Printing in Windows Forms

Creating printed documents is an essential task in most business environments, but it can be complicated and confusing. Microsoft Visual Studio 2008 provides several classes and dialog boxes that simplify the task of programming printing. This lecture describes the various print dialog boxes, the *PrintDocument* component.

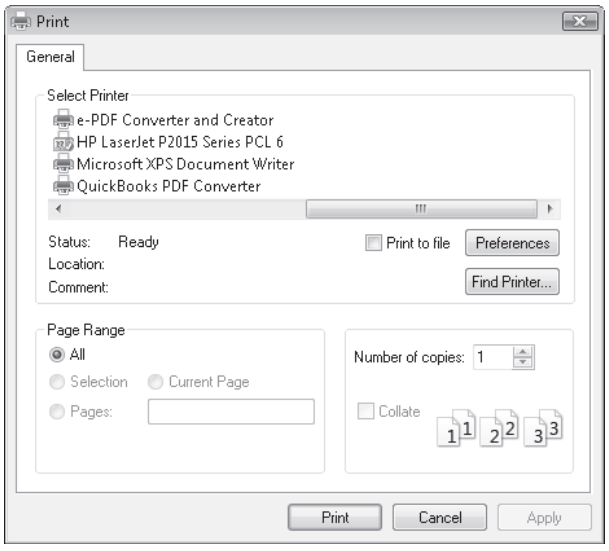
When printing a document, a user typically wants to have a high level of configurability over the printing process. Users typically want to be able to choose options like paper orientation, margins, and paper size. The .NET Framework contains classes that allow you to give your users a broad range of options while still retaining control over the range of options you offer.

The PrinterSettings Class

The class that contains all of the information about a print job and the page settings associated with that print job is the *PrinterSettings* class. Each *PrintDocument* component exposes an instance of the *PrinterSettings* class and accesses this instance to get the settings for the print job. Most of this is done automatically, and the developer need not worry about accessing the *PrinterSettings* class directly. When the user needs to configure settings for a print job, you can enable the user to make appropriate selections by displaying the *PrintDialog* component and the *PageSetupDialog* component.

The *PrintDialog* Component

The *PrintDialog* component encapsulates a Print dialog box. This dialog box can be displayed to the user at run time to allow the user to configure a variety of options for printing and also to choose a printer. The *PrintDialog* component at run time is shown in Figure .



The PrintDialog component also exposes to the developer a variety of properties that allow him or her to configure the interface of the PrintDialog component when it is shown. The important properties of the PrintDialog component are described in Table.

PROPERTY	DESCRIPTION
AllowCurrentPage	Indicates whether the Current Page option button is displayed.
AllowPrintToFile	Indicates whether the Print To File check box is enabled.
AllowSelection	Indicates whether the Selection option button is enabled.
AllowSomePages	Indicates whether the Pages option button is enabled.
Document	The *PrintDocument* that is associated with the *PrintDialog.*
PrinterSettings	The *PrinterSettings* object associated with the selected printer. These settings are modified when the user changes settings in the dialog box.
PrintToFile	Indicates whether the Print To File check box is selected.
ShowHelp	Indicates whether the Help button is shown.
ShowNetwork	Indicates whether the Network button is shown.

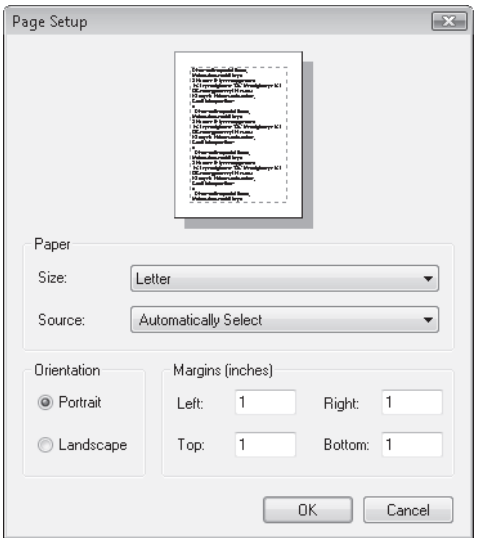
Using the *PrintDialog* to Print a *PrintDocument* Component

After options have been selected, you can query the *DialogResult* returned by the *PrintDialog. ShowDialog* method. If the *DialogResult* is *OK,* you can print the associated *PrintDocument* component by calling the *PrintDocument.Print* method. An example is shown here:

```
DialogResult aResult;
aResult = PrintDialog1.ShowDialog();
if (aResult == DialogResult.OK)
       PrintDialog1.Document.Print();
```

The PageSetupDialog Component

The *PageSetupDialog* component enables users to select options about the setup of the pages for a print job. A *PageSetupDialog* component at run time is shown in Figure.



Changes made by the user in the PageSetupDialog are automatically reflected in the PrinterSettings class of the PrintDocument component that is associated with the PageSetupDialog component. You can set the options that will be displayed to the user by setting the properties of the PageSetupDialog component. Important properties of the PageSetupDialog component are shown in Table.

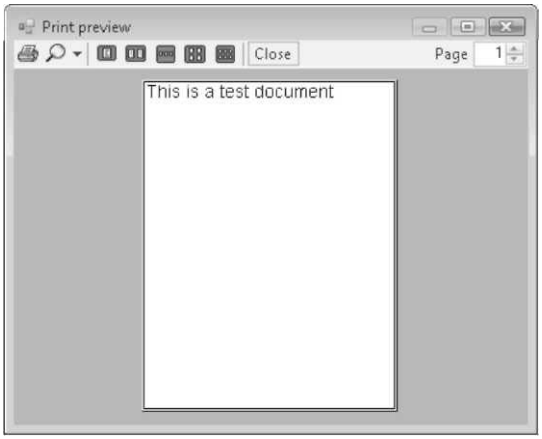
PROPERTY	DESCRIPTION
AllowMargins	Indicates whether the margins section of the dialog box is enabled
AllowOrientation	Indicates whether the orientation section of the dialog box (landscape versus portrait) is enabled
AllowPaper	Indicates whether the paper section of the dialog box (paper source and paper size) is enabled AllowPaper
AllowPrinter	Indicates whether the printer button is enabled AllowPrinter
Document	The PrintDocument associated with this component Document
EnableMetric	Indicates whether the margin settings, when displayed in millimeters, should be automatically converted to and from hundredths of an inch EnableMetric
MinMargins	Indicates the minimum margins, in hundredths of an inch, that the user is allowed to select MinMargins
ShowHelp	Indicates whether the Help button is visible ShowHelp
ShowNetwork	Indicates whether the Network button is visible

To show a *PageSetupDialog,* the component must be associated with an instance of the *PageSettings* class. The preferred way of doing this is to set the *Document* property to an instance of the *PrintDocument* class, which automatically sets the *PrintDocument.PrinterSettings* instance to the *PageSetupDialog.PrinterSettings* property. The following code example demonstrates how to set the *Document* property of the *PageSetupDialog* component and then display it to the user:

```
// Assumes an instance of PrintDocument named printDocument1
// and an  instance of PageSetupDialog named pageSetupDialog1.
pageSetupDialog1.Document = printDocument1;
pageSetupDialog1.ShowDialog();
```

The PrintPreviewDialog Component

The PrintPreviewDialog component is a self-contained dialog box that allows you to preview a print document. It does so by calling the Print method of the PrintDocument component that is specified in the component's Document property and redirecting the output to a graphical representation of the page contained in the dialog box. The PrintDocumentDialog component is shown at run time in Figure.



The dialog box allows the user to view each page before printing, adjust the number of pages displayed, and adjust the zoom factor so that pages can be viewed close up or at a distance. After viewing the preview, the user can choose to print the document by clicking the Print button, which calls the *Print* method of the *PrintDocument* component and directs the output to the printer. The following code example demonstrates how to associate a *PrintDocument* component with a *PrintPreviewDialog* component and display it to the user at run time:

```
printPreviewDialog1.Document = printDocument1;
printPreviewDialog1.ShowDialog();
```

Constructing Print Documents

The *PrintDocument* component represents a document that is sent to a printer. You can use the *PrintDocument* component to print text documents or graphical documents. In this section you will learn how to configure the *PrintDocument* component and use it to create printed documents.

The PrintDocument Component

A printed document is represented by the *PrintDocument* component. It is a component that does not have a visual representation at run time but that appears in the component tray at design time and can be dragged from the Toolbox to the design surface. The *PrintDocument* component encapsulates all of the information necessary to print a page.

Overview of Printing

In the .NET Framework printing model, printed content is provided directly by the application logic. A print job is initiated by the *PrintDocument.Print* method. This starts the print job and then raises one or more *PrintPage* events. If there is no

client method to handle this event, printing does not take place. By providing a method to handle this event, you specify the content to be printed.

If the print job contains multiple pages, one *PrintPage* event is raised for each page in the job. This, in turn, causes the method handling the *PrintPage* event to execute multiple times. Thus, that method must implement functionality to track the print job and ensure that successive pages of a multipage document are printed. Otherwise, the first page of the document will print multiple times.

The PrintPage Event

The *PrintPage* event is the main event involved in printing documents. To actually send content to the printer, you must handle this event and provide code to render the content in the *PrintPage* event handler. All of the objects and information needed to send content to the printer are wrapped in the PrintPageEventArgs object, which is received by the handler for this event. Following table describes the properties of the PrintPageEventArgs object.

PROPERTY	DESCRIPTION
Cancel	Indicates whether a print job should be cancelled. You can cancel a pending print job by setting the *Cancel* property to *True*.
Graphics	The *Graphics* object is used to render content to the printed page. It encapsulates the drawing surface represented by the printer.
HasMorePages	Gets or sets a value that indicates whether additional pages should be printed. Set this property to *True* in the event handler to raise the event again.
MarginBounds	Gets the *Rectangle* object that represents the portion of the page within the margins.
PageBounds	Gets the *Rectangle* object that represents the total page area.
PageSettings	Gets or sets the *PageSettings* object for the current page.

Content is rendered to the printed page by the *Graphics* object provided in the *PrintPageEventArgs* object. The printed page behaves just like a form, control, or any other drawing surface that can be represented by a *Graphics* object. To render content, you use the same methods that you use to render content to a form. The following code example demonstrates a simple method to print a page:

```
// This method inscribes an ellipse inside the bounds of the page.
// this method must handle the PrintPage event in order to send
// content to the printer

public void PrintEllipse(object sender,
                 System.Drawing.Printing.PrintPageEventArgs e)
{
       e.Graphics.DrawEllipse(Pens.Black, e.MarginBounds);
}
```

The *MarginBounds* and *PageBounds* properties represent areas of the page surface. You can specify printing to occur inside the margin bounds of the page by calculating printing coordinates based on the *MarginBounds* rectangle. Printing

that is to take place outside of the margin bounds, such as headers or footers, can be specified by calculating the printing coordinates based on the *PageBounds* rectangle. Print coordinates are in pixels by default.

You can specify that a print job has multiple pages by using the *HasMorePages* property. By default, this property is set to *False.* When your application logic determines that multiple pages are required to print a job, you should set this property to *True.* When the last page is printed, you should reset the property to *False.* Note that the method handling the *PrintPage* event must keep track of the number of pages in the job. Failure to do so can cause unexpected results while printing. For example, if you fail to set the *HasMorePages* property to *False* after the last page is printed, the application will continue to raise *PrintPage* events.

You can cancel a pending print job without finishing by setting the *Cancel* property to *True.* You might do this, for example, if the user clicked a Cancel button on your form.

You can create an event handler for the *PrintPage* event by double-clicking the *PrintDocument* instance in the Designer to create a default event handler.

Printing Graphics

Printing graphics is essentially the same as rendering them to the screen. You use the *Graphics* object supplied by *PrintPageEventArgs* to render graphics to the screen. Simple shapes can be printed or complex shapes can be created and printed using the *GraphicsPath* object. The following code example shows how to print a complex shape with a *GraphicsPath* object:

```
// This method must handle the PrintPage event
public void PrintGraphics(object sender,
                          System.Drawing.Printing.PrintPageEventArgs e)
{
System.Drawing.Drawing2D.GraphicsPath myPath = new
                                  System.Drawing.Drawing2D.GraphicsPath();
myPath.AddPolygon(new Point[] {new Point(1,1), new Point(12, 55),
                           new Point(34, 8), new Point(52, 53),
                                                 new Point(99, 5)});
 myPath.AddRectangle(new Rectangle(33, 43, 20, 20));
 e.Graphics.DrawPath(Pens.Black, myPath);
}
```

To print a graphics job that has multiple pages, you must manually divide the job among pages and implement the appropriate logic. For example, the following method uses 12 pages to draw an ellipse that is six times as long as the page and two times as wide:

```
// These two variables are used to keep track of which page is printing int x; int y;
// This method must handle the PrintPages event.
private void printDocument1_PrintPage(object sender, System.Drawing.Printing.
PrintPageEventArgs e)
{
// Draws the Ellipse at different origination points, which has the
// effect of sending successive page-sized pieces of the ellipse to the
// printer based on the value of x and y
e.Graphics.FillEllipse(Brushes.Blue, new RectangleF(-e.PageBounds.Width * x, -
      e.PageBounds.Height * y, e.PageBounds.Width * 2, e.PageBounds.Height * 6));
```

```
y += 1;
if (y == 6 & x == 0)
   {
     y = 0;
     x ++;
     e.HasMorePages = true;
   }
else if (y == 6 & x == 1)
   {
      // The print job is finished
      e.HasMorePages = false;
   }
else
   {
      // Fires the print event again
      e.HasMorePages = true;
   }
}
```

In this example the method redraws the complete ellipse each time it is executed but the point of origin is changed, so successive "slices" of the ellipse are printed each time the application executes.

Printing Images

You can use the *Graphics* object contained in the *PrintPageEventArgs* to send image content to the printer with the *Graphics.DrawImage* method, as shown here:

```
// Assumes an image called myImage. This method must handle the PrintPages
//event
private void PrintDocument1_PrintPage(object sender,
                                System.Drawing.Printing.PrintPageEventArgs e)
{
    e.Graphics.DrawImage(myImage, new PointF(0,0));
}
```

Note that when printing an image that is larger than a single page, you must handle the paging manually, as you would for printing any other print job.

Printing Text

Printing text is similar to printing graphics. Text is printed through the *Graphics.DrawString* method, which renders a string in the specified font to the printer. As with rendering text to the screen, to print you must specify a font for rendering the text, the text to render, a *Brush* object, and coordinates at which to print. For example:

```
Font myFont = new Font("Tahoma", 12,  FontStyle.Regular,
        GraphicsUnit.Pixel); string Hello   = "Hello World!";
e.Graphics.DrawString(Hello, myFont, Brushes.Black, 20, 20);
```

Note that when printing text, you must take steps in your code to ensure that you do not attempt to print outside the bounds of the page. If you do attempt to print outside the bounds of the page, content that falls outside the bounds will not be printed.

Printing Multiple Lines

When printing multiple lines of text, such as an array of strings or lines read from a text file, you must include logic to calculate the line spacing. You can calculate the number of lines per page by dividing the height of the margin bounds by the height of the font. Similarly, you can calculate the position of each line by multiplying the line number by the height of the font. The following code example demonstrates how to print an array of strings called *myStrings:*

```
int ArrayCounter = 0;
private void PrintStrings(object sender, System.Drawing.Printing.PrintPageEventArgs e)
   {
      // Declares the variables that will be used to keep track of spacing and paging
      float LeftMargin = e.MarginBounds.Left;
      float TopMargin = e.MarginBounds.Top;
      float MyLines = 0;
      float YPosition = 0;
      int Counter = 0;
      string CurrentLine;
      // Calculate the number of lines per page
      MyLines = e.MarginBounds.Height / myFont.GetHeight(e.Graphics);
      // Prints each line of the array, but stops at the end of a page
      while (Counter < MyLines && ArrayCounter <= myStrings.Length -1)
      {
      CurrentLine = myStrings[ArrayCounter];
      YPosition = TopMargin + Counter * myFont.GetHeight(e.Graphics);
      e.Graphics.DrawString(CurrentLine, myFont, Brushes.Black, LeftMargin,
                           YPosition, new StringFormat()); Counter ++; ArrayCounter++;
      }
   // If more lines exist, print another page
   if (!(ArrayCounter == myStrings.GetLength(0) -1))
        e.HasMorePages = true;
   else
        e.HasMorePages = false;
   }
```

Notifying the User When Printing Is Complete

Because printing can be time-consuming and because the process is carried out asynchronously, it can be useful to inform the user when a print job has finished. You can create a notification of the end of printing by handling the *PrintDocument.EndPrint* event.

The *PrintDocument.EndPrint* event is raised after all pages of a print job have been printed. It has a signature that is the same as the *PrintPages* event that is, it includes an object that represents the sender of the event and an instance of *System.Drawing.Printing. PrintPageEventArgs.* The following example demonstrates how to notify the user with a message box when the print job is complete:

```
   // This method handles the PrintDocument.EndPrint method
private void PrintDocument1_EndPrint(object sender System.Drawing.Printing.
                                                              PrintEventArgs e)
   {
       MessageBox.Show("Your print job is complete")
   }
```