

Integrando sua aplicação com a busca de CEPs dos Correios



RODRIGO SENDIN

(rodrigo.sendin@terra.com.br)

é tecnólogo formado pela FATEC de Americana. Trabalha com desenvolvimento de software há mais de 10 anos, escreve artigos para MSDN Magazine e Clube Delphi. Também desenvolve treinamentos de .NET, e atualmente é desenvolvedor na TauNet Consulting.



LUCIANO PIMENTA

(lucianopimenta@clubedelphi.net)

é Técnico em Processamento de Dados, Editor Técnico da Revista ClubeDelphi e Editor do Portal DevMedia (www.devmedia.com.br). Palestrante da 4ª edição da Borland Conference (BorCon).

Você já precisou incluir em suas aplicações uma rotina para busca de CEPs? Se você já desenvolveu algo que envolva cadastros com dados de endereços para entrega, cobrança, faturamento etc., com certeza você já precisou disso.

Uma solução muito comum era usar um arquivo Access que o próprio Correio disponibilizou em seu site ou através de CD, há algum tempo. Nesse banco, tínhamos uma tabela com todos os CEPs do país inteiro, com dados das ruas, bairros, cidades e estados.

Dava um bocadinho de trabalho, pois tínhamos que manter um banco enorme (como também migrar o banco para a versão que usamos em produção), sem falar das atualizações que eram frequentes. Hoje, esse banco não é mais disponibilizado e quem utiliza não tem mais atualizações, podendo trabalhar com dados defasados. Então como fazer isso hoje? É exatamente o que veremos neste artigo.

Solução HTML

Se entrarmos no site dos Correios hoje (www.correios.com.br), na página principal, temos uma opção na parte inferior chamada *Busca CEP*. Clique sobre ela e veja que temos uma série de opções que envolvem a busca de CEP dos Correios (**Figura 1**).

Nota: As opções do site descritas neste artigo estavam disponíveis até o fechamento dessa edição.

A primeira opção que temos no menu lateral esquerdo, chama-se *Coloque a busca ao CEP no seu site*. Ao clicar nessa opção você verá que basta preencher um pequeno formulário e é possível fazer o download de duas páginas HTML (*busca_cep.html* e *busca_por_cep.html*). O primeiro possui um formulário onde podemos preencher os dados de um endereço e o segundo pergunta apenas o CEP.

Em ambas as páginas o resultado é mostrado em uma nova instância do browser, em uma página dos Correios. Essa é uma solução boa se você quer apenas pesquisar as informações de CEP em seu site.

Mas para alguns, essa pode ser uma solução incompleta, imaginando por exemplo que você queira receber esses dados e colocá-los diretamente em componentes da aplicação, seja ela Web ou Windows.

Consultar o CEP no Office

Outra opção que os Correios oferece é incluir a pesquisa de CEP no Office. Ainda na **Figura 1** veja que a última opção do menu é a *Veja como consultar o CEP no OFFICE da Microsoft*. Clique nessa opção e veja que temos uma explicação para incluir a pesquisa de CEP nas opções de pesquisa do Office.

Seguindo os passos, você registrará um Web Service que permite realizar as pesquisas de CEP. Se você não sabia disso, siga os passos e veja que é uma solução bem interessante, principalmente se você ou seus usuários usam muito o Office.

O Web Service

Apesar de não estar claramente divulgado no site dos Correios, a busca de CEP no Office usa um Web Service, que naturalmente pode ser utilizado em nossas aplicações .NET. O endereço é <http://consultacep.correios.com.br/office2003/Query.asmx>. Devo adverti-los que essa não é uma tarefa muito simples.

Esse Web Service foi desenvolvido para ser consumido pelo Office, portanto temos um certo esforço para utilizá-lo em aplicações .NET. Vamos começar com um teste simples. Abra o Delphi for .NET (versão 8, 2005 ou 2006) e crie uma nova aplicação ASP.NET (você pode fazer em Windows Forms – Desktop se quiser). Dê o nome de “CEP” para a aplicação. Na página *Default.aspx* crie uma interface igual a demonstrada na **Figura 2**.

Veja que esse é um formulário bem simples onde estamos solicitando o CEP em um *TextBox* chamado “txtCEP”. Abaixo temos um botão chamado “btnPesquisar” e um outro *TextBox* chamado “txtRetorno”, onde apresentaremos o retorno do Web Service dos Correios.

No *Project Manager* clique com o botão direito sobre o projeto. Escolha a opção *Add Web Reference*, informe no endereço de pesquisa o Web Service dos Correios (<http://consultacep.correios.com.br/office2003/Query.asmx>) e clique no botão *Go* (uma seta azul).

Veja na **Figura 3** que o Web Service dos Correios expõe uma

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como criar e consumir Web Services no Delphi 2006.

www.devmedia.com.br/articles/viewcomp.asp?comp=2878

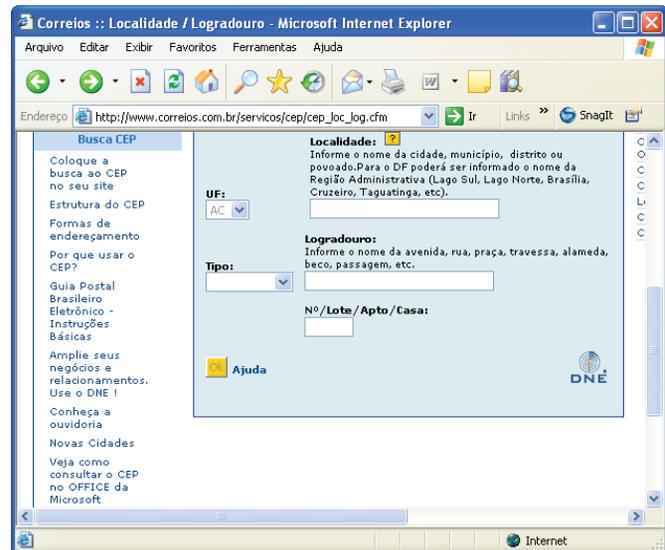


Figura 1. Opções de busca de CEP disponíveis no site dos Correios



Figura 2. Interface para a busca de CEP

classe chamada *QueryProcessor*, que possui um método chamado *Query*. Para adicionar a referência, basta clicar em *Add Reference*.

Para continuarmos, vá ao código da *Default.aspx* e inclua no *uses* os seguintes namespaces:

```
uses System.IO, System.Xml, System.Text,
    correios.Query;
```

Em seguida, ainda no código da página *Default.aspx*, crie uma função, na seção *private*, chamada *GeraXML*. Adicione o código da **Listagem 1** na referida função.

Como você deve estar imaginando, esse método gerará um XML contendo o CEP que será informado no *txtCEP*. Isso é necessário, pois o Web Service foi feito para receber pesquisas de CEP do Office, portanto ele recebe um XML formatado nesses padrões.

Veja que através de um *MemoryStream* estamos criando um *XmlTextWriter* em memória, que no final é retornado como *string* por um *StreamReader*. A única informação que esse XML carrega é o CEP informado no *TextBox*.

Para finalizar, volte ao *design* da página, dê um duplo clique no botão e inclua o código da **Listagem 2**.

Listagem 1. Código que gera o XML para a busca de CEP

```
function GeraXML: string;
var
  Stream: MemoryStream;
  XmlWriter: XmlTextWriter;
  Reader: StreamReader;
begin
  Stream := MemoryStream.Create;
  XmlWriter := XmlTextWriter.Create(
    Stream, Encoding.UTF8);
  XmlWriter.WriteStartDocument();
  XmlWriter.WriteStartElement('QueryPacket',
    'urn:Microsoft.Search.Query');
  XmlWriter.WriteStartElement('Query');
  XmlWriter.WriteStartElement('Context');
  XmlWriter.WriteStartElement('Context');
  XmlWriter.WriteStartElement('QueryText');
  XmlWriter.WriteString(txtCEP.Text);
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndElement();
  XmlWriter.WriteStartElement('OfficeContext',
    'urn:Microsoft.Search.Query.Office.Context');
  XmlWriter.WriteStartElement('ApplicationContext');
  XmlWriter.WriteStartElement('Name');
  XmlWriter.WriteString('Microsoft Office');
  XmlWriter.WriteEndElement();
  XmlWriter.WriteStartElement('Version');
  XmlWriter.WriteString('11.0.6568');
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndElement();
  XmlWriter.Flush();
  Stream.Flush();
  Stream.Position := 0;
  Reader := StreamReader.Create(Stream);
  XmlWriter := nil;
  Stream := nil;
  Result := Reader.ReadToEnd().ToString();
end;
```

Listagem 2. Evento Click que fará a chamada do Web Service

```
var
  Busca: correios.Query.QueryProcessor;
begin
  txtRetorno.Text := '';
  try
    Busca := correios.Query.QueryProcessor.Create;
    txtRetorno.Text := Busca.Query(GeraXML);
  except
    on E: Exception do
      txtRetorno.Text := E.ToString();
  end;
end;
```

Observe que estamos fazendo a chamada ao Web Service utilizando o *Query*, passando como parâmetro a *string* que contém o XML que criamos no *GeraXML*. Vamos fazer um teste? Salve, compile e execute o projeto.

Como mostra a **Figura 4**, informe o CEP sem nenhum caractere de máscara e clique em *Pesquisar*.

Veja que o retorno do Web Service também é um XML e que se você observar o conteúdo desse, verá que dentro da *Tag*

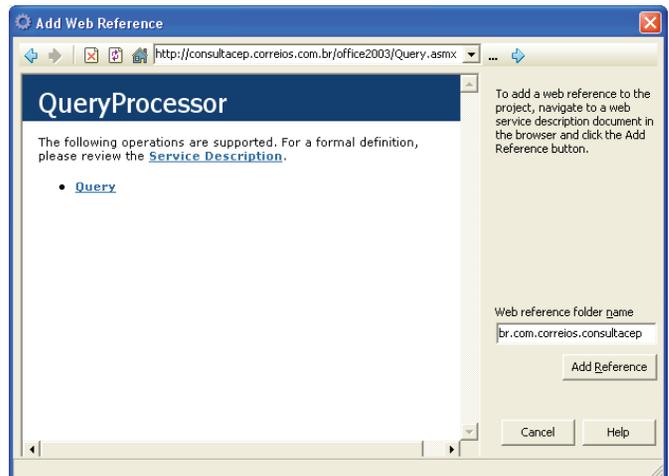


Figura 3. Adicionando a referência ao Web Service

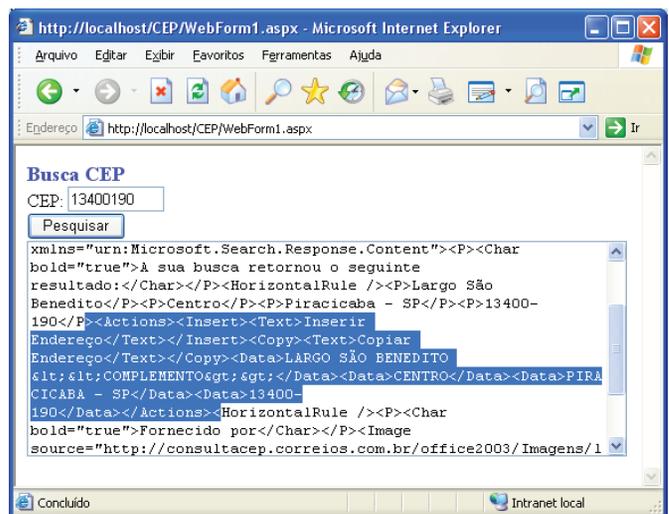


Figura 4. Testando a busca de CEP simples

Actions, estão os dados de Endereço do CEP em questão. Basta então, recuperar essas informações utilizá-las da forma mais conveniente para a aplicação.

Busca avançada

Vimos como realizar uma busca simples, onde através do CEP podemos pesquisar qual é o endereço. Agora vamos fazer uma busca avançada, onde, através do endereço podemos pesquisar qual é o CEP.

Para isso, vamos criar uma nova página (você pode usar a mesma página, mas por questões de organização, vamos criar outra). Inclua alguns componentes, como mostra a **Figura 5** (adicionei uma tabela para organizar melhor o alinhamento dos componentes).

Veja que esse é um formulário onde preenchemos todos os dados de endereço. Utilizamos apenas *TextBoxes*, e cada um deles foi nomeado de acordo com o seu conteúdo: "txtEstado", "txtCidade", "txtLogradouro", "txtEndereco", "txtNumero" e "txtCep".

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como criar arquivos XML com o *XmlTextWriter*.

www.devmedia.com.br/articles/viewcomp.asp?comp=3275

O *TextBox* que receberá o XML tem o mesmo nome do anterior (*txtRetorno*), assim como o botão *Pesquisar* (*btnPesquisar*). Em seguida, adicione mais uma função, chamada *GeraXMLAvancado* e codifique-a com o código da **Listagem 3**.

Listagem 3. Função que gera o XML para a busca avançada

```
uses System.IO, System.Xml, System.Text,
    correios.Query;
...
function GeraXMLAvancado: string;
var
    Stream: MemoryStream;
    XmlWriter: XmlTextWriter;
    Reader: StreamReader;
begin
    Stream := MemoryStream.Create();
    XmlWriter := XmlTextWriter.Create(Stream, Encoding.UTF8);
    XmlWriter.WriteStartDocument();
    XmlWriter.WriteStartElement('QueryPacket',
        'urn:Microsoft.Search.Query');
    XmlWriter.WriteStartElement('Query');
    XmlWriter.WriteStartElement('Context');
    XmlWriter.WriteStartElement('QueryText');
    XmlWriter.WriteString('Logradouro');
    XmlWriter.WriteEndElement();
    XmlWriter.WriteStartElement('Requery');
    XmlWriter.WriteStartElement('ServiceParameters',
        'urn:Microsoft.Search.Office.ServiceParameters');
    XmlWriter.WriteStartElement('Parameters');
    XmlWriter.WriteStartElement('AdvancedSearchEditUF');
    XmlWriter.WriteString(txtEstado.Text);
    XmlWriter.WriteEndElement();
    XmlWriter.WriteStartElement('AdvancedSearchEditLocalidade');
    XmlWriter.WriteString(txtCidade.Text);
    XmlWriter.WriteEndElement();
    XmlWriter.WriteStartElement('AdvancedSearchEditTipoLogradouro');
    XmlWriter.WriteString(txtLogradouro.Text);
    XmlWriter.WriteEndElement();
    XmlWriter.WriteStartElement('AdvancedSearchEditLogradouro');
    XmlWriter.WriteString(txtEndereco.Text);
    XmlWriter.WriteEndElement();
    XmlWriter.WriteStartElement('AdvancedSearchEditNumero');
    XmlWriter.WriteString(txtNumero.Text);
    XmlWriter.WriteEndElement();
    XmlWriter.WriteEndElement();
    XmlWriter.WriteEndElement();
    XmlWriter.WriteEndElement();
    XmlWriter.WriteEndElement();
    XmlWriter.WriteStartElement('OfficeContext',
        'urn:Microsoft.Search.Query.Office.Context');
    XmlWriter.WriteStartElement('ApplicationContext');
    XmlWriter.WriteStartElement('Name');
    XmlWriter.WriteString('Microsoft Office');
    XmlWriter.WriteEndElement();
    XmlWriter.WriteStartElement('Version');
    XmlWriter.WriteString('(11.0.6568)');
    XmlWriter.WriteEndElement();
    XmlWriter.WriteEndElement();
    XmlWriter.WriteEndElement();
    XmlWriter.WriteEndElement();
    XmlWriter.WriteEndElement();
    XmlWriter.Flush;
    Stream.Flush;
    Stream.Position := 0;
    Reader := StreamReader.Create(Stream);
    XmlWriter := nil;
    Stream := nil;
    Result := Reader.ReadToEnd.ToString;
end;
```

Busca CEP

Figura 5. Formulário para busca de CEP avançada

Como você pode observar, o XML gerado é bem parecido com o anterior, seguindo os mesmos padrões. A grande diferença é que nesse, estamos passando os dados de um Endereço para que o Web Service retorne o CEP ou os CEPs do mesmo.

Como você poderá ver na prática, é possível realizar a pesquisa com ou sem o número do logradouro. Caso seja feita uma pesquisa sem o número, é provável que mais de um CEP seja encontrado. Vamos fazer um teste, adicione no *btnPesquisar* o código da **Listagem 4**.

Listagem 4. Realiza a busca avançada

```
var
    Busca: correios.Query.QueryProcessor;
begin
    txtRetorno.Text := '';
    try
        Busca := correios.Query.QueryProcessor.Create;
        txtRetorno.Text := Busca.Query(GeraXMLAvancado);
    except
        on E: Exception do
            txtRetorno.Text := E.ToString;
    end;
end;
```

Salve, compile e execute (seja você criou uma nova página, verifique se a mesma é a página inicial da aplicação). Como mostra a **Figura 6**, preencha os campos: *Estado*, *Cidade*, *Tipo Logradouro* e *Endereço* e em seguida clique em *Pesquisar*.

Veja no *TextBox* de retorno que o XML é bem maior e mais complexo que o anterior, na busca simples. Nesse XML temos

+ de 80.000 membros cadastrados
 + de 15.000 exemplos com fontes
 + de 900 apostilas
 + de 4.000 dicas
 Fórum Delphi
 Artigos

TOTALMENTE GRÁTIS
www.delphi.eti.br

Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

primeiramente uma orientação do próprio Web Service, dizendo que os campos de *Estado* e *Tipo Logradouro* precisam respeitar um conjunto de valores válidos, informados no XML. Isso ajuda a preencher nossas próprias listas de seleção para que o usuário possa escolher os valores.

E veja que o resultado da pesquisa é retornado dentro da *tag Heading*. Faça outros testes e veja que alguns endereços poderão retornar mais que um CEP identificados também na *tag Heading*.

Recuperando o resultado em componentes

Vamos agora criar uma rotina para recuperar os valores retornados pelo Web Service no XML e incluir esses valores em componentes da nossa página. Faremos um exemplo com a consulta simples de CEP, para isso sua página deve ficar como demonstrada na **Figura 7** (crie uma nova página na aplicação).

Veja que nesse formulário apenas perguntaremos o CEP (no "txtCEP") e colocaremos o resultado da pesquisa em quatro *TextBoxes*: "txtEndereco", "txtBairro", "txtCidade" e "txtEstado". Assim podemos simular o preenchimento automático desses campos em um formulário de cadastro de clientes.

Adicione na página criada o *GeraXML* do primeiro exemplo. Não esqueça de adicionar no *uses* os namespaces mostrados anteriormente, juntamente com *System.Collections* para que possamos trabalhar com um *ArrayList*. Em seguida, inclua o código da **Listagem 5** no evento *Click* do *btnPesquisar*.

Listagem 5. Recupera valores do XML e coloca em *TextBoxes*

```
var
  Busca: correios.Query.QueryProcessor;
  retorno: string;
  XmlDoc: XmlDocument;
  campos: ArrayList;
  CidadeUF: &Array;
begin
  try
    Busca := correios.Query.QueryProcessor.Create;
    retorno := Busca.Query(GeraXML);
    XmlDoc := XmlDocument.Create;
    XmlDoc.LoadXml(retorno);
    campos := ArrayList.Create;
    campos := VarreXML(XmlDoc.DocumentElement,
      ChildNodes, campos);
    ( Armazenando Endereco e Bairro )
    txtEndereco.Text := campos[0].ToString.Replace(
      '<<COMPLEMENTO>>', '');
    txtBairro.Text := campos[1].ToString;
    ( Separando Cidade do Estado )
    CidadeUF := campos[2].ToString.Split(['.']);
    txtCidade.Text := CidadeUF[0].ToString.Trim;
    txtEstado.Text := CidadeUF[1].ToString.Trim;
  except
    on E: Exception do
      Response.Write(E.ToString);
  end;
end;
```

Observações da Listagem 5

Veja que através do *Replace* estamos removendo a string "<<COMPLEMENTO>>" que vem no *Endereco* no XML. Através do *Split* separamos a *Cidade* do *Estado*, que no XML vem no mesmo campo.

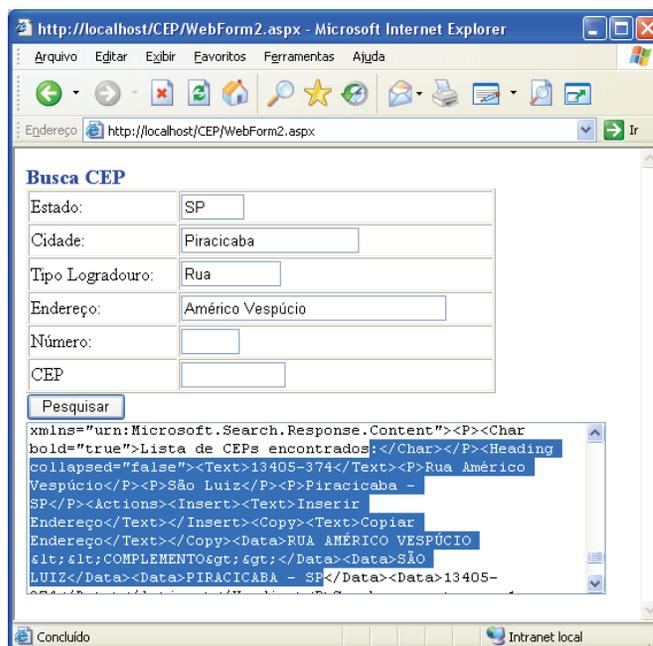


Figura 6. Resultado da busca avançada de CEP

Cadastro de Clientes

CEP:	<input type="text"/>	<input type="button" value="Pesquisar"/>
Endereço:	<input type="text"/>	
Número:	<input type="text"/>	
Bairro:	<input type="text"/>	
Cidade:	<input type="text"/>	
Estado:	<input type="text"/>	

Figura 7. Formulário para retorno do resultado em componentes

Veja que estamos fazendo a chamada ao *GeraXML* que criamos anteriormente para a busca de CEP simples, e o retorno está sendo armazenado em uma *string*. Através de um objeto da classe *XmlDocument* estamos carregando a *string* com o conteúdo do XML.

Existem diversas formas de ler um XML, estamos utilizando o *XmlDocument* por ser uma das mais simples, porém você pode utilizar a classe que preferir.

Observe que após carregar o *XmlDocument* estamos criando um *ArrayList* chamado *campos*. Em seguida estamos chamando o *VarreXML*, passando como parâmetro os *ChildNodes* do *XmlDocument* e a *ArrayList* que criamos.

O retorno dessa função é a própria *ArrayList*, que conterá os valores para preencher os *TextBoxes*. Crie o *VarreXML*, conforme a **Listagem 6**.

Veja que através de um *for in*, percorremos todos os *XmlNode*s da coleção *nodes*. E caso o *node* tenha o nome *Data* estamos adicionando o seu valor no *ArrayList*. Observe que cada *XmlNode* de um *XmlDocument* pode possuir mais um conjunto de *nodes*, que podem ser acessados pela propriedade *ChildNodes*.

Listagem 6. Percorre todo o XML em busca dos Valores

```
function VarreXML(nodes: XmlNodeList;
  campos: ArrayList): ArrayList;
var
  node: XmlNode;
begin
  for node in nodes do
  begin
    if (node.Name.Trim = 'Data') then
      campos.Add(node.InnerText);
      VarreXml(node.ChildNodes, campos);
    end;
  end;
  Result := campos;
end;
```

Dessa forma, para cada *XmlNode* estamos novamente chamando o *VarreXml*, e assim varreremos todo o conteúdo do XML de forma recursiva. Ao final a função, devolverá o *ArrayList* com os valores encontrados.

Vamos fazer um teste? Salve, compile e execute o projeto. Como mostra a **Figura 8**, informe um CEP válido e clique em *Pesquisar*.

Veja que os *TextBoxes* serão preenchidos com os valores retornados pelo Web Service, bastando que o usuário preencha o campo número. Essa é uma ótima solução em telas de cadastro com endereços, imagine o tempo que ganhamos ao ter esses dados preenchidos automaticamente pelo Web Service.

Recuperando o resultado em um DataGridView

Como sabemos, existem ruas que possuem mais de um CEP, para esses casos, quando fizermos uma pesquisa avançada o Web Service retornará uma lista de todos os CEPs possíveis do endereço em questão.

Para finalizarmos o artigo, vamos fazer um último exemplo onde o usuário informará um Endereço, e os CEPs retornados serão apresentados em um *DataGridView*. Para isso, crie uma nova página e adicione componentes para que fique igual à da **Figura 9**.

Veja que temos todos os campos para a busca de CEP avançada (adicione a função *GeraXMLAvancado* nessa página e dê aos controles os nomes que utilizamos nos exemplos anteriores), e que incluímos um *DataGridView* na página.

Nesse *DataGridView* incluímos apenas uma coluna *Select* do tipo *Button Column* (acesse o editor do *DataGridView* e adicione a coluna), assim quando o usuário clicar em *Select* o CEP escolhido será

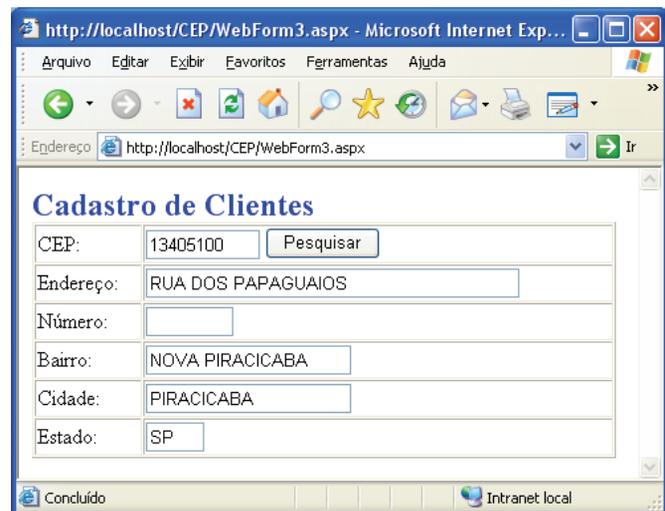


Figura 8. Testando a pesquisa de CEP tratada

mostrado no *TextBox* (mostraremos esse código a seguir).

Ainda no *DataGrid*, é preciso manter a propriedade *AutoGenerateColumns* igual a *True*. Dê um duplo clique no botão *Pesquisar* e inclua o código da **Listagem 7**.

Veja que o código é bem parecido com o do exemplo anterior, com a diferença que estamos chamando a pesquisa de CEP avançada e que ao invés de recuperarmos os dados em um *ArrayList*, estamos usando um *DataTable*.

Veja que criamos quatro colunas no *DataTable*: *CEP*, *Logradouro*, *Bairro* e *CidadeUF*. Para preencher o *DataTable* com os resultados do XML, estamos utilizando o *VarreXMLAvancado*, que é um pouco diferente do que foi utilizado no exemplo anterior, pois retorna um *DataTable*. Confira como ficou o código na **Listagem 8**.

Nesse código, estamos novamente varrendo o XML de forma recursiva, só que dessa vez procurando pelo node *Heading*. Cada resultado retornado pela pesquisa virá dentro desse node. Para finalizar, basta codificarmos o evento *SelectedIndexChanged* do *DataGrid*.



Esse evento é disparado toda vez que clicarmos no *Select* de qualquer uma das linhas. Para codificá-lo, acesse o evento e adicione o seguinte código:

```
txtCEP.Text := DataGrid1.SelectedItem.Cells[1].Text;
```

Vamos testar? Salve, compile e execute seu projeto. No formulário informe um endereço que contenha mais que um CEP, e clique em *Pesquisar*. Veja que no *DataGrid* são apresentados todos os CEPs encontrados para esse endereço, assim como mostra a **Figura 10**. Agora, basta você clicar no botão *Select* de qualquer uma das linhas que o CEP correspondente será preenchido no *TextBox*.

Conclusão

Como você pôde conferir neste artigo, é possível realizar buscas de CEP no Web Service que os Correios disponibilizam para o Office da Microsoft. Infelizmente esse Web Service está formatado para receber e retornar um XML nos padrões da ferramenta de pesquisa do Office, mas nada que nos impeça de utilizá-lo, respeitando esses padrões. Veja que as possibilidades de tratamento são muitas, basta você avaliar as suas necessidades de pesquisa e por a mão na massa. Daqui para frente é com você!

Não perca na próxima edição, onde mostraremos como criar uma consulta ao mesmo Web Service a partir de uma aplicação Delphi 7 (VCL Win32). Até lá! ■

Links

Site dos Correios

www.correios.com.br

Busca por Logradouro

Estado:	<input type="text"/>
Cidade:	<input type="text"/>
Tipo Logradouro:	<input type="text"/>
Endereço:	<input type="text"/>
Número:	<input type="text"/>
CEP:	<input type="text"/>

	Column0	Column1	Column2
<input type="button" value="Select"/>	abc	abc	abc
<input type="button" value="Select"/>	abc	abc	abc
<input type="button" value="Select"/>	abc	abc	abc
<input type="button" value="Select"/>	abc	abc	abc

Figura 9. Pesquisa de CEP em um DataGrid

Listagem 7. Evento para a busca de CEP

```
uses System.IO, System.Xml, System.Text,
    correios.Query;
...
var
    Busca: correios.Query.QueryProcessor;
    retorno: string;
    XmlDoc: XmlDocument;
    Ceps: DataTable;
begin
    try
        Busca := correios.Query.QueryProcessor.Create;
        retorno := Busca.Query(GeraXMLAvancado);
        XmlDoc := XmlDocument.Create;
        XmlDoc.LoadXml(retorno);
        Ceps := DataTable.Create;
        Ceps.Columns.Add('CEP');
        Ceps.Columns.Add('Logradouro');
        Ceps.Columns.Add('Bairro');
        Ceps.Columns.Add('CidadeUF');
        Ceps := VarreXMLAvancado(
            XmlDoc.DocumentElement.ChildNodes, Ceps);
        DataGrid1.DataSource := Ceps;
        DataGrid1.DataBind;
    except
        on E: Exception do
            Response.Write(E.ToString);
        end;
    end;
end;
```

Listagem 8. Percorre todo o XML e preenche o DataTable

```
function VarreXMLAvancado(
    nodes: XmlNodeList; Ceps: DataTable): DataTable;
var
    node: XmlNode;
    Row: DataRow;
begin
    for node in nodes do
        begin
            if (node.Name.Trim = 'Heading') then
                begin
                    Row := Ceps.NewRow;
                    Row['CEP'] := node.ChildNodes[0].InnerText;
                    Row['Logradouro'] :=
                        node.ChildNodes[1].InnerText;
                    Row['Bairro'] :=
                        node.ChildNodes[2].InnerText;
                    Row['CidadeUF'] :=
                        node.ChildNodes[3].InnerText;
                    Ceps.Rows.Add(Row);
                end;
                VarreXMLAvancado(node.ChildNodes, Ceps);
            end;
        end;
    Result := Ceps;
end;
```

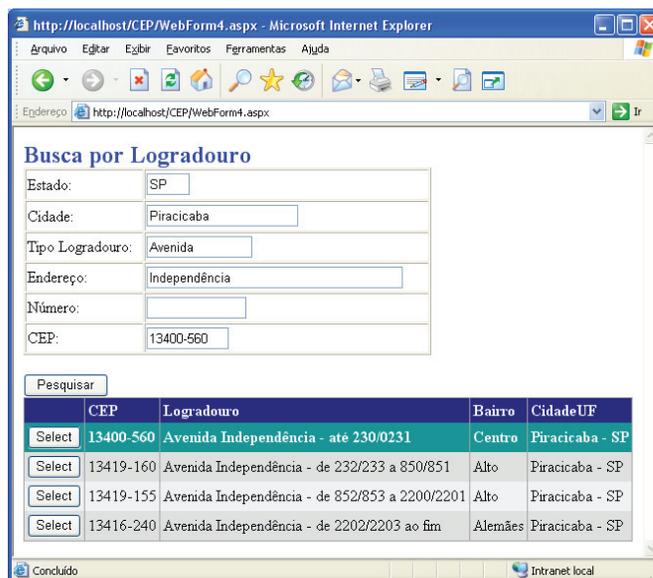


Figura 10. Teste da pesquisa de CEP em um DataGrid