

The Ultimate ASP.NET Code Sourcebook

ASP.NET Cookbook™



O'REILLY®

Michael A. Kittel & Geoffrey T. LeBlond

Dynamic Images

12.0 Introduction

The ability to draw or retrieve and display graphic images on your web pages on the fly can add powerful functionality to an application. This is a nearly impossible task in classic ASP, unless you use a third-party component of some kind. By contrast, the drawing library provided in the .NET Framework makes it relatively easy to create your own images when you need them. Indeed, it provides the ability to do almost anything you can imagine in the way of image generation. The examples shown in this chapter show you how to:

- Draw button images on the fly using text generated during the running of your application
- Create bar charts on the fly
- Display images stored in a database
- Display thumbnails from full-sized images stored in a database

These represent just a sampling of what you can do with the .NET drawing libraries and a little bit of custom code.

12.1 Drawing Button Images on the Fly

Problem

You need to create a button image on the fly using text generated during the running of your application.

Solution

Create a web form that is responsible for creating the button image using the `System.Drawing` classes and then streaming the image to the `Response` object.

In the *.aspx* file, enter an @ Page directive, but omit any head or body tags. The @ Page directive links the ASP.NET page to the code-behind class that draws the image.

In the code-behind class for the page, use the .NET language of your choice to:

1. Import the System.Drawing and System.Drawing.Imaging namespaces.
2. Create a makeButton (or similarly named) method that creates a bitmap for a button using text generated during the running of the application—for example, text passed in on the URL.
3. Create a MemoryStream object and save the bitmap in JPEG format (or other format) to the memory stream.
4. Write the resulting binary stream to Response object.

Examples 12-1 through 12-3 show the *.aspx* file and VB and C# code-behind files for an application that creates a button image whose label is provided by the application user.

To use a dynamically generated image in your application, you need to set the Src attribute of the image tags for your button bitmaps to the URL of the ASP.NET page that creates the images, passing the image text in the URL.

In the *.aspx* file for the page, add an img tag for displaying the dynamically created image.

In the code-behind class for the page that uses the image, use the .NET language of your choice to set to the Src attribute of the image tag to the URL for the web form that will draw the image, passing the text it needs in the URL.

Examples 12-4 through 12-6 show the *.aspx* file and VB and C# code-behind files for an application that uses the dynamic image generation. Figure 12-1 shows some typical output from the application.

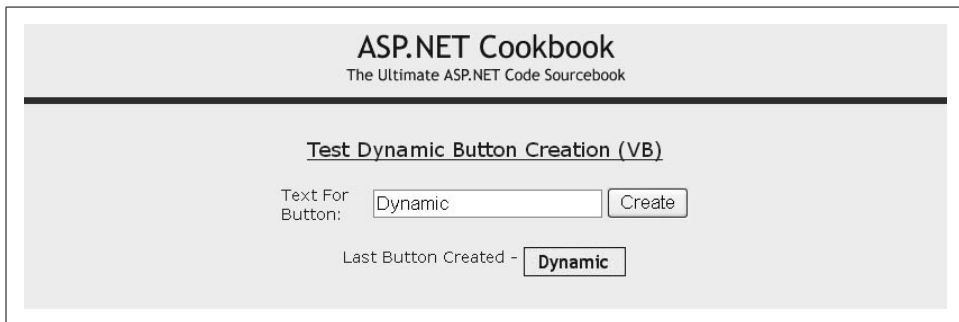


Figure 12-1. Creating a button image on the fly

Discussion

Creating button images on the fly can be handy for a couple of related reasons. First, using button images may help provide the look you want for your application. Second, generating them on the fly can avoid you having to create and save a whole

series of button images to the filesystem on the prospect that they may be needed someday. For example, you might want to use images to improve the appearance of reports.

The approach we favor for generating images on the fly involves first drawing them and then streaming the images to the Response object. How does this work? The process begins when the browser first makes a request for a page to display. During the rendering of the page, whenever the browser encounters an image tag, it sends a request for that image to the server. The browser expects the server to stream the requested image back to the browser with the content type set to indicate an image of a certain type is being returned—for example, "image/jpeg", indicating an image in JPEG format. Our approach does exactly that, but with a unique twist. Instead of a static image, which is the norm, our approach returns an image that has been created on the fly on the server. The browser neither knows nor cares where the stream is coming from, which is why our approach works just fine.

Two web forms are used in our example that illustrates this solution. The first one renders no HTML but instead processes a user request for a dynamically created button image. A second web form is used to display the requested image.

The *.aspx* file of the first form contains no head or body; it simply contains the @ Page directive to link the code-behind class for the page.

In the Page_Load event of the code-behind of the first page, the text for the image button passed in the URL is retrieved and passed to the `makeButton` method to create the button image.

The `makeButton` method first creates the font that will be used to label the button image and then measures the space that will be required to display the text. Because we have no `Graphics` object in this scenario and a `Graphics` object cannot be created by itself (it must always be associated with a specific device context), we have to create a `Bitmap` solely for the purpose of creating the `Graphics` object. Here a dummy 1-pixel-by-1-pixel bitmap is created:

```
VB      bfont = New Font("Trebuchet MS", 10)
         button = New Bitmap(1, 1, PixelFormat.Format32bppRgb)
         g = Graphics.FromImage(button)
         tSize = g.MeasureString(buttonText, bfont)
```

```
C#      bfont = new Font("Trebuchet MS", 10);
         button = new Bitmap(1, 1, PixelFormat.Format32bppRgb);
         g = Graphics.FromImage(button);
         tSize = g.MeasureString(buttonText, bfont);
```

Next, we need to calculate the size of the image required to contain the text, allowing for some space around the text. The constants `HORT_PAD` and `VERT_PAD` are used to define the space at the ends of the text and above/below the text.

```
VB      buttonWidth = CInt(Math.Ceiling(tSize.Width + (HORT_PAD * 2)))
         buttonHeight = CInt(Math.Ceiling(tSize.Height + (VERT_PAD * 2)))
```

```

C#
    buttonWidth = Convert.ToInt32(Math.Ceiling(tSize.Width +
        (HORT_PAD * 2)));
    buttonHeight = Convert.ToInt32(Math.Ceiling(tSize.Height +
        (VERT_PAD * 2)));

```

Now that we know how big to create the image, the `Bitmap` that will be used for the button image can be created. The `PixelFormat.Format32bppRgb` defines the `Bitmap` to be 32 bits per pixel, using 8 bits each for the red, green, and blue color components. For the image being created here, the format is not that important. For more graphically appealing images, however, the format plays a significant role.

```

VB
    button = New Bitmap(buttonWidth, _
        buttonHeight, _
        PixelFormat.Format32bppRgb)

```

```

C#
    button = new Bitmap(buttonWidth,
        buttonHeight,
        PixelFormat.Format32bppRgb);

```

Next, the entire image is filled with a background color. This requires creating a brush with the desired color and calling the `FillRectangle` method with the full size of the image being created. (Remember, the graphics coordinates always start at 0.) The `FromHtml` method of the `ColorTranslator` class is used to convert the HTML style color designation to the color required for the brush being created.



When filling an image with a background color, be careful of the color choices you make, since browsers do not consistently display all colors. It is best to stick to the 216 colors of the web-safe palette.

```

VB
    g = Graphics.FromImage(button)
    g.FillRectangle(New SolidBrush(ColorTranslator.FromHtml("#F0F0F0")), _
        0, _
        0, _
        buttonWidth - 1, _
        buttonHeight - 1)

```

```

C#
    g = Graphics.FromImage(button);
    g.FillRectangle(new SolidBrush(ColorTranslator.FromHtml("#F0F0F0")),
        0,
        0,
        buttonWidth - 1,
        buttonHeight - 1);

```

After filling the button image background color, a border is drawn around the perimeter of the image. This requires creating a pen with the desired color and width to do the drawing.

```

VB
    g.DrawRectangle(New Pen(Color.Navy, 1), _
        0, _
        0, _
        buttonWidth - 1, _
        buttonHeight - 1)

```

```

C#
    g.DrawRectangle(new Pen(Color.Navy, 1),
                    0,
                    0,
                    buttonWidth - 1,
                    buttonHeight - 1);

```

Finally, the text is drawn in the center of the image button. The centering is accomplished by offsetting the upper-left corner of the text block by the same amount as the spacing that was allowed around the text.

```

VB
    g.DrawString(buttonText, _
                 bfont, _
                 New SolidBrush(Color.Navy), _
                 HORT_PAD, _
                 VERT_PAD)

```

```

C#
    g.DrawString(buttonText,
                 bfont,
                 new SolidBrush(Color.Navy),
                 HORT_PAD,
                 VERT_PAD);

```

When the button image bitmap is returned to the Page_Load method, we need to create a MemoryStream object and save the bitmap in JPEG format to the memory stream. (We chose the JPEG format because it happened to work well with the images in this example. Depending on the circumstances, you may need to use another format, such as GIF.)

```

VB
    ms = New MemoryStream
    button.Save(ms, ImageFormat.Jpeg)

```

```

C#
    ms = new MemoryStream();
    button.Save(ms, ImageFormat.Jpeg);

```

The final step in generating the button image is to write the binary stream to the Response object. This requires setting the ContentType property to match the format we saved the image to in the memory stream. In this example, the image was saved in JPEG format, so the ContentType must be set to "image/jpg". This informs the browser that the stream being returned is an image of the proper type. We then use the BinaryWrite method to write the image to the Response object.

```

VB
    Response.ContentType = "image/jpg"
    Response.BinaryWrite(ms.ToArray())

```

```

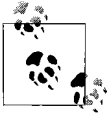
C#
    Response.ContentType = "image/jpg";
    Response.BinaryWrite(ms.ToArray());

```

Using our example web form that dynamically creates button images merely requires setting the Src attribute of the image tag to the URL for the web form just described, passing the text for the image in the URL. A sample URL is shown here:

```
src="CH12CreateButtonVB.aspx?ButtonText=Test Button"
```

In our example, the `Src` attribute of the `img` tag is set in the create button click event of the test page code-behind. To make things a little easier and avoid hardcoding page names and `QueryString` information, two constants (`PAGE_NAME` and `QS_BUTTON_TEXT`) are defined in the code-behind of the image creation page and are used here in building the URL.



Dynamically generating images can be resource intensive. To improve the performance of your application, the generated images should be cached. Recipe 13.2 provides an example of how to cache the results as a function of the data passed in the `QueryString`.

See Also

Recipe 13.2; MSDN Help for more on the .NET drawing libraries

Example 12-1. Create images dynamically (.aspx)

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="CH12CreateButtonVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH12CreateButtonVB" %>
```

Example 12-2. Create images dynamically code-behind (.vb)

```
Option Explicit On
Option Strict On
'-----
'
'   Module Name: CH12CreateButtonVB.aspx.vb
'
'   Description: This module provides the code behind for the
'               CH12CreateButtonVB.aspx page. It streams a dynamically
'               created button image to the browser.
'
'*****
Imports System
Imports System.Drawing
Imports System.Drawing.Imaging
Imports System.IO

Namespace ASPNetCookbook.VBExamples
    Public Class CH12CreateButtonVB
        Inherits System.Web.UI.Page

        'constants used to create URLs to this page
        Public Const PAGE_NAME As String = "CH12CreateButtonVB.aspx"
        Public Const QS_BUTTON_TEXT As String = "ButtonText"

        '*****
        '
        '   ROUTINE: Page_Load
        '
        '
    End Class
End Namespace
```

Example 12-2. Create images dynamically code-behind (.vb) (continued)

```
' DESCRIPTION: This routine provides the event handler for the page load
' event. It is responsible for initializing the controls
' on the page.
'-----
Private Sub Page_Load(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles MyBase.Load

    Dim buttonText As String
    Dim button As Bitmap
    Dim ms As MemoryStream

    'get button text from the query string and create the image
    buttonText = Request.QueryString(QS_BUTTON_TEXT)
    button = makeButton(buttonText)

    'write image to response object
    ms = New MemoryStream
    button.Save(ms, ImageFormat.Jpeg)
    Response.ContentType = "image/jpeg"
    Response.BinaryWrite(ms.ToArray())
End Sub 'Page_Load

'*****
'
' ROUTINE: MakeButton
'
' DESCRIPTION: This routine creates a button with the passed text.
'-----
Private Function makeButton(ByVal buttonText As String) As Bitmap
    'define the space around the text on the button
    Const HORT_PAD As Integer = 10
    Const VERT_PAD As Integer = 2

    Dim button As Bitmap
    Dim g As Graphics
    Dim bfont As Font
    Dim tSize As SizeF
    Dim buttonHeight As Integer
    Dim buttonWidth As Integer

    'create the font that will used then create a dummy button to get
    'a graphics object that provides the ability to measure the height
    'and width required to display the passed string
    bfont = New Font("Trebuchet MS", 10)
    button = New Bitmap(1, 1, PixelFormat.Format32bppRgb)
    g = Graphics.FromImage(button)
    tSize = g.MeasureString(buttonText, bfont)

    'calculate the size of button required to display the text adding
    'some space around the text
    buttonWidth = CInt(Math.Ceiling(tSize.Width + (HORT_PAD * 2)))
    buttonHeight = CInt(Math.Ceiling(tSize.Height + (VERT_PAD * 2)))
```

Example 12-2. Create images dynamically code-behind (.vb) (continued)

```
'create a new button using the calculated size
button = New Bitmap(buttonWidth, _
                    buttonHeight, _
                    PixelFormat.Format32bppRgb)

'fill the button area
g = Graphics.FromImage(button)
g.FillRectangle(New SolidBrush(ColorTranslator.FromHtml("#F0F0F0")), _
               0, _
               0, _
               buttonWidth - 1, _
               buttonHeight - 1)

'draw a rectangle around the button perimeter using a pen width of 1
g.DrawRectangle(New Pen(Color.Navy, 1), _
               0, _
               0, _
               buttonWidth - 1, _
               buttonHeight - 1)

'draw the text on the button (centered)
g.DrawString(buttonText, _
             bfont, _
             New SolidBrush(Color.Navy), _
             HORT_PAD, _
             VERT_PAD)
g.Dispose()
Return (button)
End Function 'makeButton
End Class 'CH12CreateButtonVB
End Namespace
```

Example 12-3. Create images dynamically code-behind (.cs)

```
//-----
//
//  Module Name: CH12CreateButtonCS.aspx.cs
//
//  Description: This module provides the code behind for the
//               CH12CreateButtonCS.aspx page
//
//*****
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;

namespace ASPNetCookbook.CSExamples
{
    public class CH12CreateButtonCS : System.Web.UI.Page
    {
```

Example 12-3. Create images dynamically code-behind (.cs) (continued)

```
// constants used to create URLs to this page
public const String PAGE_NAME = "CH12CreateButtonCS.aspx";
public const String QS_BUTTON_TEXT = "ButtonText";

//*****
//
// ROUTINE: Page_Load
//
// DESCRIPTION: This routine provides the event handler for the page
//              load event. It is responsible for initializing the
//              controls on the page.
//-----
private void Page_Load(object sender, System.EventArgs e)
{
    String buttonText = null;
    Bitmap button = null;
    MemoryStream ms = null;

    // get button text from the query string and create image
    buttonText = Request.QueryString[QS_BUTTON_TEXT];
    button = makeButton(buttonText);

    // write image to response object
    ms = new MemoryStream();
    button.Save(ms, ImageFormat.Jpeg);
    Response.ContentType = "image/jpeg";
    Response.BinaryWrite(ms.ToArray());
} // Page_Load

//*****
//
// ROUTINE: makeButton
//
// DESCRIPTION: This routine creates a button with the passed text.
//-----
private Bitmap makeButton(String buttonText)
{
    // define the space around the text on the button
    const int HORT_PAD = 10;
    const int VERT_PAD = 2;

    Bitmap button = null;
    Graphics g = null;
    Font bfont = null;
    SizeF tSize;
    int buttonHeight;
    int buttonWidth;

    // create the font that will used then create a dummy button to get
    // a graphics object that provides the ability to measure the height
    // and width required to display the passed string
```

Example 12-3. Create images dynamically code-behind (.cs) (continued)

```
bfont = new Font("Trebuchet MS", 10);
button = new Bitmap(1, 1, PixelFormat.Format32bppRgb);
g = Graphics.FromImage(button);
tSize = g.MeasureString(buttonText, bfont);

// calculate the size of button required to display the text adding
// some space around the text
buttonWidth = Convert.ToInt32(Math.Ceiling(tSize.Width +
    (HORT_PAD * 2)));
buttonHeight = Convert.ToInt32(Math.Ceiling(tSize.Height +
    (VERT_PAD * 2)));

// create a new button using the calculated size
button = new Bitmap(buttonWidth,
    buttonHeight,
    PixelFormat.Format32bppRgb);

// fill the button area
g = Graphics.FromImage(button);
g.FillRectangle(new SolidBrush(ColorTranslator.FromHtml("#FOFOFO")),
    0,
    0,
    buttonWidth - 1,
    buttonHeight - 1);

// draw a rectangle around the button perimeter using a pen width of 1
g.DrawRectangle(new Pen(Color.Navy, 1),
    0,
    0,
    buttonWidth - 1,
    buttonHeight - 1);

// draw the text on the button (centered)
g.DrawString(buttonText,
    bfont,
    new SolidBrush(Color.Navy),
    HORT_PAD,
    VERT_PAD);
g.Dispose();
return (button);
} // makeButton
} // CH12CreateButtonCS
}
```

Example 12-4. Using the dynamically created images (.aspx)

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="CH12TestCreateButtonVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH12TestCreateButtonVB" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Test Dynamic Button Creation</title>
```

Example 12-4. Using the dynamically created images (.aspx) (continued)

```
<link rel="stylesheet" href="css/ASPNetCookbook.css">
</head>
<body leftmargin="0" marginheight="0" marginwidth="0" topmargin="0">
  <form id="frmTestCreateButton" method="post" runat="server">
    <table width="100%" cellpadding="0" cellspacing="0" border="0">
      <tr>
        <td align="center">
          
        </td>
      </tr>
      <tr>
        <td align="center" class="dividerLine">
          </td>
        </tr>
      </table>
      <table width="90%" align="center" border="0">
        <tr>
          <td align="center">&nbsp;</td>
        </tr>
        <tr>
          <td align="center" class="PageHeading">
            Test Dynamic Button Creation (VB)
          </td>
        </tr>
        <tr>
          <td></td>
        </tr>
        <tr>
          <td align="center">
            <table width="50%">
              <tr>
                <td class="LabelText">Text For Button: </td>
                <td>
                  <asp:TextBox ID="txtButtonText" Runat="server" />
                </td>
                <td>
                  <input id="btnCreate" runat="server" type="button"
                    value="Create" name="btnCreate">
                </td>
              </tr>
              <tr>
                <td id="tdCreatedButton" runat="server"
                  colspan="3" align="center" class="LabelText"><br />
                  Last Button Created -
                  <img id="imgButton" runat="server"
                    border="0" align="middle">
                </td>
              </tr>
            </table>
          </td>
        </tr>
      </table>
    </form>
  </body>
</table>
```

Example 12-4. Using the dynamically created images (.aspx) (continued)

```
</form>
</body>
</html>
```

Example 12-5. Using the dynamically created images code-behind (.vb)

```
Option Explicit On
Option Strict On
'-----
'
'   Module Name: CH12TestCreateButtonVB.aspx.vb
'
'   Description: This module provides the code behind for the
'               CH12TestCreateButtonVB.aspx page.
'
'*****
Namespace ASPNetCookbook.VBExamples
    Public Class CH12TestCreateButtonVB
        Inherits System.Web.UI.Page

        'controls on the form
        Protected txtButtonText As System.Web.UI.WebControls.TextBox
        Protected WithEvents btnCreate As System.Web.UI.HtmlControls.HtmlInputButton
        Protected tdCreatedButton As System.Web.UI.HtmlControls.HtmlTableCell
        Protected imgButton As System.Web.UI.HtmlControls.HtmlImage

        '*****
        '
        '   ROUTINE: Page_Load
        '
        '   DESCRIPTION: This routine provides the event handler for the page load
        '               event. It is responsible for initializing the controls
        '               on the page.
        '-----
        Private Sub Page_Load(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) Handles MyBase.Load

            If (Not Page.IsPostBack) Then
                'make image button table cell invisible initially
                tdCreatedButton.Visible = False
            End If
        End Sub 'Page_Load

        '*****
        '
        '   ROUTINE: btnCreate_Click
        '
        '   DESCRIPTION: This routine provides the event handler for the create
        '               button click event. It is responsible for initializing
        '               the source property of the image button to the URL of
        '               the dynamic button creation page.
        '-----
```

Example 12-5. Using the dynamically created images code-behind (.vb) (continued)

```
Private Sub btnCreate_ServerClick(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
    Handles btnCreate.ServerClick
    'update the image tag with the URL to the page that will
    'create the button and with the button text in the URL
    imgButton.Src = CH12CreateButtonVB.PAGE_NAME & _
                  "?" & CH12CreateButtonVB.QS_BUTTON_TEXT & _
                  "=" & txtButtonText.Text
    tdCreatedButton.Visible = True
End Sub 'btnCreate_Click
End Class 'CH12TestCreateButtonVB
End Namespace
```

Example 12-6. Using the dynamically created images code-behind (.cs)

```
//-----
//
//  Module Name: CH12TestCreateButtonCS.aspx.cs
//
//  Description: This module provides the code behind for the
//               CH12TestCreateButtonCS.aspx page
//
//*****
using System;

namespace ASPNetCookbook.CSExamples
{
    public class CH12TestCreateButtonCS : System.Web.UI.Page
    {
        // controls on the form
        protected System.Web.UI.WebControls.TextBox txtButtonText ;
        protected System.Web.UI.HtmlControls.HtmlInputButton btnCreate;
        protected System.Web.UI.HtmlControls.HtmlTableCell tdCreatedButton;
        protected System.Web.UI.HtmlControls.HtmlImage imgButton;

        //*****
        //
        //  ROUTINE: Page_Load
        //
        //  DESCRIPTION: This routine provides the event handler for the page
        //               load event. It is responsible for initializing the
        //               controls on the page.
        //-----
        private void Page_Load(object sender, System.EventArgs e)
        {
            // wire the create button click event
            this.btnCreate.ServerClick +=
                new EventHandler(this.btnCreate_ServerClick);

            if (!Page.IsPostBack)
            {
                // make image button table cell invisible initially
                tdCreatedButton.Visible = false;
            }
        }
    }
}
```

Example 12-6. Using the dynamically created images code-behind (.cs) (continued)

```
    }  
} // Page_Load  
  
//*****  
//  
// ROUTINE: btnCreate_ServerClick  
//  
// DESCRIPTION: This routine provides the event handler for the create  
//               button click event. It is responsible for initializing  
//               the source property of the image button to the URL of  
//               the dynamic button creation page.  
//-----  
private void btnCreate_ServerClick(Object sender,  
                                System.EventArgs e)  
  
{  
    // update the image tag with the URL to the aspx page that will  
    // create the button and with the button text in the URL  
    imgButton.Src = CH12CreateButtonCS.PAGE_NAME +  
        "?" + CH12CreateButtonCS.QS_BUTTON_TEXT +  
        "=" + txtButtonText.Text;  
    tdCreatedButton.Visible = true;  
} // btnCreate_ServerClick  
} // CH12TestCreateButtonCS  
}
```

12.2 Creating Bar Charts on the Fly

Problem

You want to create a simple bar chart on the fly without having to resort to a commercial package to do so.

Solution

Use a combination of data binding with a Repeater control and the well-known HTML trick of stretching an image to create the bars.

In the *.aspx* file, add a Repeater control with an *ItemTemplate*.

In the code-behind class for the page, use the .NET language of your choice to:

1. Assign the data source to the Repeater control and bind it.
2. In the *ItemDataBound* event handler that is called for each item in the Repeater, set the width of the bar in the passed Repeater row.

Figure 12-2 shows some typical output. Examples 12-7 through 12-9 show the *.aspx* file and VB and C# code-behind files for an application that implements this solution.

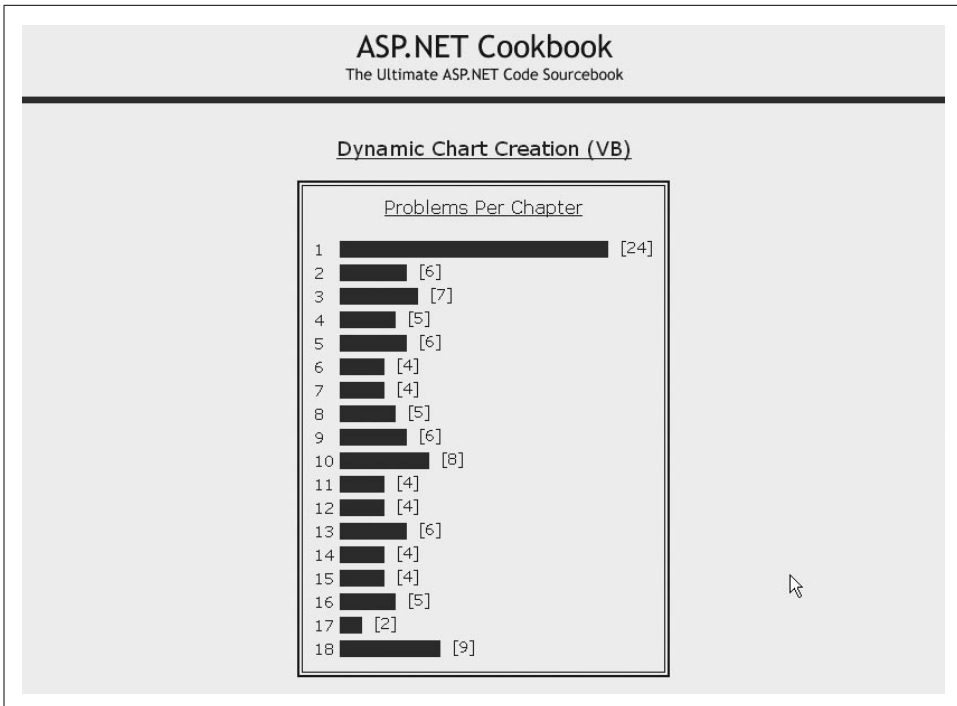


Figure 12-2. Create bar chart output dynamically

Discussion

This recipe provides a rather simple approach that combines data binding and HTML tricks to create a bar graph with very little coding, and without the need to purchase any additional components. By using more complex HTML, you can add more labels and other enhancements to this recipe, which may make it more useful for your situation.

The example we use to illustrate this solution generates a bar chart from chapter and problem data in a database. (The source of the data is not that important. Rather, it's the technique for generating the graph on the fly that is the main focus of this recipe.) The bar chart is created from an HTML table, with the top row used to label the chart.

This recipe advocates using a Repeater control to generate the rows in a table that represent the bars on the chart. The rows generated by the Repeater are defined in the `ItemTemplate` element, which in our example contains two columns. The first column is used to output the chapter number. In our example, the chapter number is obtained by binding the cell text to the `Chapter` column in the data source.

The second column contains the bar representing the number of problems in the chapter. The bar is created by using an HTML image tag with the source set to a 1-pixel-by-1-pixel image. The height and width attributes of the image “stretch” the

image to the size of the bar needed to represent the number of problems in the chapter. In our example, the height is set to a fixed value of 15 pixels, but the width is adjusted to represent the number of problems in the chapter. The width is adjusted in the code-behind and is discussed later.

The second column also contains a label to indicate the actual number of problems in the chapter. The number of problems in a chapter is obtained by binding the cell text to the `ProblemCount` column in the data source. This label is placed at the end of the bar with a nonbreaking space (` `) to separate the label from the end of the bar.

The `Page_Load` method in the example code-behind reads the data from the database and then binds the data to the `Repeater` control on the page.

The code-behind class also implements the `ItemDataBound` event handler to provide the ability to adjust the width of the image used for the bar. The `ItemDataBound` event fires once per row in the `Repeater` as the row is data bound. In this event we need to get a reference to the HTML image in the row using the `FindControl` method of the row and then set the width of the image to reflect the number of problems in the chapter represented by the row.

If this recipe does not provide the richness you need for your chart, you can create an image using the concepts presented in Recipe 12.1. The `System.Drawing` classes provide all of the functionality to create very sophisticated charts using the GDI+. They do require considerably more coding, however, as you must build your graphs from the ground up using the basic ingredients of pen, brush, point, rectangle, and the like.

See Also

Recipe 12.1; MSDN Help for more information on the `System.Drawing` class

Example 12-7. Create bar chart dynamically (.aspx.vb)

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="CH12CreateChartVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH12CreateChartVB" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>Create Chart</title>
    <link rel="stylesheet" href="css/ASPNetCookbook.css">
  </head>
  <body leftmargin="0" marginheight="0" marginwidth="0" topmargin="0">
    <form id="frmTestDynamicChart" method="post" runat="server">
      <table width="100%" cellpadding="0" cellspacing="0" border="0">
        <tr>
          <td align="center">
            
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

Example 12-7. Create bar chart dynamically (.aspx/vb) (continued)

```

        </tr>
        <tr>
            <td class="dividerLine">
                </td>
            </tr>
        </table>
        <table width="90%" align="center" border="0">
            <tr>
                <td align="center">&nbsp;</td>
            </tr>
            <tr>
                <td align="center" class="PageHeading">
                    Dynamic Chart Creation (VB)
                </td>
            </tr>
            <tr>
                <td></td>
            </tr>
            <tr>
                <td align="center">
                    <table border="2" bordercolor="#000080" cellpadding="10">
                        <tr>
                            <td>
                                <table cellpadding="0" cellspacing="0" border="0">
                                    <tr>
                                        <td align="center" colspan="2" class="SubHeading">
                                            Problems Per Chapter<br /><br /></td>
                                    </tr>
                                </table>
                            </td>
                        </tr>
                    </table>
                </td>
            </tr>
        <asp:Repeater ID="repChartBar" Runat="server">
            <ItemTemplate>
                <tr>
                    <td class="LabelText">
                        <%=# DataBinder.Eval(Container.DataItem, "Chapter") %>&nbsp;&nbsp;&nbsp;
                    </td>
                    <td class="LabelText">
                        
                            &nbsp;&nbsp;&nbsp;[<%=# DataBinder.Eval(Container.DataItem, "ProblemCount") %>]
                    </td>
                </tr>
            </ItemTemplate>
        </asp:Repeater>
    </table>
    </td>
</tr>
</table>
</td>
</tr>
</table>
</form>
</body>
</html>

```

Example 12-8. Create bar chart dynamically code-behind (.vb)

```
Option Explicit On
Option Strict On
'-----
'
'   Module Name: CH12CreateChartVB.aspx.vb
'
'   Description: This module provides the code behind for the
'               CH12CreateChartVB.aspx page.
'
'*****
Imports Microsoft.VisualBasic
Imports System
Imports System.Configuration
Imports System.Data
Imports System.Data.Common
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
    Public Class CH12CreateChartVB
        Inherits System.Web.UI.Page

        'controls on the form
        Protected WithEvents repChartBar As System.Web.UI.WebControls.Repeater

        '*****
        '
        '   ROUTINE: Page_Load
        '
        '   DESCRIPTION: This routine provides the event handler for the page load
        '               event. It is responsible for initializing the controls
        '               on the page.
        '-----
        Private Sub Page_Load(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) Handles MyBase.Load
            Dim dbConn As OleDbConnection
            Dim dc As OleDbCommand
            Dim dr As OleDbDataReader
            Dim strConnection As String
            Dim strSQL As String

            If (Not Page.IsPostBack) Then
                Try
                    'get the connection string from web.config and open a connection
                    'to the database
                    strConnection = _
                        ConfigurationSettings.AppSettings("dbConnectionString")
                    dbConn = New OleDb.OleDbConnection(strConnection)
                    dbConn.Open()

                    'build the query string and get the data from the database
                    strSQL = "SELECT DISTINCT ChapterID AS Chapter, " & _
                        "count(*) AS ProblemCount " & _
                        "FROM Problem " & _
```

Example 12-8. Create bar chart dynamically code-behind (.vb) (continued)

```
        "GROUP BY ChapterID"

        dc = New OleDbCommand(strSQL, dbConn)
        dr = dc.ExecuteReader()

        'set the source of the data for the repeater control and bind it
        repChartBar.DataSource = dr
        repChartBar.DataBind()

    Finally
        'cleanup
        If (Not IsNothing(dbConn)) Then
            dbConn.Close()
        End If
    End Try
End If
End Sub

'*****
',
',
', ROUTINE: repChartBar_ItemDataBound
',
', DESCRIPTION: This routine is the event handler that is called for each
',
', item in the datagrid after a data bind occurs. It is
',
', responsible for setting the width of the bar in the
',
', passed repeater row to reflect the number of problems in
',
', the chapter the row represents
',-----
Private Sub repChartBar_ItemDataBound(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.RepeaterItemEventArgs) _
    Handles repChartBar.ItemDataBound
    Dim img As System.Web.UI.HtmlControls.HtmlImage

    'get a reference to the image used for the bar in the row
    img = CType(e.Item.FindControl("imgChartBar"), _
        System.Web.UI.HtmlControls.HtmlImage)

    'set the width to the number of problems in the chapter for this row
    'multiplied by a constant to stretch the bar a bit more
    img.Width = _
        Cint(CType(e.Item.DataItem, DbDataRecord)("ProblemCount")) * 10
End Sub 'repChartBar_ItemDataBound
End Class 'CH12CreateChartVB
End Namespace
```

Example 12-9. Create bar chart dynamically code-behind (.cs)

```
//-----
//
// Module Name: CH12CreateChartCS.aspx.cs
//
// Description: This module provides the code behind for the
//              CH12CreateChartCS.aspx page
```

Example 12-9. Create bar chart dynamically code-behind (.cs) (continued)

```
//
//*****
using System;
using System.Configuration;
using System.Data;
using System.Data.Common;
using System.Data.OleDb;
using System.Web.UI.WebControls;

namespace ASPNetCookbook.CSExamples
{
    public class CH12CreateChartCS : System.Web.UI.Page
    {
        // controls on the form
        protected System.Web.UI.WebControls.Repeater repChartBar;

        //*****
        //
        // ROUTINE: Page_Load
        //
        // DESCRIPTION: This routine provides the event handler for the page
        //                load event. It is responsible for initializing the
        //                controls on the page.
        //-----
        private void Page_Load(object sender, System.EventArgs e)
        {
            OleDbConnection dbConn = null;
            OleDbCommand dc = null;
            OleDbDataReader dr = null;
            string strConnection = null;
            String strSQL = null;

            // bind the item data bound event
            this.repChartBar.ItemDataBound +=
                new RepeaterItemEventHandler(this.repChartBar_ItemDataBound);

            if (!Page.IsPostBack)
            {
                try
                {
                    // get the connection string from web.config and open a connection
                    // to the database
                    strConnection =
                        ConfigurationSettings.AppSettings["dbConnectionString"];
                    dbConn = new OleDbConnection(strConnection);
                    dbConn.Open();

                    // build the query string and get the data from the database
                    strSQL = "SELECT DISTINCT ChapterID AS Chapter, " +
                        "count(*) AS ProblemCount " +
                        "FROM Problem " +
                        "GROUP BY ChapterID";
                    dc = new OleDbCommand(strSQL, dbConn);
                }
            }
        }
    }
}
```

Example 12-9. Create bar chart dynamically code-behind (.cs) (continued)

```
        dr = dc.ExecuteReader();

        // set the source of the data for the repeater control and bind it
        repChartBar.DataSource = dr;
        repChartBar.DataBind();
    }

    finally
    {
        // clean up
        if (dbConn != null)
        {
            dbConn.Close();
        }
    }
} // Page_Load

//*****
//
// ROUTINE: repChartBar_ItemDataBound
//
// DESCRIPTION: This routine is the event handler that is called for
//               each item in the datagrid after a data bind occurs. It
//               is responsible for setting the width of the bar in the
//               passed repeater row to reflect the number of problems
//               in the chapter the row represents.
//-----
private void repChartBar_ItemDataBound(Object sender,
                                       System.Web.UI.WebControls.RepeaterItemEventArgs e)
{
    System.Web.UI.HtmlControls.HtmlImage img = null;

    // get a reference to the image used for the bar in the row
    img = (System.Web.UI.HtmlControls.HtmlImage)
        (e.Item.FindControl("imgChartBar"));

    // set the width to the number of problems in the chapter for this row
    // multiplied by a constant to stretch the bar a bit more
    img.Width =
        (int)((DbDataRecord)(e.Item.DataItem)["ProblemCount"]) * 10;
} // repChartBar_ItemDataBound
} // CH12CreateChartCS
}
```

12.3 Displaying Images Stored in a Database

Problem

Your application stores images in a database that you want to display on a web form.

Solution

Create a web form that reads the image data from the database and streams the image to the Response object.

In the *.aspx* file, enter an @ Page directive—omitting any head or body tags to link the *.aspx* page to the code-behind class that retrieves and displays the images.

1. Read the image ID that is generated by the running application—for example, the image ID passed in the URL for accessing the web form.
2. Open a connection to the database that contains the images.
3. Build a query string, and read the byte array of the desired image from the database.
4. Set the content type for the image and write it to the Response object.

Examples 12-10 through 12-12 show the *.aspx* file and VB and C# code-behind files for an application that implements the image-building portion of the solution.

To use this dynamic image generation technique in your application, set the `src` attribute of the image tags used to display the images to the URL of the ASP.NET page that reads the images from the database, passing the image ID in the URL.

In the *.aspx* file for the page, add an `img` tag for displaying the image.

In the code-behind class for the page that uses the image, use the .NET language of your choice to set the `src` attribute of the image tag to the URL for the web form just described, passing the ID of the image in the URL.

Examples 12-13 through 12-15 show the *.aspx* file and VB and C# code-behind files for an application that uses the dynamic image generation. Figure 12-3 shows some typical output from the application.

Discussion

If you have images in a database that you want to display on a web form, chances are you've considered using an image tag to display them. Nevertheless, you may be searching for a practical way to set the image tag's `src` attribute and move the image data to the browser while maintaining the maximum flexibility in selecting the images you need.

The solution we favor that meets these requirements involves creating a web form that reads an image from a database and streams it to the Response object. A convenient way to specify the image to read from the database is to include an image ID in the URL used to call the web form that retrieves and returns the image to the browser.

Our example that illustrates this solution consists of two web forms. The first web form renders no HTML but instead processes the request for reading an image from the database. The second web form is used to demonstrate displaying an image from the database.

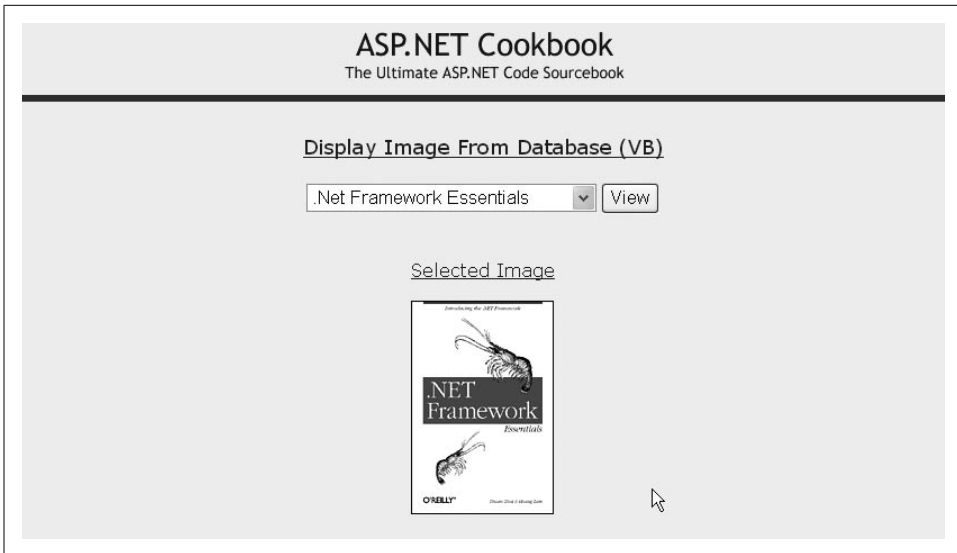


Figure 12-3. Displaying images from a database

The `.aspx` file of the first form contains no head or body; it simply contains the `@ Page` directive to link the page to its code-behind class.

The `Page_Load` method of the code-behind performs the following steps to retrieve the requested image from the database and then send it to the browser:

1. Retrieve the ID of the requested image from the URL
2. Read the byte array of the image from the database
3. Set the `ContentType` of the type of image stored in the database (GIF in this example)
4. Write the byte array of the image to the `Response` object

To use the ASP.NET page to retrieve images from the database, we need to set the `src` attribute of an `img` tag to the name of the page just described, passing the ID of the desired image in the URL. A sample URL is shown here:

```
src="CH12ImageFromDatabaseVB.aspx?ImageID=5"
```

In our example, the `src` attribute of an `img` tag is set in the view image button click event of the test page code-behind. To make things a little easier and avoid hardcoding page names and `queryString` information, two constants (`PAGE_NAME` and `QS_IMAGE_ID`) are defined in the code-behind of the page that reads the images from the database and are used here in building the URL.

The performance of this solution can be significantly improved by caching the images retrieved from the database instead of retrieving them for each request. Refer to Recipe 13.2 for an example of how to cache the results as a function of the data passed in the `QueryString`.

An `HttpHandler` can be used to implement the same functionality described in this recipe. Refer to Recipe 17.1 for an example of retrieving images from a database using an `HttpHandler`.

Images can be stored in a database using the technique described in Recipe 15.4.

See Also

Recipe 13.2; Recipe 15.4; Recipe 17.1

Example 12-10. Reading images from a database (.aspx)

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="CH12ImageFromDatabaseVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH12ImageFromDatabaseVB" %>
```

Example 12-11. Reading images from a database code-behind (.vb)

```
Option Explicit On
Option Strict On
'-----
'
'   Module Name: CH12ImageFromDatabaseVB.aspx.vb
'
'   Description: This module provides the code behind for the
'               CH12ImageFromDatabaseVB.aspx page.
'
'*****
Imports Microsoft.VisualBasic
Imports System
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb
Imports System.IO

Namespace ASPNetCookbook.VBExamples
    Public Class CH12ImageFromDatabaseVB
        Inherits System.Web.UI.Page

        'constants used to create URLs to this page
        Public Const PAGE_NAME As String = "CH12ImageFromDatabaseVB.aspx"
        Public Const QS_IMAGE_ID As String = "ImageID"

        '*****
        '
        '   ROUTINE: Page_Load
        '
        '   DESCRIPTION: This routine provides the event handler for the page load
        '               event. It is responsible for initializing the controls
        '               on the page.
        '-----
        Private Sub Page_Load(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) Handles MyBase.Load
```

Example 12-11. Reading images from a database code-behind (.vb) (continued)

```
Dim dbConn As OleDbConnection
Dim dCmd As OleDbCommand
Dim imageData() As Byte
Dim strConnection As String
Dim strSQL As String
Dim imageID As String

If (Not Page.IsPostBack) Then
    Try
        'get the ID of the image to retrieve from the database
        imageID = Request.QueryString(QS_IMAGE_ID)

        'get the connection string from web.config and open a connection
        'to the database
        strConnection = _
            ConfigurationSettings.AppSettings("dbConnectionString")
        dbConn = New OleDb.OleDbConnection(strConnection)
        dbConn.Open()

        'build the query string and get the data from the database
        strSQL = "SELECT ImageData " & _
            "FROM BookImage " & _
            "WHERE BookImageID=?"
        dCmd = New OleDbCommand(strSQL, dbConn)
        dCmd.Parameters.Add(New OleDbParameter("BookImageID", imageID))
        imageData = CType(dCmd.ExecuteScalar(), Byte())

        'set the content type for the image and write it to the response
        Response.ContentType = "image/gif"
        Response.BinaryWrite(imageData)

    Finally
        'clean up
        If (Not IsNothing(dbConn)) Then
            dbConn.Close()
        End If
    End Try
End If
End Sub 'Page_Load
End Class 'CH12ImageFromDatabaseVB
End Namespace
```

Example 12-12. Reading images from a database code-behind (.cs)

```
//-----
//
//  Module Name: CH12ImageFromDatabaseCS.aspx.cs
//
//  Description: This module provides the code behind for the
//               CH12ImageFromDatabaseCS.aspx page
//
//*****
```

Example 12-12. Reading images from a database code-behind (.cs) (continued)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;

namespace ASPNetCookbook.CSExamples
{
    public class CH12ImageFromDatabaseCS : System.Web.UI.Page
    {
        // constants used to create URLs to this page
        public const String PAGE_NAME = "CH12ImageFromDatabaseCS.aspx";
        public const String QS_IMAGE_ID = "ImageID";

        //*****
        //
        // ROUTINE: Page_Load
        //
        // DESCRIPTION: This routine provides the event handler for the page
        //                load event. It is responsible for initializing the
        //                controls on the page.
        //-----
        private void Page_Load(object sender, System.EventArgs e)
        {
            OleDbConnection dbConn = null;
            OleDbCommand dCmd = null;
            byte[] imageData = null;
            String strConnection = null;
            String strSQL = null;
            String imageID = null;

            if (!Page.IsPostBack)
            {
                try
                {
                    // get the ID of the image to retrieve from the database
                    imageID = Request.QueryString[QS_IMAGE_ID];

                    // get the connection string from web.config and open a connection
                    // to the database
                    strConnection =
                        ConfigurationSettings.AppSettings["dbConnectionString"];
                    dbConn = new OleDbConnection(strConnection);
                    dbConn.Open();

                    // build the query string and get the data from the database
                    strSQL = "SELECT ImageData " +
                        "FROM BookImage " +
                        "WHERE BookImageID=?";
                    dCmd = new OleDbCommand(strSQL, dbConn);
                    dCmd.Parameters.Add(new OleDbParameter("BookImageID", imageID));
                    imageData = (byte[])dCmd.ExecuteScalar();
                }
            }
        }
    }
}
```

Example 12-12. Reading images from a database code-behind (.cs) (continued)

```
        // set the content type for the image and write it to the response
        Response.ContentType = "image/gif";
        Response.BinaryWrite(imageData);
    }

    finally
    {
        // clean up
        if (dbConn != null)
        {
            dbConn.Close();
        }
    }
} // Page_Load
} // CH12ImageFromDatabaseCS
}
```

Example 12-13. Displaying images from a database (.aspx)

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="CH12TestImageFromDatabaseVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH12TestImageFromDatabaseVB" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>Test Image From Database</title>
    <link rel="stylesheet" href="css/ASPNetCookbook.css">
  </head>
  <body leftmargin="0" marginheight="0" marginwidth="0" topmargin="0">
    <form id="frmTestImageDB" method="post" runat="server">
      <table width="100%" cellpadding="0" cellspacing="0" border="0">
        <tr>
          <td align="center">
            
          </td>
        </tr>
        <tr>
          <td class="dividerLine">
            </td>
          </tr>
      </table>
      <table width="90%" align="center" border="0">
        <tr>
          <td align="center">&nbsp;</td>
        </tr>
        <tr>
          <td align="center" class="PageHeading">
            Display Image From Database (VB)
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

Example 12-13. Displaying images from a database (.aspx) (continued)

```
<td></td>
</tr>
<tr>
  <td align="center">
    <table>
      <tr>
        <td>
          <asp:DropDownList ID="ddImages" Runat="server" />
        </td>
        <td>
          <input id="btnViewImage" runat="server" type="button"
            value="View">
        </td>
      </tr>
      <tr>
        <td id="tdSelectedImage" runat="server"
          colspan="2" align="center" class="SubHeading">
          <br /><br />
          Selected Image<br /><br />
          <img id="imgBook" runat="server" border="0">
        </td>
      </tr>
    </table>
  </td>
</tr>
</table>
</form>
</body>
</html>
```

Example 12-14. Displaying images from a database code-behind (.vb)

```
Option Explicit On
Option Strict On
'-----
'
'   Module Name: CH12TestImageFromDatabaseVB.aspx.vb
'
'   Description: This module provides the code behind for the
'               CH12TestImageFromDatabaseVB.aspx page.
'
'*****
Imports Microsoft.VisualBasic
Imports System
Imports System.Configuration
Imports System.Data
Imports System.Data.OleDb

Namespace ASPNetCookbook.VBExamples
  Public Class CH12TestImageFromDatabaseVB
    Inherits System.Web.UI.Page
```

Example 12-14. Displaying images from a database code-behind (.vb) (continued)

```
'controls on the form
Protected ddImages As System.Web.UI.WebControls.DropDownList
Protected WithEvents btnViewImage As System.Web.UI.HtmlControls.HtmlInputButton
Protected imgBook As System.Web.UI.HtmlControls.HtmlImage
Protected tdSelectedImage As System.Web.UI.HtmlControls.HtmlTableCell

'*****
'
' ROUTINE: Page_Load
'
' DESCRIPTION: This routine provides the event handler for the page load
'               event. It is responsible for initializing the controls
'               on the page.
'-----
Private Sub Page_Load(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles MyBase.Load
    Dim dbConn As OleDbConnection
    Dim dc As OleDbCommand
    Dim dr As OleDbDataReader
    Dim strConnection As String
    Dim strSQL As String

    If (Not Page.IsPostBack) Then
        'initially hide the selected image since one is not selected
        tdSelectedImage.Visible = False

    Try
        'get the connection string from web.config and open a connection
        'to the database
        strConnection = _
            ConfigurationSettings.AppSettings("dbConnectionString")
        dbConn = New OleDb.OleDbConnection(strConnection)
        dbConn.Open()

        'build the query string and get the data from the database
        strSQL = "SELECT BookImageID, Title " & _
            "FROM BookImage"
        dc = New OleDbCommand(strSQL, dbConn)
        dr = dc.ExecuteReader()

        'set the source of the data for the repeater control and bind it
        ddImages.DataSource = dr
        ddImages.DataTextField = "Title"
        ddImages.DataValueField = "BookImageID"
        ddImages.DataBind()

    Finally
        'clean up
        If (Not IsNothing(dbConn)) Then
            dbConn.Close()
        End If
    End Try
End Sub
```

Example 12-14. Displaying images from a database code-behind (.vb) (continued)

```
End If
End Sub 'Page_Load

'*****
'
' ROUTINE: btnViewImage_ServerClick
'
' DESCRIPTION: This routine provides the event handler for the view
'              image click event. It is responsible for setting the
'              src attribute of the imgBook tag to the page that will
'              retrieve the image data from the database and stream
'              it to the browser.
'-----
Private Sub btnViewImage_ServerClick(ByVal sender As Object, _
                                     ByVal e As System.EventArgs) _
    Handles btnViewImage.ServerClick
    'set the source for the selected image tag
    imgBook.Src = CH12ImageFromDatabaseVB.PAGE_NAME & "?" & _
                 CH12ImageFromDatabaseVB.QS_IMAGE_ID & "=" & _
                 ddImages.SelectedItem.Value.ToString()

    'make the selected image visible
    tdSelectedImage.Visible = True
End Sub 'btnViewImage_ServerClick
End Class 'CH12TestImageFromDatabaseVB
End Namespace
```

Example 12-15. Displaying images from a database code-behind (.cs)

```
//-----
//
// Module Name: CH12TestImageFromDatabaseCS.aspx.cs
//
// Description: This module provides the code behind for the
//              CH12TestImageFromDatabaseCS.aspx page
//
//*****
using System;
using System.Configuration;
using System.Data;
using System.Data.OleDb;

namespace ASPNetCookbook.CSExamples
{
    public class CH12TestImageFromDatabaseCS : System.Web.UI.Page
    {
        // controls on the form
        protected System.Web.UI.WebControls.DropDownList ddImages;
        protected System.Web.UI.HtmlControls.HtmlInputButton btnViewImage;
        protected System.Web.UI.HtmlControls.HtmlImage imgBook;
        protected System.Web.UI.HtmlControls.HtmlTableCell tdSelectedImage;
```

Example 12-15. Displaying images from a database code-behind (.cs) (continued)

```

/*****
//
// ROUTINE: Page_Load
//
// DESCRIPTION: This routine provides the event handler for the page
//               load event. It is responsible for initializing the
//               controls on the page.
//-----
private void Page_Load(object sender, System.EventArgs e)
{
    OleDbConnection dbConn = null;
    OleDbCommand dc = null;
    OleDbDataReader dr = null;
    String strConnection = null;
    String strSQL = null;

    // wire the view button click event
    this.btnViewImage.ServerClick +=
        new EventHandler(this.btnViewImage_ServerClick);

    if (!Page.IsPostBack)
    {
        // initially hide the selected image since one is not selected
        tdSelectedImage.Visible = false;

        try
        {
            // get the connection string from web.config and open a connection
            // to the database
            strConnection =
                ConfigurationSettings.AppSettings["dbConnectionString"];
            dbConn = new OleDbConnection(strConnection);
            dbConn.Open();

            // build the query string and get the data from the database
            strSQL = "SELECT BookImageID, Title " +
                "FROM BookImage";
            dc = new OleDbCommand(strSQL, dbConn);
            dr = dc.ExecuteReader();

            // set the source of the data for the repeater control and bind it
            ddImages.DataSource = dr;
            ddImages.DataTextField = "Title";
            ddImages.DataValueField = "BookImageID";
            ddImages.DataBind();
        }

        finally
        {
            // clean up
            if (dbConn != null)
            {

```

Example 12-15. Displaying images from a database code-behind (.cs) (continued)

```
        dbConn.Close();
    }
}
} // Page_Load

//*****
//
// ROUTINE: btnViewImage_ServerClick
//
// DESCRIPTION: This routine provides the event handler for the view
//              image click event. It is responsible for setting the
//              src attribute of the imgBook tag to the page that will
//              retrieve the image data from the database and stream
//              it to the browser.
//-----
private void btnViewImage_ServerClick(Object sender,
                                     System.EventArgs e)
{
    // set the source for the selected image tag
    imgBook.Src = CH12ImageFromDatabaseCS.PAGE_NAME + "?" +
                 CH12ImageFromDatabaseCS.QS_IMAGE_ID + "=" +
                 ddImages.SelectedItem.Value.ToString();

    // make the selected image visible
    tdSelectedImage.Visible = true;
} // btnViewImage_ServerClick
} // CH12TestImageFromDatabaseCS
}
```

12.4 Displaying Thumbnail Images

Problem

You want to be able to display a page of images stored in your database in thumbnail format.

Solution

Implement the first of the two ASP.NET pages described in Recipe 12.3, changing the `Page_Load` method in the code-behind class to scale the full-sized image retrieved from the database to the appropriate size for a thumbnail presentation.

In the `Page_Load` method of the code-behind class for the page, use the .NET language of your choice to:

1. Create a `System.Drawing.Image` object from the byte array retrieved from the database.

2. Use a constant to define the height of the thumbnail and then calculate the width to maintain the aspect ratio of the image.
3. Use the `GetThumbnailImage` method of the `Image` object to scale the image to the desired size.
4. Load the thumbnail image into a `MemoryStream` and then write it to the `Response` object.

Examples 12-16 and 12-17 show the VB and C# code-behind class for our example that illustrates this solution. (See the `CH12ImageFromDatabaseVB.aspx` file and VB and C# code-behind files in Recipe 12.3 for our starting point.)

To display the thumbnails, create another ASP.NET page, add a `DataList` control with image tags in the `ItemTemplate`, and then use data binding to set the `src` attributes of the image tags.

In the `.aspx` file for the page:

1. Use a `DataList` control to provide the ability to generate a list using data binding.
2. Use a `HeaderTemplate` to label the table of images.
3. Use an `ItemTemplate` to define an image that is displayed in the `DataList`.

In the code-behind class for the page, use the .NET language of your choice to:

1. Open a connection to the database.
2. Build a query string, and read the list of images from the database.
3. Assign the data source to the `DataList` control and bind it.
4. Set the `src` attribute of the image tags used to display the thumbnail images in the `ItemDataBound` event of the `DataList`.

Examples 12-18 through 12-20 show the `.aspx` file and VB and C# code-behind files for the application that uses the dynamically generated images. The output produced by the page is shown in Figure 12-4.

Discussion

The rationale for this recipe is similar to that of Recipe 12.3. That is, you need a convenient way to display images from a database, in this case a page of thumbnail images, and it must efficiently move the image data to the browser while maintaining the maximum flexibility in selecting images from the data store.

This recipe uses the same approach as in Recipe 12.3 where, with one page, an image is retrieved from the database and streamed to the browser, and a second page is used to generate the image requests and display the results, which in this case is a set of thumbnails. Additionally, the image retrieval page must scale the images to thumbnail size.

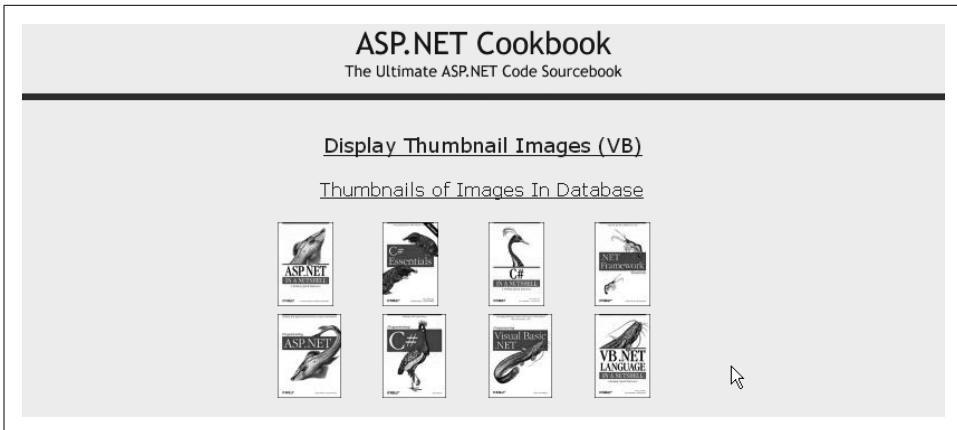


Figure 12-4. Display thumbnails output

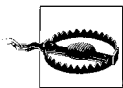
In our example that demonstrates the solution, the `Page_Load` method of the code-behind for the image building page (the `CH12ThumbnailFromDatabase` page) is modified to scale the full-sized image retrieved from the database to the appropriate size for a thumbnail presentation.

The first step to scale the image is to create a `System.Drawing.Image` object from the byte array retrieved from the database. This requires loading the byte array into a `MemoryStream` and then using the `FromStream` method of the `Image` class to create the image.

Next, we need to calculate how much to reduce the image. A constant is used to define the height of the thumbnail, and the width is calculated by determining how much the height is being reduced and multiplying the value times the width of the full-size image.



It is important to reduce the height and width by the same scale factor to keep the aspect ratio correct. If the height or width is reduced by a different amount, the image will be distorted.



Handling mixed calculations of integers and doubles can result in unexpected results. Visual Basic is more tolerant and will allow the division of two integers with the quotient set to a variable of type `double` and result in the correct value. `C#` will return an integer result from the division of two integers, effectively truncating the result. It is best to cast at least the numerator to the type of the resultant variable.

Now that the width and height of the thumbnail are determined, the `GetThumbnailImage` method of the full-size image can be used to return a scaled-down image to use as the thumbnail.

Once the thumbnail image is created it can be loaded into a `MemoryStream` and then written to the `Response` object in the same manner described in Recipe 12.1.

The web form used to display the thumbnails uses a `DataList` control to provide the ability to generate the list using data binding. The `DataList` is configured to display four columns horizontally by setting the `RepeatColumns` attribute to "4" and the `RepeatDirection` attribute to "Horizontal". This will start a new row in the table used to display the images after every fourth image.

A `HeaderTemplate` is used to label the table of images. The template can contain any HTML that can be properly displayed in a table. In this example, the template consists of a single table row containing a single cell with the heading for the table. The `colspan` attribute is set to "4" to cause the cell to span across all columns in the table, and the `align` attribute is set to "center" to center the heading.

An `ItemTemplate` is used to define an item that is displayed in the `DataList`. In this example, the template consists of an `img` tag that simply has the `ID` and `Runat` attributes set to provide access to the item from the code-behind.

The `Page_Load` method in the code-behind reads the list of images from the database and binds them to the `DataList` control. This is accomplished by opening a connection to the database, querying for a list of the image IDs, then setting the `DataSource` and calling the `dataBind` method of the `DataList`. The only data we need from the database is the list of IDs for the images. These will be used to set the image sources, and the actual reading of the image data will be done by the code described earlier.

The `src` attribute of the image tags used to display the thumbnail images is set in the `ItemDataBound` event of the `DataList`. The `ItemDataBound` event is called for every item in the `DataList`, including the header. Therefore, it is important to check the item type, since image tags only appear in the data items. Data items can be an `Item` or an `AlternatingItem`.

If the item is actually a data item, we need to get a reference to the image control in the item. This is accomplished by using the `FindControl` method of the item to locate the image control using the ID assigned in the `.aspx` file. This reference must be cast to an `HtmlImage` type since the return type of `FindControl` is an `Object`.

Once a reference to the image is obtained, the `src` attribute can be set to the name of the ASP.NET page that is used to generate the thumbnail image passing the ID of the image in the URL. As with Recipe 12.3, constants are used for the name of the page and the name of the item passed in the URL.

The ID of the image is obtained from the `DataItem` method of the item. This must be cast to a `DbDataRecord` type to allow "looking up" the ID of the image using the name of the column included in the SQL query used in the `bindData` method described earlier.

This example presents a useful method of displaying thumbnails of images stored in a database. When used with reasonably small images, the performance is acceptable

for most applications. If the images are very large, however, you may want to create the thumbnail images offline and store them in the database to avoid the performance hit for real-time conversion.

See Also

Recipe 12.1; Recipe 12.3

Example 12-16. Page_Load method for generating thumbnail image (.vb)

```
Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    'height of the thumbnail created from the original image
    Const THUMBNAIL_HEIGHT As Integer = 75

    Dim dbConn As OleDbConnection
    Dim dc As OleDbCommand
    Dim imageData() As Byte
    Dim strConnection As String
    Dim strSQL As String
    Dim ms As MemoryStream
    Dim fullsizeImage As image
    Dim thumbnailImage As image
    Dim thumbnailWidth As Integer
    Dim imageID As String

    If (Not Page.IsPostBack) Then
        Try
            'get the ID of the image to retrieve from the database
            imageID = Request.QueryString(QS_IMAGE_ID)

            'get the connection string from web.config and open a connection
            'to the database
            strConnection = _
                ConfigurationSettings.AppSettings("dbConnectionString")
            dbConn = New OleDb.OleDbConnection(strConnection)
            dbConn.Open()

            'build the query string and get the data from the database
            strSQL = "SELECT ImageData " & _
                "FROM BookImage " & _
                "WHERE BookImageID=" & imageID
            dc = New OleDbCommand(strSQL, dbConn)
            imageData = CType(dc.ExecuteScalar(), Byte())

            'create an image from the byte array
            ms = New MemoryStream(imageData)
            fullsizeImage = System.Drawing.Image.FromStream(ms)

            'calculate the amount to shrink the height and width
            thumbnailWidth = _
                CInt(Math.Round((Cdbl(THUMBNAIL_HEIGHT) / _
                    fullsizeImage.Height) * fullsizeImage.Width))
        
```

Example 12-16. *Page_Load* method for generating thumbnail image (.vb) (continued)

```
'create the thumbnail image
thumbnailImage = fullsizeImage.GetThumbnailImage(thumbnailWidth, _
                                                THUMBNAI_HEIGHT, _
                                                Nothing, _
                                                IntPtr.Zero)

'write thumbnail to the response object
ms = New MemoryStream
thumbnailImage.Save(ms, ImageFormat.Jpeg)
Response.ContentType = "image/jpeg"
Response.BinaryWrite(ms.ToArray())

Finally
'clean up
If (Not IsNothing(dbConn)) Then
    dbConn.Close()
End If
End Try
End If
End Sub 'Page_Load
```

Example 12-17. *getImage* method for generating thumbnail image (.cs)

```
private void Page_Load(object sender, System.EventArgs e)
{
    // height of the thumbnail created from the original image
    const int THUMBNAI_HEIGHT = 75;

    OleDbConnection dbConn = null;
    OleDbCommand dc = null;
    byte[] imageData = null;
    String strConnection = null;
    String strSQL = null;
    MemoryStream ms = null;
    System.Drawing.Image fullsizeImage = null;
    System.Drawing.Image thumbnailImage = null;
    int thumbnailWidth;
    String imageID = null;

    if (!Page.IsPostBack)
    {
        try
        {
            // get the ID of the image to retrieve from the database
            imageID = Request.QueryString[QS_IMAGE_ID];

            // get the connection string from web.config and open a connection
            // to the database
            strConnection =
                ConfigurationSettings.AppSettings["dbConnectionString"];
            dbConn = new OleDbConnection(strConnection);
            dbConn.Open();
```

Example 12-17. *getImage* method for generating thumbnail image (.cs) (continued)

```
// build the query string and get the data from the database
strSQL = "SELECT ImageData " +
        "FROM BookImage " +
        "WHERE BookImageID=" + imageID;
dc = new OleDbCommand(strSQL, dbConn);
imageData = (byte[])(dc.ExecuteScalar());

// create an image from the byte array
ms = new MemoryStream(imageData);
fullsizeImage = System.Drawing.Image.FromStream(ms);

// calculate the amount to shink the height and width
thumbnailWidth =
    Convert.ToInt32(Math.Round((Convert.ToDouble(THUMBNAIL_HEIGHT) /
        fullsizeImage.Height) * fullsizeImage.Width));

// create the thumbnail image
thumbnailImage = fullsizeImage.GetThumbnailImage(thumbnailWidth,
        THUMBNAIL_HEIGHT,
        null,
        IntPtr.Zero);

// write thumbnail to the response object
ms = new MemoryStream();
thumbnailImage.Save(ms, ImageFormat.Jpeg);
Response.ContentType = "image/jpeg";
Response.BinaryWrite(ms.ToArray());
}

finally
{
    if (dbConn != null)
    {
        dbConn.Close();
    }
}
}
} // Page_Load
```

Example 12-18. *Display thumbnail images (.aspx)*

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="CH12TestThumbnailsFromDatabaseVB.aspx.vb"
    Inherits="ASPNetCookbook.VBExamples.CH12TestThumbnailsFromDatabaseVB" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
    <title>Display Thumbnails</title>
    <link rel="stylesheet" href="css/ASPNetCookbook.css">
</head>
<body leftmargin="0" marginheight="0" marginwidth="0" topmargin="0">
    <form id="frmDisplayThumbnails" method="post" runat="server">
        <table width="100%" cellpadding="0" cellspacing="0" border="0">
```

Example 12-18. Display thumbnail images (.aspx) (continued)

```

<tr>
  <td align="center">
    
  </td>
</tr>
<tr>
  <td class="dividerLine">
    </td>
</tr>
</table>
<table width="90%" align="center" border="0">
  <tr>
    <td align="center">&nbsp;</td>
  </tr>
  <tr>
    <td align="center" class="PageHeading">
      Display Thumbnail Images (VB)
    </td>
  </tr>
  <tr>
    <td></td>
  </tr>
  <tr>
    <td align="center">
      <asp:DataList ID="dlImages" Runat="server"
        RepeatColumns="4" RepeatDirection="Horizontal"
        RepeatLayout="Table" Width="50%">
        <HeaderTemplate>
          <tr>
            <td colspan="4" class="SubHeading" align="center">
              Thumbnails of Images In Database<br /><br />
            </td>
          </tr>
        </HeaderTemplate>
        <ItemTemplate>
          <img id="imgThumbnail" runat="server" border="0" >
        </ItemTemplate>
      </asp:DataList>
    </td>
  </tr>
</table>
</form>
</body>
</html>

```

Example 12-19. Display thumbnail images code-behind (.vb)

```

Option Explicit On
Option Strict On

```

```

'-----
'
'   Module Name: CH12TestThumbnailsFromDatabaseVB.aspx.vb
'

```

Example 12-19. Display thumbnail images code-behind (.vb) (continued)

```
' Description: This module provides the code behind for the
'             CH12TestThumbnailsFromDatabaseVB.aspx page.
'
'*****
Imports Microsoft.VisualBasic
Imports System
Imports System.Configuration
Imports System.Data
Imports System.Data.Common
Imports System.Data.OleDb
Imports System.Web.UI.WebControls

Namespace ASPNetCookbook.VBExamples
    Public Class CH12TestThumbnailsFromDatabaseVB
        Inherits System.Web.UI.Page

        'controls on the form
        Protected WithEvents dlImages As System.Web.UI.WebControls.DataList

        '*****
        '
        ' ROUTINE: Page_Load
        '
        ' DESCRIPTION: This routine provides the event handler for the page load
        '              event. It is responsible for initializing the controls
        '              on the page.
        '-----
        Private Sub Page_Load(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) Handles MyBase.Load
            Dim dbConn As OleDbConnection
            Dim dc As OleDbCommand
            Dim dr As OleDbDataReader
            Dim strConnection As String
            Dim strSQL As String

            If (Not Page.IsPostBack) Then
                Try
                    'get the connection string from web.config and open a connection
                    'to the database
                    strConnection = _
                        ConfigurationSettings.AppSettings("dbConnectionString")
                    dbConn = New OleDb.OleDbConnection(strConnection)
                    dbConn.Open()

                    'build the query string and get the data from the database
                    strSQL = "SELECT BookImageID " & _
                        "FROM BookImage"
                    dc = New OleDbCommand(strSQL, dbConn)
                    dr = dc.ExecuteReader()

                    'set the source of the data for the repeater control and bind it
                    dlImages.DataSource = dr
                Catch ex As Exception
                    '
                End Try
            End If
        End Sub
    End Class
End Namespace
```

Example 12-19. Display thumbnail images code-behind (.vb) (continued)

```
        dlImages.DataBind()

    Finally
        'clean up
        If (Not IsNothing(dbConn)) Then
            dbConn.Close()
        End If
    End Try
End If
End Sub 'Page_Load

'*****
'
' ROUTINE: dlImages_ItemDataBound
'
' DESCRIPTION: This routine is the event handler that is called for each
'              item in the datalist after a data bind occurs. It is
'              responsible for setting the source of the image tag to
'              the URL of the page that will generate the thumbnail
'              images with the ID of the appropriate image for the item.
'-----
Private Sub dlImages_ItemDataBound(ByVal sender As Object, _
    ByVal e As System.Web.UI.WebControls.DataListItemEventArgs) _
    Handles dlImages.ItemDataBound

    Dim img As System.Web.UI.HtmlControls.HtmlImage

    'make sure this is an item in the data list (not header etc.)
    If ((e.Item.ItemType = ListItemType.Item) Or _
        (e.Item.ItemType = ListItemType.AlternatingItem)) Then
        'get a reference to the image used for the bar in the row
        img = CType(e.Item.FindControl("imgThumbnail"), _
            System.Web.UI.HtmlControls.HtmlImage)

        'set the source to the page that generates the thumbnail image
        img.Src = CH12ThumbnailFromDatabaseVB.PAGE_NAME & "?" & _
            CH12ThumbnailFromDatabaseVB.QS_IMAGE_ID & "=" & _
            CStr(CType(e.Item.DataItem, DbDataRecord)("BookImageID"))
    End If
End Sub 'dlImages_ItemDataBound
End Class 'CH12TestThumbnailsFromDatabaseVB
End Namespace
```

Example 12-20. Display thumbnail images code-behind (.cs)

```
//-----
//
// Module Name: CH12TestThumbnailsFromDatabaseCS.aspx.cs
//
// Description: This module provides the code behind for the
//              CH12TestThumbnailsFromDatabaseCS.aspx page
//
//*****
```

Example 12-20. Display thumbnail images code-behind (.cs) (continued)

```
using System;
using System.Configuration;
using System.Data;
using System.Data.Common;
using System.Data.OleDb;
using System.Web.UI.WebControls;

namespace ASPNetCookbook.CSExamples
{
    public class CH12TestThumbnailsFromDatabaseCS : System.Web.UI.Page
    {
        // controls on the form
        protected System.Web.UI.WebControls.DataList dlImages;

        //*****
        //
        // ROUTINE: Page_Load
        //
        // DESCRIPTION: This routine provides the event handler for the page
        // load event. It is responsible for initializing the
        // controls on the page.
        //-----
        private void Page_Load(object sender, System.EventArgs e)
        {
            OleDbConnection dbConn = null;
            OleDbCommand dc = null;
            OleDbDataReader dr = null;
            String strConnection = null;
            String strSQL = null;

            // wire the item data bound event
            this.dlImages.ItemDataBound +=
                new DataListItemEventHandler(this.dlImages_ItemDataBound);

            if (!Page.IsPostBack)
            {
                try
                {
                    // get the connection string from web.config and open a connection
                    // to the database
                    strConnection =
                        ConfigurationSettings.AppSettings["dbConnectionString"];
                    dbConn = new OleDbConnection(strConnection);
                    dbConn.Open();

                    // build the query string and get the data from the database
                    strSQL = "SELECT BookImageID " +
                        "FROM BookImage";
                    dc = new OleDbCommand(strSQL, dbConn);
                    dr = dc.ExecuteReader();
                }
            }
        }
    }
}
```

Example 12-20. Display thumbnail images code-behind (.cs) (continued)

```
        // set the source of the data for the repeater control and bind it
        dlImages.DataSource = dr;
        dlImages.DataBind();
    }

    finally
    {
        // clean up
        if (dbConn != null)
        {
            dbConn.Close();
        }
    }
} // Page_Load

//*****
//
// ROUTINE: dlImages_ItemDataBound
//
// DESCRIPTION: This routine is the event handler that is called for
//               each item in the datalist after a data bind occurs.
//               It is responsible for setting the source of the image
//               tag to the URL of the page that will generate the
//               thumbnail images with the ID of the appropriate image
//               for the item.
//-----
private void dlImages_ItemDataBound(Object sender,
    System.Web.UI.WebControls.DataListItemEventArgs e)
{
    System.Web.UI.HtmlControls.HtmlImage img = null;

    // make sure this is an item in the data list (not header etc.)
    if ((e.Item.ItemType == ListItemType.Item) ||
        (e.Item.ItemType == ListItemType.AlternatingItem))
    {
        // get a reference to the image used for the bar in the row
        img = (System.Web.UI.HtmlControls.HtmlImage)
            (e.Item.FindControl("imgThumbnail"));

        // set the source to the page that generates the thumbnail image
        img.Src = CH12ThumbnailFromDatabaseCS.PAGE_NAME + "?" +
            CH12ThumbnailFromDatabaseCS.QS_IMAGE_ID + "=" +
            (((DbDataRecord)(e.Item.DataItem))["BookImageID"]).ToString();
    }
} // dlImages_ItemDataBound
} // CH12TestThumbnailsFromDatabaseCS
}
```