
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1//EM"
"http://www.wapforum.org/DTD/wml_1.1.xml">'+
    '<wml><card id="cartaum"><p>Ola Mundo !</p></card></wml>';
end;

procedure TWebModule1.WebModule1theclubAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.ContentType:='text/vnd.wap.wml';
  Response.Content:='<?xml version="1.0"?>'+
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1//EM"
"http://www.wapforum.org/DTD/wml_1.1.xml">'+
    '<wml><card id="cartaum"><p>Sejam bem-vindos ao THE CLUB</p></card></wml>';
end;

end.
```

Segundo Exemplo (Banco de Dados)

A aplicação que iremos desenvolver, consiste em apresentar uma lista de e-mails que está armazenada em nosso banco de dados.

Através das opções *File/New...*, selecione o item *Web Server Application* (figura 10.10) e clique em OK.

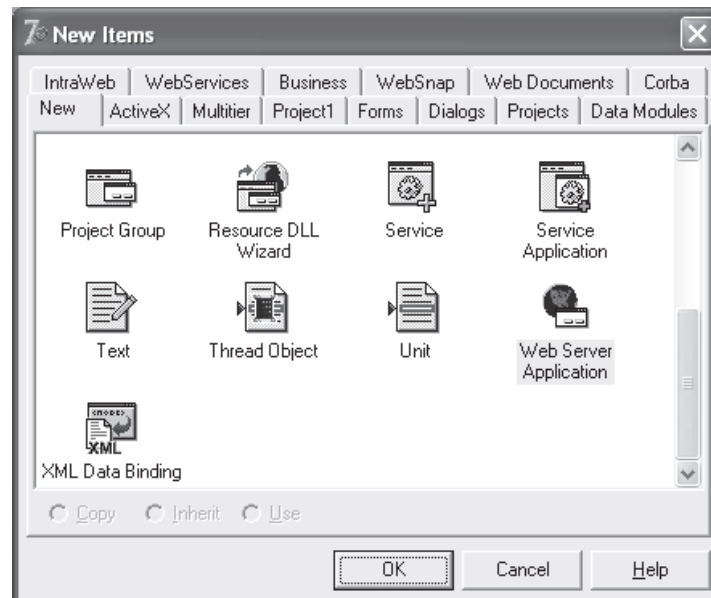


Figura 10.10 Item *Web Server Application*

Em seguida, selecione o tipo de aplicação *CGI*, e marque a opção *Cross Platform* (figura 10.11). Perceba que com isso poderemos executar a mesma aplicação num servidor Linux, sendo que o código deverá ser recompilado em Kylix.

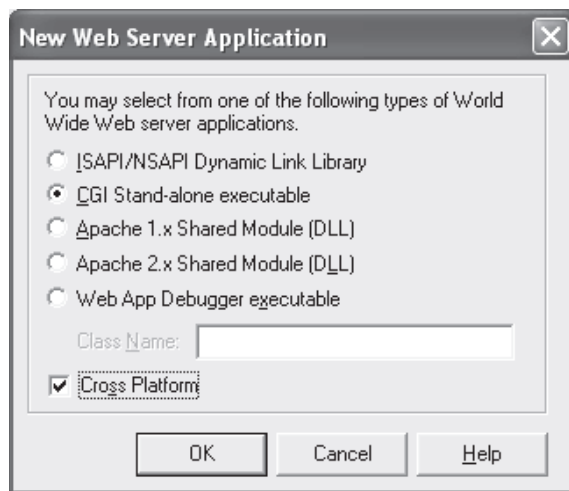


Figura 10.11 Escolha do tipo da aplicação servidora

Agora, neste ponto, precisamos estabelecer a conexão com o nosso banco de dados. Insira um objeto do tipo *TSQLConnection*, e através do duplo-clique, já na tela de configuração, crie uma nova conexão do tipo Interbase, alterando as propriedades que seguem. A figura 10.12 ilustra o diálogo de propriedades do componente *TSQLConnection*.

PROPRIEDADE	VALOR
CommitRetain	False
Database	<i>localhost:c:\cursoweb\clientes.gdb</i>
Password	a famosa masterkey
UserName	o famoso SYSDBA
Name	ConexaoBD

Altere também a propriedade *LoginPrompt* para false. Nunca esqueça de fazer esta alteração, pois numa aplicação servidora, não existe a possibilidade do usuário interagir no login do banco de dados.

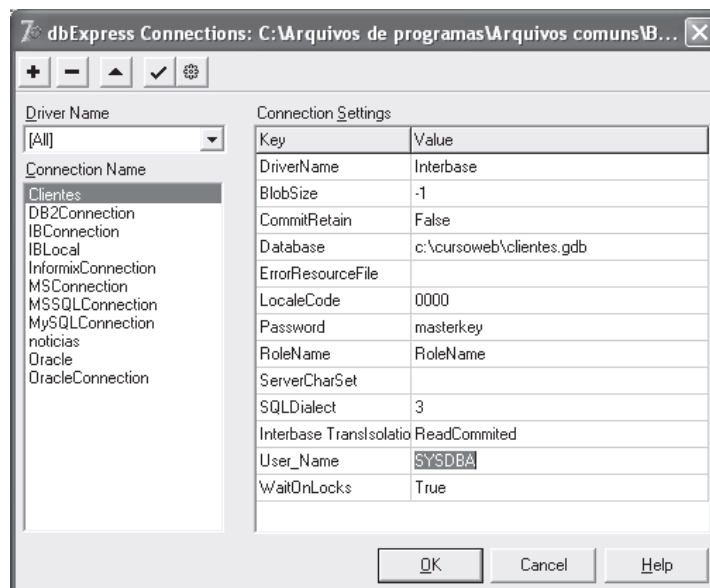


Figura 10.12 Configuração da Conexão

Agora vamos inserir o objeto para manipular nossa tabela de clientes. Insira um objeto do tipo *TSQLEDataSet*, e altere as seguintes propriedades:

PROPRIEDADE	VALOR
SQLConnection	ConexaoBD
CommandText	select * from TBCLIENTE
Active	True

Oba, já estamos chegando lá. Agora vamos inserir um objeto do tipo *TDataSetTableProducer*. Não confundam com *TDataSetPageProducer*. Altere o nome do componente para *producerCliente*, e a propriedade *DataSet* para *SQLCliente*. Através de um duplo-clique teremos a seguinte tela (assistente de configuração – figura 10.13).

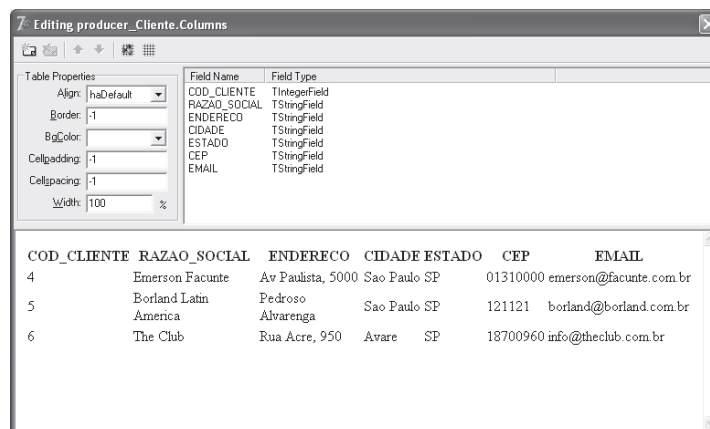


Figura 10.13 Assistente de Configuração

Caso não esteja vendo uma tela parecida (perceba que os dados já estão inseridos no banco e que você deverá inserir seus próprios registros), verifique se está tudo certo com a sua conexão.

Deixe somente o campo *e-mail*, pois iremos listar apenas esta informação em nosso celular. Para realizar a tarefa, selecione os outros campos e aperte a tecla *delete*.

Veja o resultado desta operação, na *figura 10.14*.

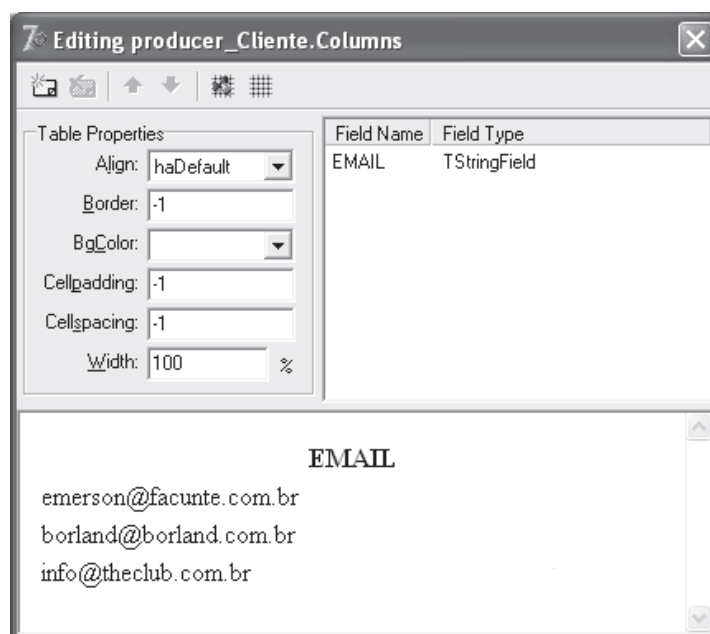


Figura 10.14

Neste ponto, iremos criar nossa *Action* para apresentar a informação no celular.

Crie uma *Action* com o nome *dados*, através do duplo-clique no *WebModule*. Veja a *figura 10.15*.

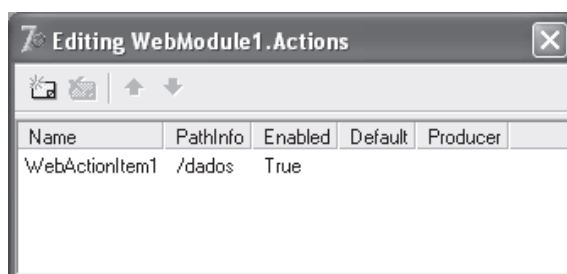


Figura 10.15 Action dados

No evento *OnClick* da *Action dados*, insira o código que segue.

```
// Abre o SQLCliente
SQLCliente.Open;

// Define o tipo do conteúdo
Response.ContentType:='text/vnd.wap.wml';

// Monta a cabeçalho padrão
```



```

Response.Content:='<?xml version="1.0"?>'+
'<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EM"
"http://www.wapforum.org/DTD/wml_1.1.xml">'+
'<wml><card id="cartaum"><p>';

// Insere o conteudo da nossa Tabela producer_Cliente na resposta e finaliza
Response.Content:=Response.Content+producerCliente.Content+

'</p></card></wml>';

// Fecha o SQLCliente
SQLCliente.Close;

```

Com base no exemplo anterior, fica bastante simples entender este código. Primeiro, estamos abrindo o nosso *DataSet* (*SQLCliente*).

```
SQLCliente.Open;
```

Em seguida estamos atribuindo à resposta, o tipo de empacotamento WML.

```
Response.ContentType:='text/vnd.wap.wml';
```

Na próxima instrução, estamos fazendo as definições iniciais, como *cabeçalho* *<?xml...>*, *card* *<card>*, e *início de um parágrafo* *<P>*. Isso é **imprescindível**.

```

Response.Content:='<?xml version="1.0"?>'+
'<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EM"
"http://www.wapforum.org/DTD/wml_1.1.xml">'+
'<wml><card id="cartaum"><p>';

```

Agora vem a parte interessante. Estamos adicionando ao pacote de resposta (*Response.Content*), a informação obtida através do componente *producer_Cliente*, e finalizando o *parágrafo* *<P>*, o *Card* *</card>* e o documento WML *</wml>*.

```

Response.Content:=Response.Content+producerCliente.Content+

'</p></card></wml>';

```

Amigos, agora é só compilar e testar no *Deck-It*. http://localhost/cgi-bin/wap_bd.exe/dados

Vejam o resultado (*figura 10.16*).

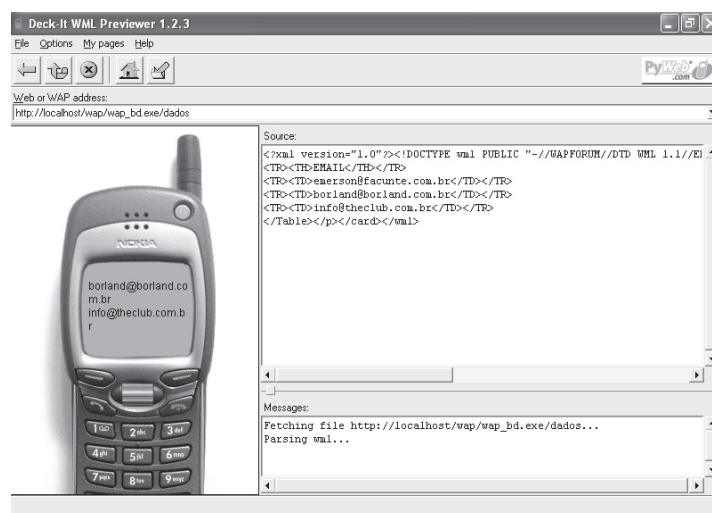


Figura 10.16 Resultado da aplicação.

Listagem 10.2

```
unit un_wap_bd;

interface

uses
  SysUtils, Classes, HTTPApp, DBXpress, FMTBcd, DB, SqlExpr, DBWeb;

type
  TWebModule1 = class(TWebModule)
    ConexaoBD: TSQLConnection;
    SQLCliente: TSQLDataSet;
    producer_Cliente: TDataSetTableProducer;
    procedure WebModule1WebActionItem1Action(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.xrm}

procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  // Abre o SQLCliente
  SQLCliente.Open;

  // Define o tipo do conteúdo
  Response.ContentType:='text/vnd.wap.wml';

  // Monta a cabeçalho padrão
```

```

Response.Content:='<?xml version="1.0"?>'+
'<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EM"
"http://www.wapforum.org/DTD/wml_1.1.xml">'+
'<wml><card id="cartaum"><p>';

// Insere o conteudo da nossa Tabela producer_Cliente na resposta e finaliza
Response.Content:=Response.Content+producer_Cliente.Content+

'</p></card></wml>';

// Fecha o SQLCliente
SQLCliente.Close;

end;

end.

```

Terceiro Exemplo (Inclusão)

Agora vamos desenvolver algo mais interativo, onde o usuário do celular poderá se cadastrar em nosso banco de dados. Através das opções *File/New...*, selecione o item *Web Server Application* (figura 10.10) e clique em OK.

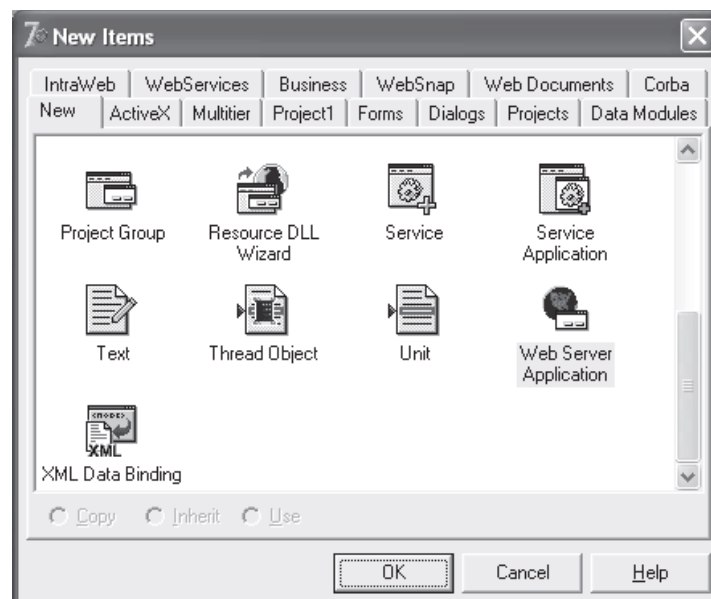


Figura 10.17 Item Web Server Application

Em seguida, selecione o tipo de aplicação *CGI*, e marque a opção *Cross Platform* (figura 10.18).

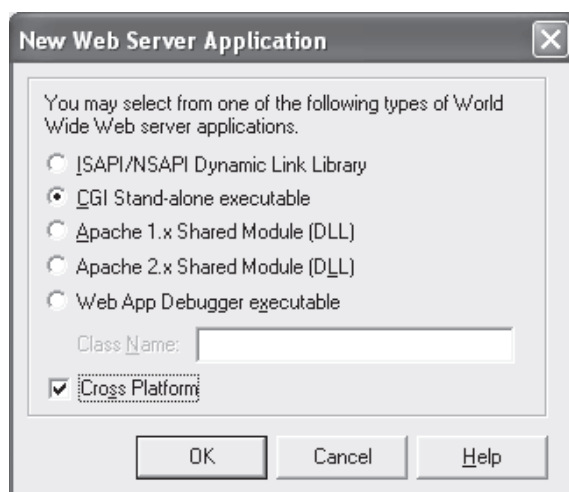


Figura 10.18 Escolha do tipo da aplicação servidora

Agora, neste ponto, precisamos estabelecer a conexão com o nosso banco de dados.

Insira um objeto do tipo *TSQLConnection*, e através do duplo-clique, já na tela de configuração, aponte para a nossa conexão Clientes, criada anteriormente. Vamos relembra os atributos da conexão.

PROPRIEDADE	VALOR
CommitRetain	False
Database	<i>localhost:c:\cursoweb\clientes.gdb</i>
Password	a famosa masterkey
UserName	o famoso SYSDBA
Name	ConexaoBD

Altere também a propriedade *LoginPrompt* para false. Nunca esqueça de fazer esta alteração, pois numa aplicação servidora, não existe a possibilidade do usuário interagir no login do banco de dados.

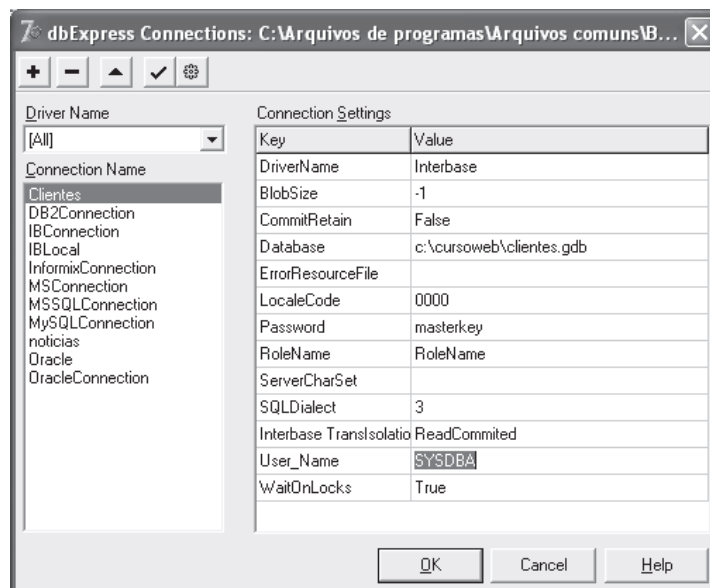


Figura 10.18 Configuração da Conexão

Agora vamos inserir o objeto para manipular nossa tabela de clientes. Insira um objeto do tipo *TSQLDataSet*, e altere as seguintes propriedades:

PROPRIEDADE	VALOR
SQLConnection	ConexaoBD
CommandText	INSERT INTO TBCLIENTE (COD_CLIENTE, RAZAO_SOCIAL, EMAIL) VALUÉS (0,:pRAZAO,:pEMAIL)

Repare que estamos colocando dois parâmetros, (pRAZAO e pEMAIL), que deverão ser configurados através da propriedade *PARAMS* (figura 10.20).

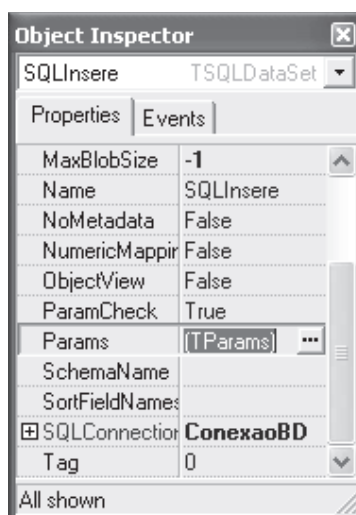


Figura 10.20 Params

Através do duplo-clique na propriedade *Params*, acessamos o editor de *parâmetros* (figura 10.21). Configure os dois parâmetros para o tipo *fiString*, através da propriedade *DataType*.

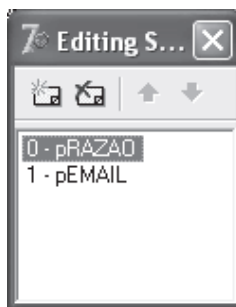


Figura 10.21 Configuração dos Parâmetros

Ok amigos, agora vamos continuar nossa aplicação inserindo um componente do tipo *TPageProducer* e alterar as seguintes propriedades:

PROPRIEDADE	VALOR
Name	ppInclusao
HTMLDoc	(Veja a listagem de código que segue)

Insira este código na propriedade *HTMLDoc* do objeto *ppInclusao*.

```
<wml>
  <card id="inclusao">
    <p align="center">
      Inclusao Clientes
    <br/>
    <br/>
    </p>
    <p>
      Razao Social: <input type="text" name="razao"/>
    <br/>
    eMail : <input type="text" name="email"/>
    <do type="accept" label="CONFIRMA">
      <go href="confirma" method="post">
        <postfield name="razao" value="$razao"/>
        <postfield name="email" value="$email"/>
      </go>
    </do>
    </p>
  </card>
</wml>
```

Agora vamos criar uma variável em nosso WebModule para facilitar nas respostas ao *cliente*, em nosso caso, o *celular*. Antes da seção **implementation**, crie a nossa variável **cabecalho**, como segue:

```
var
  WebModule1: TWebModule1;
  cabecalho: string = '<?xml version="1.0"?>'+
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM/DTD WML 1.1//EM"
```

```
"http://www.wapforum.org/DTD/wml_1.1.xml">';
```

implementation

Neste ponto, iremos criar nossas *Actions* (figura10.22).

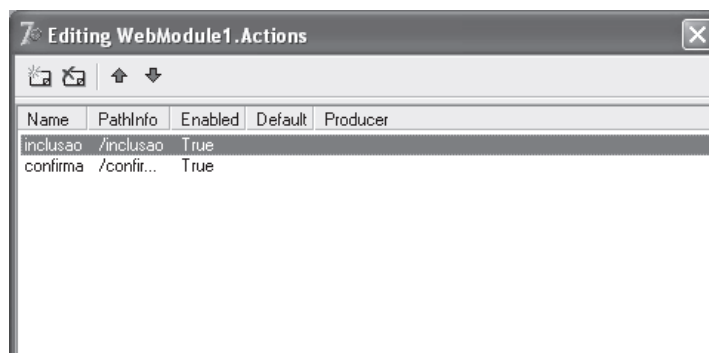


Figura 10.22 Actions

Teremos uma *Action* para apresentar a tela de inclusão, e outra para *confirmar*. Primeiro vamos criar a *Action inclusao*. (sem *acento mesmo*). No evento *OnAction*, insira o código que segue:

```
Response.ContentType:='text/vnd.wap.wml';
Response.Content:=cabecalho+ppInclusao.Content;
```

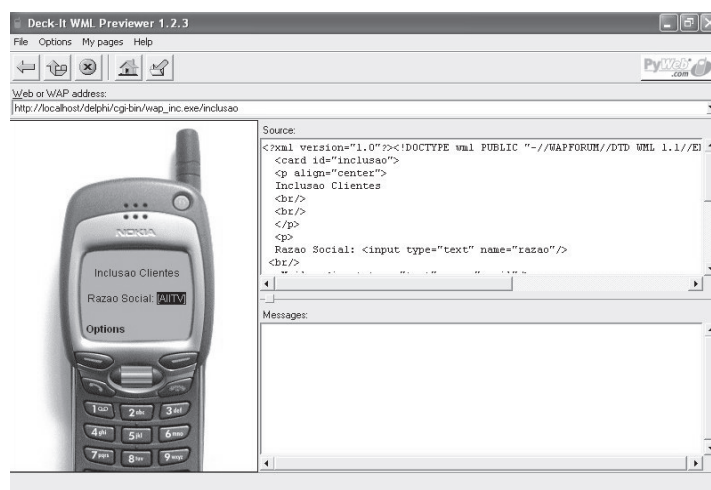
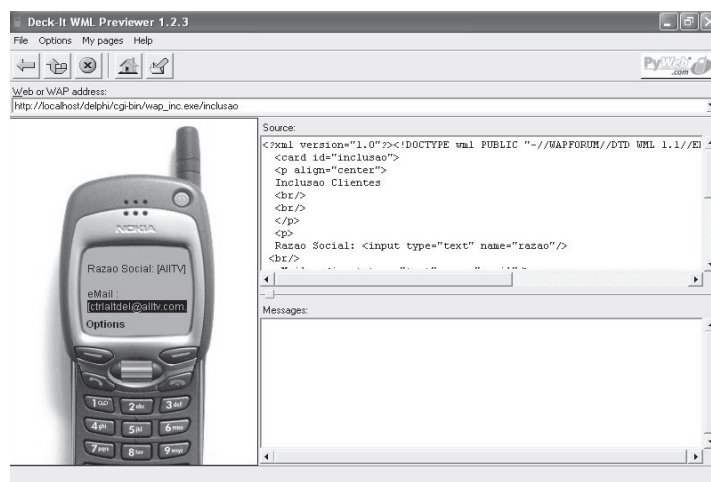
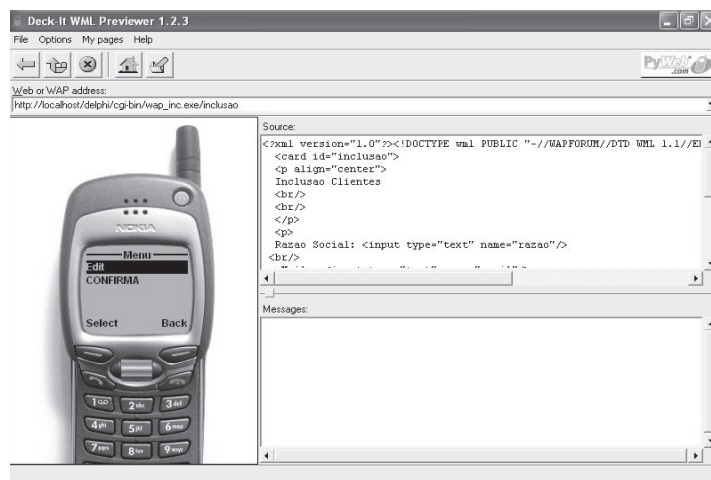
Neste código, estamos informando o tipo de pacote e utilizando nossa variável auxiliar **cabecalho** para compor a nossa resposta, além, é claro, do conteúdo do *Producer ppInclusao*. Bem simples, não?

Em seguida, precisamos criar a nossa *Action confirma*. No evento *OnAction*, insira o código que segue:

```
try
    SQLInserere.ParamByName('pRazao').Value:=Request.ContentFields.Values['razao'];
    SQLInserere.ParamByName('pemail').Value:=Request.ContentFields.Values['email'];
    SQLInserere.ExecSQL();
    Response.ContentType:='text/vnd.wap.wml';
    Response.Content:='<wml><card id="ok"><p>Registro Incluído</p></card></wml>';
except
    Response.ContentType:='text/vnd.wap.wml';
    Response.Content:='<wml><card id="ok"><p>Erro na Inclusão</p></card></wml>';
end;
```

Neste código estamos atribuindo aos parâmetros de nossa *Query SQLInserere*, o conteúdo informado pelo usuário, e transmitido através do método *POST*. O código está protegido através dos comandos *try/except* e, em caso de sucesso, apresentamos ao usuário a mensagem “**Registro Incluído**”. Caso contrário, apresentamos a mensagem “**Erro na Inclusão**”. As figuras 10.23, 10.24, 10.25 e 10.26 ilustram nossa aplicação em ação.

Para chamar a aplicação, utilize: **http://localhost/cgi-bin/wap_inc.exe/dados**

*Figura 10.23 Digitando Razão Social**Figura 10.24 Digitando e-mail**Figura 10.25 Confirmando dados*

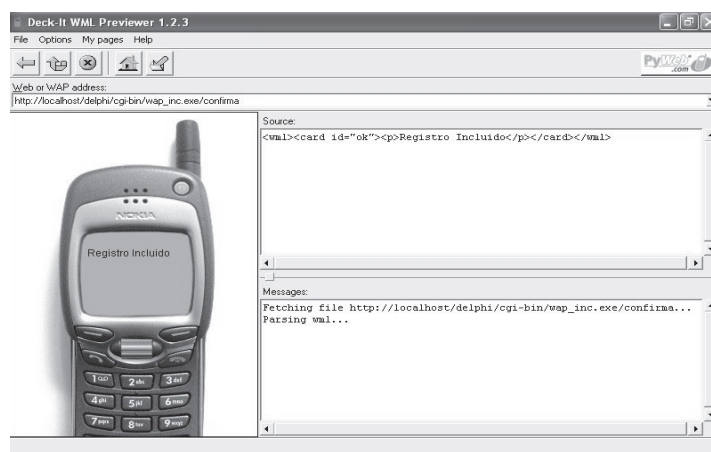


Figura 10.26 Registro incluído com sucesso

Amigos, com um pouquinho de esforço, dá para criar uma aplicação bem completa para celulares. Digo esforço pois existem vários limites nos aparelhos, como tamanho de tela, capacidade de dados, entre outras coisas, que nos fazem voltar ao tempo da magia e dedicação, onde fazíamos malabarismos com o Clipper, o C e o Turbo Pascal para oferecer telas agradáveis aos nossos usuários.

Encerramos por aqui este tópico, e espero que tenham gostado.

Listagem 10.3 *WAP_Inclusao*

```
unit un_wap_inclusao;

interface

uses
  SysUtils, Classes, HTTPApp, DBXpress, FMTBcd, HTTPProd, DB, SqlExpr;

type
  TWebModule1 = class(TWebModule)
    ConexaoBD: TSQLConnection;
    SQLInsere: TSQLDataSet;
    ppInclusao: TPageProducer;
    procedure WebModulelinclusaoAction(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
    procedure WebModule1confirmaAction(Sender: TObject;
      Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  WebModule1: TWebModule1;
  cabecalho:string = '<?xml version="1.0"?>'+
    '<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EM"
"http://www.wapforum.org/DTD/wml_1.1.xml">';
```

```
implementation

{$R *.xfm}

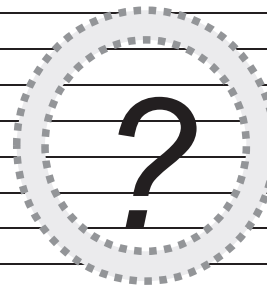
procedure TWebModule1.WebModule1inclusaoAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.ContentType:='text/vnd.wap.wml';
  Response.Content:=cabecalho+ppInclusao.Content;
end;

procedure TWebModule1.WebModule1confirmaAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  try
    SQLInsere.ParamByName('pRazao').Value:=Request.ContentFields.Values['razao'];
    SQLInsere.ParamByName('pemail').Value:=Request.ContentFields.Values['email'];
    SQLInsere.ExecSQL();
    Response.ContentType:='text/vnd.wap.wml';
    Response.Content:='<wml><card id="ok"><p>Registro Incluido</p></card></wml>';
  except
    Response.ContentType:='text/vnd.wap.wml';
    Response.Content:='<wml><card id="ok"><p>Erro na Inclusão</p></card></wml>';
  end;

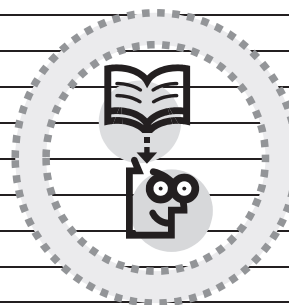
end;

end.
```

Anotações de Dúvidas



Preciso Revisar



Anotações Gerais

