

# PrintForm .NET Technical Overview

## Background

When a .NET project requires some kind of custom printed output, the .NET framework provides the PrintDocument and related classes to facilitate the process. Whilst these classes make a very good job of abstracting your code away from printer specifics, writing all the necessary code to render a complicated form can be very time consuming, and the resulting code can be difficult to reuse or adapt to future requirements.

For example, consider the typical TaxForm data entry application below (this is included as a sample application with PrintForm). What is required here is a printout that matches the on-screen display as closely as possible, but anyone who has developed such an application will tell you how long the process of developing the necessary graphics code takes.

Our PrintForm technology arose from the questions 'Why can't we just print the Form?' and '.NET knows how to render it onto the screen, so surely it can render it to the printer as well?'

## PrintForm Technology

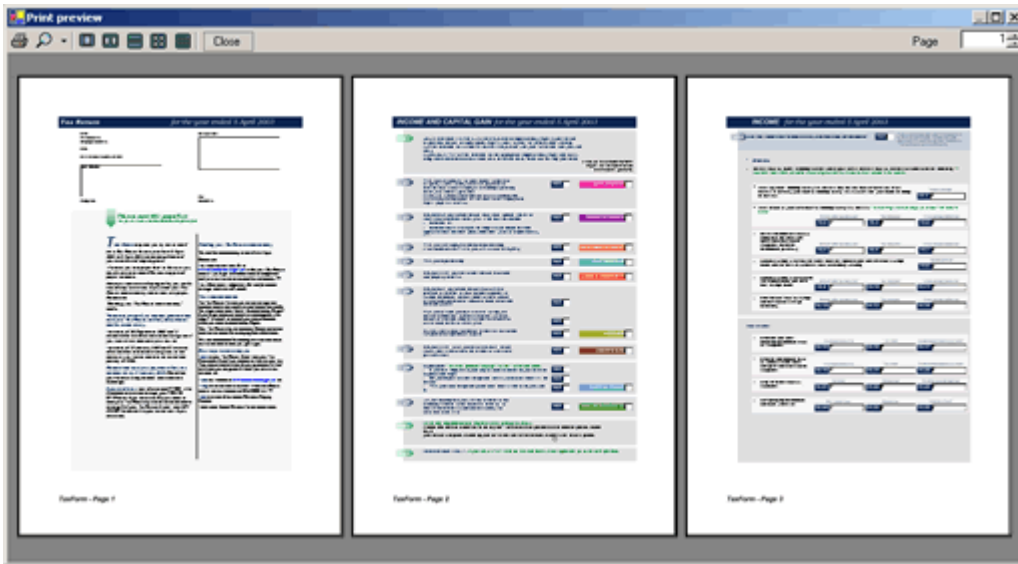
Well, as usual with these things, 'Just printing the Form' turned out to be a little more complex.

The simplest way of printing a Form is to use the underlying Windows API take a bitmap (BitBlt) 'snapshot' of the window. This is unsatisfactory for many reasons, however - the resulting image doesn't scale well to the higher resolutions of most modern printers, resulting in a 'blocky' printout, it doesn't cope with scrollable Forms and you have no control over what is displayed at print time - you literally get a bitmapped image of the Form as you see it on screen at the screen resolution.

By contrast, PrintForm renders each control individually, using the graphics calls the control uses to render itself onto the screen. This means that printouts are resolution independent, and the entire contents of a scrollable region can be rendered, whether or not the controls are in view. It also allows us to selectively omit some controls from the printout - for example you may not want the OK and Cancel buttons to appear on the printout. By the same mechanism we can specify an alternative background color for individual controls at print time, so a Form which is gray at runtime can be printed out with a more appropriate white background etc.

The results of this approach can be seen below. Printing was added to the TaxForm application by simply adding a

PrintForm component to the Form, and setting the PrintForm.BodyContainer property to point at the entire Form. This makes PrintForm render the entire contents of the Form into the MarginBounds area of the printed page:



(Note: in this example there are several Forms in the application, each of which has been rendered on a separate page).

As you can see, designing the printed output is as simple as designing a Windows Form.

## Technical Details

### Using PrintForm in Place of PrintDocument

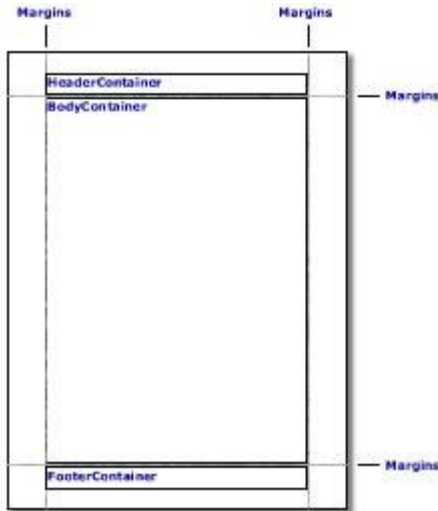
PrintForm is derived from the .NET framework PrintDocument class. This means you can use it in any place where you might use a PrintDocument component.

For example, if you were implementing custom printing in your application without the use of PrintForm you would add a PrintDocument and a PrintPreviewDialog component to your Form, and set the PrintPreviewDialog.Document property to the PrintDocument. You would then handle the PrintDocument.PrintPage event and do all your graphics calls there to layout the page. (See the Microsoft documentation on this at <http://samples.gotdotnet.com/quickstart/winforms/doc/WinFormsPrinting.aspx>).

Custom printing with PrintForm is a very similar process - you add PrintForm and PrintPreviewDialog components to your Form, and set the PrintPreviewDialog.Document property to the PrintForm. Then, instead of writing the PrintPage code yourself, you set PrintForm's BodyContainer property to whichever control you want to print on the page and PrintForm handles all the drawing.

### Headers and Footers with PrintForm

In addition to the BodyContainer, PrintForm also has HeaderContainer and FooterContainer properties. The controls referenced by these two properties are rendered in the positions shown below. The bottom of the HeaderContainer control is aligned with the top of the Margin rectangle, and the top of the FooterContainer control is aligned with the bottom.



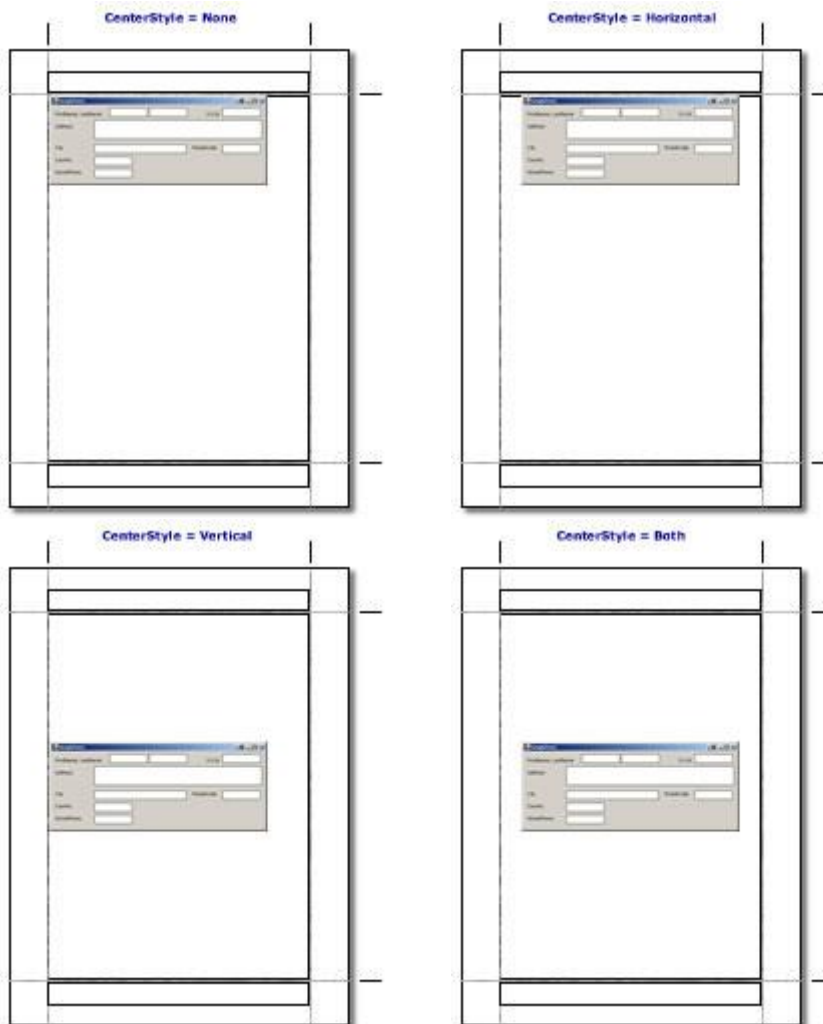
## Controlling the Printed Output

PrintForm provides a number of properties to control how the output is rendered, including scaling, centering, visibility and background color. When the outermost container control (e.g. Form) has a non-client border area this can optionally be printed as well.

Property	Type	Description
AutoFit	PrintForm property	Specifies the elements of the page (BodyContainer, HeaderContainer and FooterContainer) that PrintForm applies automatic scaling to. PrintForm can automatically scale the printed output it produces to allow a large Form to fit onto a single printed page. When AutoFit is set to PageElement.None the output is scaled according to the setting of the ManualZoom property.
CenterStyle	PrintForm property	Indicates how the page contents are centered when printing*. When the printed output from PrintForm does not entirely fill the MarginBounds rectangle, the CenterStyle property can be set to automatically center the output in either the horizontal or vertical dimension.
ManualZoom	PrintForm property	The scale to zoom to when AutoFit is not enabled. When AutoFit is set to PageElement.None the printed output is scaled according to the setting of this property.
PrintBorders	PrintForm property	The borders, or non-client area, of the outermost control you are rendering can be turned on or off using the PrintForm.PrintBorders property. For example if you are printing a Form in a report style document you may not want the Form title bar and border to be printed, in this case you would set PrintBorders = None. Alternatively you may be printing a 'screen shot' style representation of a Form where you want the title bar and border to be shown, but you don't want the borders on your header and footer controls to print - in which case you would set PrintBorders = Body. If you want borders on header, body and footer set PrintBorders = All.
BackColorWhilePrinting	Extended property provided to all Form controls by PrintForm	A System.Drawing.Color which is used for the specified control when printing.

VisibleWhilePrinting	Extended property provided to all Form controls by PrintForm	Determines whether the specified control should be visible when printing.
----------------------	--	---

\* The CenterStyle options result in the following placement of the output:



## Using PrintForm.PrintControl for Custom Printing

It may not always be appropriate to use PrintForm as a PrintDocument, for example:

- | you may want to print each tab of a tab control one after the other on a single page. PrintForm in PrintDocument mode would only allow you to specify one tab in the BodyContainer property
- | you may want to print more than one form on a single page. PrintForm in PrintDocument mode would again only allow you to specify one Form in the BodyContainer property
- | you may have your own PrintDocument derived class already and you just want to use PrintForm to render a specific control or group of controls as a part of your page layout

For these reasons PrintForm provides a utility method **PrintControl**. Using this you can render any control into a given rectangle at a given scale. Using this you can:

- | Print a control or group of controls to a Bitmap or file
- | Print multiple controls on a page in a different arrangement to their layout on the Form
- | Print multiple Forms on a page

A number of samples are provided with PrintForm which illustrate each of these procedures.

## Chaining Multiple PrintDocuments Into a Single Print Job with PrintChainManager

The life cycle of printing using a PrintDocument class is explained in detail in the .NET documentation, but briefly when Print is called on any PrintDocument or PrintDocument derived class, the sequence is:

**PrintDocument.Print** called.

**OnBeginPrint** (base raises **BeginPrint** event) is called at start of print job.

For each page:

**OnQueryPageSettings** (base raises **QueryPageSettings** event) is called.

**OnPrintPage** (base raises **PrintPage** event) is called, looping continues while **HasMorePages** is set **True**.

**OnEndPrint** (base raises **QueryPageSettings** event) is called.

Print job ends.

So if we have many Forms in an application and each one has a PrintDocument (or a PrintForm) how do we make them all part of the same print job? For this PrintForm provides the PrintChainManager class and the IChainedDocument interface.

PrintChainManager is itself derived from PrintDocument and will daisy chain together a sequence of objects which implement the IChainedDocument interface into a single print job. The TaxForm sample at the start of this document demonstrates this approach, allowing an application consisting of a number of separate Forms to be presented as an integrated application with the ability to print the sequence of Forms as a single document.

## Summary

This short overview has covered the how and why of PrintForm. If you'd like to give it a try you can download a free trial copy from <http://www.winformreports.co.uk>

This article © 2003 TMG Development Ltd. All Rights Reserved.