






# PrintWorks Help Contents

## Introduction

-  [Overview](#)
-  [File Distribution](#)
-  [Integrating With Visual Forms for Windows](#)
-  [Using Metric and Other Units](#)
-  [Support for Visual C++](#)

## Properties

### **Document Management**

- [CreateDocument](#)
- [CreatePage](#)
- [DeleteDocument](#)
- [DocTitle](#)
- [HideButton](#)
- [hMetaDC](#)
- [Landscape](#)
- [PageTitle](#)
- [PaperHeight](#)
- [PaperWidth](#)
- [Resolution](#)
- [SavePage](#)

### **Object Sizing & Positioning**

- [Aspect](#)
- [ObjHeight](#)
- [ObjWidth](#)
- [EndAngle](#)
- [Radius](#)
- [StartAngle](#)
- [XPos](#)
- [X2](#)
- [YPos](#)
- [Y2](#)

### **Object Appearance**

- [BackgroundColor](#)
- [EdgeColor](#)
- [FaceColor](#)
- [LineColor](#)
- [LineWeight](#)
- [LineWeightR](#)
- [MakeTransparent](#)
- [MakeOpaque](#)
- [Pattern](#)

### **Three-D Objects**

- [ThreeDAngle](#)
- [ThreeDBar](#)

ThreeDPie  
ThreeDThickness

## **Drawing Text**

FieldWidth  
GetXTextPos  
GetYTextPos  
Indent  
Italic  
Justify  
LineSpacing  
Multiline  
NameFont  
Paragraph  
PrintText  
Rotation  
SelectFont  
SizeFont  
Underline  
WeightFont

## **Line Drawing**

DrawCircle  
DrawLine  
Pie  
Rectangle

## **Area Fills**

CurrentPoint  
FillCircle  
FillPolygon  
FillRectangle  
NumPoints  
SetPoint

## **Adding Graphics**

DrawBitmap  
Filename  
MergeMetaFile  
MetaFile  
StretchBitmap

## **Filling In Forms**

ClearFields  
FieldData  
FieldID  
FieldNumber  
FillFieldID  
FillFieldNumber  
FormFile  
MergeForm  
TemplateFile

## **Printing**

FirstPrintPage  
LastPrintPage  
PrintAll  
PrintDocument  
PrintDialog  
PrintEventOnly  
XPrintOffset  
YPrintOffset

 **Previewing**

DeskColor  
DisplayPageNum  
DrawNow  
PercentScreen  
PictureScale  
Preview  
ShadowWidth  
StartCloseUp

## Events

ClickIn  
PreviewClosing  
PrintEvent

# CreateDocument Property

CreateDocument is the first step in using PrintWorks. This property must be called to prepare for [Creating Pages](#), [Printing](#), and [Previewing](#).

Set the [DocTitle](#) property before calling CreateDocument.

Refer to [Document Management](#) for an overview.

Data Type: Action - executed when set to True

# CreatePage Property

CreatePage initializes a new page in your document.

Set the [TemplateFile](#), [PaperWidth](#), [PaperHeight](#), [Resolution](#), and [Landscape](#) properties before calling CreatePage.

Refer to [Document Management](#) for an overview.

Data Type: Action - executed when set to True

# DeleteDocument Property

DeleteDocument deletes a document from memory and performs other important housekeeping chores. You must call DeleteDocument before exiting your program if a document has been created, unless Preview was called. Preview calls DeleteDocument itself, so it is not necessary to do it in your code.

Data Type: Action - executed when set to True

# PageTitle Property

PageTitle sets the name of the page that is displayed in the status bar of the Preview window.

Data Type: String (char\*)

# PaperWidth Property

PaperWidth sets the physical width of the paper to be displayed or printed. Keep in mind that PrintWorks printing logic automatically offsets the printed page 1/4 inch to the left and 1/4 inch up to allow for the unprintable zone in most printers. This allows you to specify a page as its actual width (for example 8.5 inches) rather than a narrower width, and then having to offset the output on the page. Also, the page will be displayed correctly on the screen, which has no unprintable zone. Make sure you do not put any graphics or text in the unprintable zone, because on most printers they will not be displayed properly, or at all.

When including Landscape oriented pages, reverse the height and width when setting PaperWidth and PaperHeight. This will cause the page to be readable on the screen. Use the Landscape property so the page will be properly rotated when printing.

Data Type: Real (float)



# PaperHeight Property

PaperHeight sets the physical height of the paper to be displayed or printed. Keep in mind that PrintWorks printing logic automatically offsets the printed page 1/4 inch to the left and 1/4 inch up to allow for the unprintable zone in most printers. This allows you to specify a page as its actual height (for example 11.0 inches) rather than a shorter height, and then having to offset the output on the page. Also, the page will be displayed correctly on the screen, which has no unprintable zone. Make sure you do not put any graphics or text in the unprintable zone, because on most printers they will not be displayed properly, or at all.

When including Landscape oriented pages, reverse the height and width when setting PaperWidth and PaperHeight. This will cause the page to be readable on the screen. Use the Landscape property so the page will be properly rotated when printing.

Data Type: Real (float)

# Resolution Property

Resolution sets the nominal resolution for a page. Although PrintWorks creates scalable metafiles, Resolution is used to calculate the thickness of lines and the height of fonts. You can make the line thickness independent of resolution by using [LineWeightR](#) instead of [LineWeight](#) to set line thickness.

If you are using inches as your unit of measure, a resolution of 300 is recommended because it is one pixel on a laser printer at standard resolution. If you are using metric units, use 118 ( $300 / 2.54$ ). Also, if you are using metric (or other) units, use [LineWeightR](#) to set the line weight. Refer to [Using Metric and Other Units](#).

Data Type: Integer (int)

# DocTitle Property

DocTitle sets the title in the title bar of the Preview window.

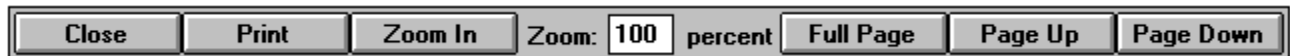
Data Type: String (char\*)

# Preview Property

Preview activates the Preview window. This is a fully functional print preview window that allows you to scroll through the pages in your document, zoom in or out, set the zoom scale, and print.

Set PercentScreen, StartCloseUp and the print option properties before calling Preview. Also, you must have created a document and at least one page. You do not need to call DeleteDocument after exiting Preview, because it is done automatically. When Preview is closing, it fires the PreviewClosing event. Respond to this event to execute any clean up code.

Click on the toolbar controls shown below for details on their operation.



Data Type: Action - executed when set to True

# FormFile Property

FormFile specifies the file name of a Visual Forms for Windows form that has been saved as a Windows metafile. A form can be added to a page using the MergeForm property.

Refer to Filling In Forms for details on working with forms and templates.

Data Type: String (char\*)

# MergeForm Property

MergeForm adds a [Visual Forms for Windows](#) form to a page.

Set the [FormFile](#) property before calling MergeForm.

Refer to [Filling In Forms](#) for details on working with forms and templates.

Data Type: Action - executed when set to True

# XPos Property

XPos sets the horizontal position of the starting point of a figure, in inches (or other units). It is measured from the left edge of the page. If you are working with a Landscape page, the left edge is the short side (it will be rotated when printed).

Data Type: Real (float)

# YPos Property

YPos sets the vertical position of the starting point of a figure, in inches (or other units). It is measured from the top edge of the page. If you are working with a Landscape page, the left edge is the long side (it will be rotated when printed).

Data Type: Real (float)



# ObjWidth Property

ObjWidth sets the width of an object. The width is always parallel to the top edge of the page.

Data Type: Real (float)

# ObjHeight Property

ObjHeight sets the height of an object. The height is always parallel to the left edge of the page.

Data Type: Real (float)

# MergeMetaFile Property

MergeMetaFile adds a Windows disk based metafile, specified by MetaFile (the file name) to the page. The upper left corner of the metafile is drawn at XPos, YPos, and the width and height are ObjWidth and ObjHeight, respectively.

Set MetaFile, XPos, YPos, ObjWidth, and ObjHeight before calling MergeMetaFile.

Data Type: Action - executed when set to True

# MetaFile Property

MetaFile specifies the file name of a Windows metafile.

Data Type: String (char\*)

# PercentScreen Property

PercentScreen specifies what portion of the available Preview screen is used to display a page, in percent. If PercentScreen is 100, the page uses the maximum available screen real estate.

Data Type: Integer (int)

# ShadowWidth Property

ShadowWidth specifies the width, in pixels, of the page shadow on the screen.

Data Type: Integer (int)

# FieldNumber Property

FieldNumber specifies a data field in a TemplateFile, by its sequential number. Refer to the template.TXT file produced by Visual Forms for Windows, for a listing of the fields and their respective numbers.

Refer to Filling In Forms for details on working with forms and templates.

Data Type: Integer (int)

# FieldID Property

FieldID specifies a data field in a TemplateFile, by its name. Refer to the template.TXT file produced by Visual Forms for Windows, for a listing of the fields and their respective names.

Refer to Filling In Forms for details on working with forms and templates.

Data Type: String (char\*)



# FillFieldNumber Property

FillFieldNumber fills a data field in a template by referencing its sequential number.

Set the [FieldNumber](#) and [FieldData](#) properties before calling FillFieldNumber.

Refer to [Filling In Forms](#) for details on working with forms and templates.

Data Type: Action - executed when set to True

## FillFieldID Property

FillFieldID fills a data field in a template by referencing its name, or ID.

Set the [FieldID](#) and [FieldData](#) properties before calling FillFieldID.

Refer to [Filling In Forms](#) for details on working with forms and templates.

Data Type: Action - executed when set to True

# FieldData Property

FieldData contains the actual text string that will be placed in a template's data field.

Refer to [Filling In Forms](#) for details on working with forms and templates.

Data Type: String (char\*)

# ClearFields Property

ClearFields erases the text assigned to all of a template's data fields. Call ClearFields after creating a page, and before assigning any data to data fields.

Refer to [Filling In Forms](#) for details on working with forms and templates.

Data Type: Action - executed when set to True

# PrintDocument Property

PrintDocument allows you to print your document without invoking the Preview window. It works the same as from within the Preview window, except the PrintEvent custom event is not fired. You may print directly, or use the PrintDialog. If the PrintDialog is not used, all pages in the document are printed if PrintAll is True, otherwise, page numbers FirstPrintPage to LastPrintPage are printed.

Set PrintAll, FirstPrintPage, LastPrintPage, and PrintDialog before calling PrintDocument.

Make sure you call DeleteDocument after calling PrintDocument.

Data Type: Action - executed when set to True

# FirstPrintPage Property

FirstPrintPage specifies the first page in the document to be printed. This property is over-ridden if PrintAll is True.

Data Type: Integer (int)

# LastPrintPage Property

LastPrintPage specifies the last page in the document to be printed. This property is over-ridden if PrintAll is True.

Data Type: Integer (int)

# PrintCurrent Property

Data Type: Action - executed when set to True

Data Type: True/False (BOOL)

Data Type: Integer (int)

Data Type: Real (float)

Data Type: String (char\*)



# **PrintRange Property**

# PrintAll Property

PrintAll instructs the printer to print all the pages in the document. When True, this property over-rides the FirstPrintPage and LastPrintPage properties.

Data Type: True/False (BOOL)

# SetPoint Property

SetPoint creates a point in an array of points prior to a call to FillPolygon. It uses XPos and YPos, to set the value of point in the array specified by CurrentPoint.

Set the CurrentPoint, XPos and YPos properties before calling SetPoint.

Data Type: Action - executed when set to True

## NumPoints Property

NumPoints specifies the number of points in the point array to be used by FillPolygon. There are a maximum of 100 points available, using an index base of 1 (i.e. allowable range = 1 - 100; NOT 0 - 99).

Data Type: Integer (int)

# Radius Property

Radius specifies the radius of a circle, ellipse, or pie.

Data Type: Real (float)

# ThreeDThickness Property

ThreeDThickness specifies the edge thickness of a ThreeDBar or ThreeDPie.

Data Type: Real (float)

## ThreeDAngle Property

ThreeDAngle specifies the angle a bar is rotated to achieve a ThreeDBar. The greater the angle of rotation, the more of the edge is visible.

Data Type: Real (float)

# FaceColor Property

FaceColor is the color of the two dimensional face of a filled or three dimensional object. It is also the foreground color used to draw monochrome bitmaps. Color bitmaps ignore this property.

Data Type: RGBCOLOR (long)



# EdgeColor Property

EdgeColor is the color of border drawn around a filled object, or the edge of a ThreeDBar or ThreeDPie.

Data Type: RGBCOLOR (long)

# StartAngle Property

StartAngle is the beginning angle, in degrees, of a circle, ellipse, or pie. Circles, ellipses and pies are always drawn from the StartAngle counter-clockwise to the EndAngle. The allowable range is 0 to 360. It is allowable for the StartAngle to be greater than the EndAngle. When this is the case, drawing of the figure passes 0 degrees, and continues to the EndAngle.

Data Type: Real (float)

## EndAngle Property

EndAngle is the ending angle, in degrees, of a circle, ellipse, or pie. Circles, ellipses, and pies are always drawn from the StartAngle counter-clockwise to the EndAngle. The allowable range is 0 to 360. It is allowable for the EndAngle to be less than the StartAngle. When this is the case, drawing of the figure passes 0 degrees, and continues to the EndAngle.

Data Type: Real (float)

# Polygon Property

Data Type: Action - executed when set to True

Data Type: True/False (BOOL)

Data Type: Integer (int)

Data Type: Real (float)

Data Type: String (char\*)

# FillPolygon Property

FillPolygon fills a polygon with the FaceColor and Pattern, and encloses it with a border of EdgeColor and LineWeight. The polygon is defined by an array having NumPoints points created using SetPoint.

Create a point array using SetPoint, and set FaceColor, Pattern, LineWeight (or LineWeightR), EdgeColor and NumPoints before calling FillPolygon.

Data Type: Action - executed when set to True

## DrawCircle Property

DrawCircle draws a circle, arc or ellipse. The center is specified by XPos and YPos; the size is specified by Radius and Aspect; and, the angles are specified by StartAngle and EndAngle. The circle's color is LineColor, and line thickness is LineWeight (or LineWeightR).

Set the XPos, YPos, Radius, Aspect, StartAngle, EndAngle, LineColor and LineWeight (or, LineWeightR) before calling DrawCircle.

Data Type: Action - executed when set to True

# FillCircle Property

FillCircle fills a circular area with FaceColor and Pattern. The dimensions are the same as for DrawCircle.

Set the XPos, YPos, Radius, Aspect, StartAngle, EndAngle, FaceColor, EdgeColor, and Pattern before calling FillCircle.

Data Type: Action - executed when set to True

# Pie Property

Pie draws a pie-shaped figure. It is the same as DrawCircle, except the starting and ending radii are drawn.

Set the XPos, YPos, Radius, Aspect, StartAngle, EndAngle, LineColor, LineWeight (or, LineWeightR) properties before calling Pie.

Data Type: Action - executed when set to True



# ThreeDPie Property

ThreeDPie draws a three dimensional pie slice (or complete pie). The face of the slice is the same as FillCircle. The wedge is "extruded" downward by the amount of ThreeDThickness. The extruded portion has the color EdgeColor. ThreeDPies are solid objects, even if a pattern is used for the face. For three dimensional pie charts to render properly, start with the slices in the back and move forward. This will ensure that the edge of the most forward slices is visible, rather than the edge of one of the slices in the back.

Set XPos, YPos, Radius, Aspect, StartAngle, Endangle, FaceColor, Pattern, BackgroundColor and EdgeColor before calling ThreeDPie.

Data Type: Action - executed when set to True

# ThreeDBar Property

ThreeDBar draws a three dimensional vertical bar. The face of the bar is the same as FillRectangle, except that XPos and YPos refer to the lower left corner of the bar. The edge is "extruded" toward the background at an angle defined by ThreeDAngle, and thickness defined by ThreeDThickness. The extruded portion has the color EdgeColor. ThreeDBars are solid objects, even if a Pattern is specified for the face. Make sure that you draw bars that are in the background first, and in the foreground last. Also, if you are stacking bars, as in the demo, draw the bottom bar section first, proceeding upward to the top portion last.

Set XPos, YPos, ObjWidth, ObjHeight, FaceColor, Pattern, EdgeColor, BackgroundColor, ThreeDThickness and ThreeDAngle before calling ThreeDBar.

Data Type: Action - executed when set to True

# Aspect Property

Aspect is the ratio of the vertical radius to the horizontal radius of a circular or elliptical figure. A perfectly round figure has an Aspect of 1.0. Use a "flattened" Aspect for pie charts (less than 1.0) for a three dimensional effect.

Data Type: Real (float)

# TemplateFile Property

TemplateFile specifies the name of a template file containing a data template. The TemplateFile must have been created using Visual Forms for Windows. TemplateFile must be set before calling CreatePage.

Data Type: String (char\*)

# Pattern Property

Pattern specifies a pre-defined pattern to use in filling an area. The color of the pattern lines is the FaceColor property. There are six standard Windows patterns, numbered 0 - 5. Use a Pattern value of -1 to create a solid fill. When patterns are drawn, they are either transparent or opaque. Transparent patterns are created by calling MakeTransparent. When they are drawn, transparent patterns, leave the background intact between the pattern lines. Opaque patterns are created by calling MakeOpaque. When they are drawn, opaque patterns fill the are between the pattern lines with the BackgroundColor.

Available patterns:

-1	Solid
0	Horizontal
1	Vertical
2	Forward Diagonal
3	Backward Diagonal
4	Cross Hatch
5	Diagonal Cross Hatch

Data Type: Integer (int)

# LineColor Property

LineColor specifies the color to use when drawing line objects. Set the LineColor whenever you wish to change colors of lines you are drawing.

Data Type: RGBCOLOR (long)

# LineWeight Property

LineWeight and LineWeightR specify the thickness of lines to be drawn. The LineWeight property specifies the thickness in pixels, while the LineWeightR property specifies the thickness in units (usually inches or cm). When using metric or other units than inches, it is recommended that LineWeightR be used to ensure line thicknesses will scale properly. Using a LineWeight (or, LineWeightR) of zero, draws a one pixel thick line, no matter what the display or printer resolution.

Data Type: Integer (int)

# DrawLine Property

DrawLine draws a solid line from XPos, YPos to X2, Y2, using the current LineWeight (or, LineWeightR) and LineColor.

Set XPos, YPos, X2, Y2, LineWeight (or, LineWeightR), and LineColor before calling DrawLine.

Data Type: Action - executed when set to True



## **X2 Property**

X2 specifies the x (or, horizontal) coordinate of a destination point of a figure. It is measured in units (usually inches or cm) from the left edge of the page.

Data Type: Real (float)

## **Y2 Property**

Y2 specifies the y (or, vertical) coordinate of a destination point of a figure. It is measured in units (usually inches or cm) from the top edge of the page.

Data Type: Real (float)

# BackColor Property

BackColor specifies the color to fill in between Pattern lines when drawing an object as opaque (after a call to MakeOpaque). This color is ignored when filling transparently.

Data Type: RGBCOLOR (long)

# Rectangle Property

Rectangle draws a rectangle from XPos, YPos as the upper left corner, to X2, Y2 as the lower right corner, using the current LineWeight (or, LineWeightR) and LineColor.

Set XPos, YPos, X2, Y2, LineWeight (or, LineWeightR), and LineColor before calling Rectangle.

Data Type: Action - executed when set to True

# FillRectangle Property

FillRectangle draws a rectangle from XPos, YPos as the upper left corner, to X2, Y2 as the lower right corner, and fills it with the current FaceColor and Pattern. The outline of the rectangular area is drawn with the current EdgeColor and LineWeight (or, LineWeightR).

Set XPos, YPos, X2, Y2, LineWeight (or, LineWeightR), Pattern, and EdgeColor before calling FillRectangle.

Data Type: Action - executed when set to True

# CurrentPoint Property

CurrentPoint specifies the number (in the range of 1 to 100) of an array index for the points describing a polygon. It is used with SetPoint to add a coordinate pair (XPos and YPos) to the array.

Data Type: Integer (int)

## StartCloseUp Property

StartCloseUp, when True, instructs the Preview window to initially display pages zoomed in. When False, the Preview window initially displays pages in full-page, or zoomed out.

Data Type: True/False (BOOL)

# MakeOpaque Property

MakeOpaque sets the background filling mode to opaque. When this is True, the existing background is replaced by the BackGroundColor when filling areas, and is visible between the lines of the Pattern. If a solid Pattern is used, this property has no effect.

Data Type: Action - executed when set to True



# MakeTransparent Property

MakeTransparent sets the background filling mode to transparent. When this is True, the existing background is visible between the Pattern lines in an area fill. If the Pattern is solid, this property has no effect.

Data Type: Action - executed when set to True

# SizeFont Property

SizeFont specifies the height of a font in points; where, one point is 1/72 of an inch (or, 72 points per inch). If you are using other units than inches, scale the SizeFont appropriately.

Example Metric scaling:

For a 12.0 point font,  $\text{SizeFont} = 12.0 \times 2.54 = 30.5$

Data Type: Real (float)

# NameFont Property

NameFont specifies the name of the TrueType font (i.e. "Arial"). Only TrueType fonts may be used with SelectFont.

Data Type: String (char\*)

# WeightFont Property

WeightFont specifies the weight of the font (allowable range: 1 - 10). A medium weight or regular font would typically have a WeightFont of 5. Light fonts would typically be 3; and, bold fonts would typically be 7.

Data Type: Integer (int)

# Italic Property

Italic specifies that the font to be selected will be an Italic font.

Data Type: True/False (BOOL)

# **Underline Property**

Underline specifies that the font to be selected will be underlined.

Data Type: True/False (BOOL)

# Rotation Property

Rotation specifies the angle at which text will be printed. Normal text is printed with an angle of zero. The formatting capabilities of Paragraph, the text drawing property, are available for the four major orientations (see table below). Text drawn at other angles should always use a value of zero for Justify (i.e. left justified). Keep in mind that Rotation is a property of a selected font and is fixed when SelectFont is called. It is not a property of the text itself. If you are using the same font in a number of orientations, Rotation must be specified and SelectFont called appropriately. For efficiency draw all text having a common font and/or Rotation at one time.

The font Rotation angle is in degrees, starting from east going counter-clockwise.

Major Orientations:

Portrait	0 degrees
Landscape	90 degrees
Reverse Portrait	180 degrees
Reverse Landscape	270 degrees

Data Type: Real (float)

# PrintText Property

PrintText is the text string to be printed (or, drawn) using the Paragraph property.

Data Type: String (char\*)



## FieldWidth Property

FieldWidth specifies the width of a text field in inches (or, other units). The width of the field is used to determine how a line of text is formatted. For example, with left justified text, the first character abuts the left side of the field. Centered text is centered within the field, and with right justified text, the last character abuts the right side of the field. The field begins horizontally at XPos. For decimal aligned text, the decimal point is placed at XPos. The baseline of the field (the imaginary line that text "sits on" is YPos.

Data Type: Real (float)

# Indent Property

Indent specifies the distance in inches (or, other units) to indent the first line of a Paragraph.

Data Type: Real (float)

# LineSpacing Property

LineSpacing specifies the vertical distance between successive lines in a multi-line Paragraph. This property is ignored if MultiLine is False.

Data Type: Real (float)

# Justify Property

Justify specifies how text is formatted within a text field. It works together with FieldWidth which specifies the width of a text field in inches (or, other units). For example, with left justified text, the first character abuts the left side of the field. Centered text is centered within the field, and with right justified text, the last character abuts the right side of the field. Fully justified text is the equivalent of left and right justified. The last justification option is decimal alignment. In this format, the decimal point in the text string is placed at the left edge of the field, and the rest of the line is drawn accordingly.

Justification Options:

- |   |                 |
|---|-----------------|
| 0 | Left Justified  |
| 1 | Centered        |
| 2 | Right Justified |
| 3 | Fully Justified |
| 4 | Decimal Align   |

Data Type: Integer (int)

# SelectFont Property

SelectFont selects a TrueType font with the characteristics specified by NameFont, SizeFont, WeightFont, Italic, Underline, and Rotation. All subsequently drawn text will use this font, until another is selected. For efficiency and speed, it is recommended that all text using a particular font be drawn before selecting another font.

Set NameFont, SizeFont, WeightFont, Italic, Underline, and Rotation before calling SelectFont.

Data Type: Action - executed when set to True

# Paragraph Property

Paragraph draws one or more lines of formatted text in the current LineColor using the current font. You first call SelectFont, then Paragraph to draw the text. The text is drawn in a field, which is determined by XPos as the beginning, YPos as the baseline (imaginary line the text sits on), and FieldWidth as the width of the field. Only those words that will fit in the field are drawn. The text is formatted according to the Justify property; and may be more than one line if MultiLine is True. If more than one line are drawn, the subsequent lines are LineSpacing below the previous line. The first line is indented by the amount of Indent.

After a call to Paragraph, the GetXTextPos and GetYTextPos properties are updated to reflect the position where text drawing ended. This is useful if you wish to mix fonts on the same line. For example, drawing a bold label, followed by data. You would select a bold font, print the label, select a regular weight font, set XPos = GetXTextPos and YPos = GetYTextPos and then, print the data. You could determine the correct printing position of the data directly, without having to resort to trial and error.

Set XPos, YPos, PrintText, FieldWidth, Indent, LineSpacing, Justify, LineColor, and MultiLine before calling Paragraph.

Data Type: Action - executed when set to True

NOTE: It is not necessary to call SelectFont if the current font (i.e. last font selected) is the proper font.

## **GetXTextPos Property**

GetXTextPos contains the horizontal position, in inches (or, other units) where the last call to Paragraph finished drawing text.

Data Type: Real (float)

## GetYTextPos Property

GetYTextPos contains the vertical position, in inches (or, other units) where the last call to Paragraph finished drawing text.

Data Type: Real (float)



# MultiLine Property

MultiLine specifies whether more than one line of text can be drawn when Paragraph is called.

Data Type: True/False (BOOL)

# DrawBitmap Property

DrawBitmap adds a disk-based bitmap, contained in the file specified by Filename, to a page. It is positioned with the upper left corner at XPos, YPos. The width is ObjWidth, and the height is ObjHeight. If StretchBitmap is False, the bitmap will not be scaled to fill the ObjWidth and ObjHeight dimensions, but will be drawn at its non-scaled size when printed or displayed at full resolution. If you don't know the actual size of a bitmap, set StretchBitmap to True, so you can control its size. If you are sure of the size, and are happy with the size, set StretchBitmap to False, to obtain a more accurate rendering.

Color bitmaps are drawn with its colors un-altered. Monochrome bitmaps are drawn with the FaceColor as the foreground color, and BackGroundColor as the background color.

Set XPos, YPos, ObjWidth, ObjHeight, Filename, FaceColor, BackGroundColor, and StretchBitmap before calling DrawBitmap.

Data Type: Action - executed when set to True

# StretchBitmap Property

StretchBitmap specifies whether a bitmap can be scaled to fit a specified area (True), or should be drawn at its original dimensions (False).

Data Type: True/False (BOOL)

# Filename Property

Filename specifies the name of a disk file. It is used for some action properties that require the name of a file.

Data Type: String (char\*)

## **XPrintOffset Property**

XPrintOffset adjusts the horizontal positioning of the page to be drawn on the physical paper, with positive values moving the printing to the right on the page. It is useful for printers that do not line up paper correctly, and when you are printing close to the edge. Keep in mind that PrintWorks automatically adjusts the print position 1/4 inch to the left and upward, to account for the un-printable zone in most printers. You can override this adjustment by setting XPrintOffset and YPrintOffset to 0.25. XPrintOffset and YPrintOffset appear in the PrintDialog dialog box, so the user can make adjustments for his specific printer.

Data Type: Real (float)

## YPrintOffset Property

YPrintOffset adjusts the vertical positioning of the page to be drawn on the physical paper, with positive values moving the printing toward the bottom of the page. It is useful for printers that do not line up paper correctly, and when you are printing close to the edge. Keep in mind that PrintWorks automatically adjusts the print position 1/4 inch to the left and upward, to account for the un-printable zone in most printers. You can over-ride this adjustment by setting XPrintOffset and YPrintOffset to 0.25. XPrintOffset and YPrintOffset appear in the PrintDialog dialog box, so the user can make adjustments for his specific printer.

Data Type: Real (float)

## **hMetaDC Property**

hMetaDC contains the metafile device context of the current page. It is useful if you wish to access the Windows API directly. You can have all the capabilities of the Windows GDI for your own specialize routines. Any calls to the GDI will add records to the metafile representing the current page.

Data Type: Integer (int)

NOTE: C/C++ users must cast this to a handle to a device context (i.e. (HDC) hMetaDC)

# SavePage Property

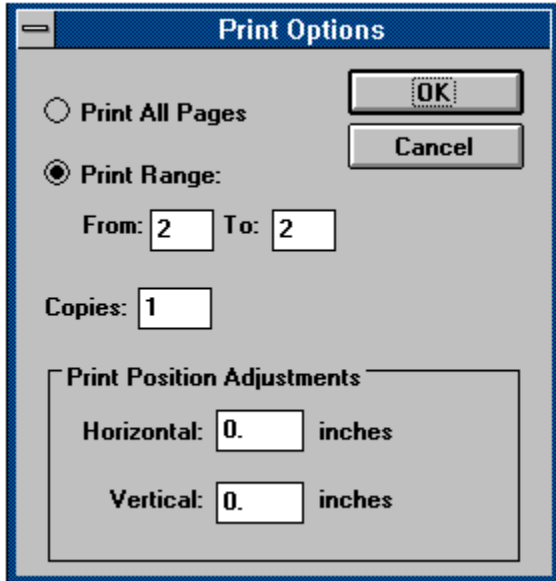
SavePage allows you to save the current page to a disk file, specified by Filename. When SavePage is called, the page is closed, and you cannot add additional graphics. If you are planning to Preview or Print the document, make sure the page is complete. You can create a page for the sole purpose of saving it. For example, you might like to create a scalable logo as a Windows metafile. Make sure to call DeleteDocument if you do not call Preview.

Data Type: Action - executed when set to True



# PrintDialog Property

PrintDialog, when True, causes the print dialog box shown below, to be displayed when you are about to print. This can be from within the Preview window, or from a call to PrintDocument. The PrintDialog gives the user the opportunity to select the pages to print, adjust the printer offsets, or cancel printing. Click on the fields in the dialog shown below to view help topics for that field.



Data Type: True/False (BOOL)

# Landscape Property

Landscape instructs the printing logic to rotate the page to Landscape orientation for printing. This allows you to display your Landscape pages with the long edge of the paper horizontal on the screen, for easy reading. If you use this approach, make sure you set PaperWidth to the length of the paper (typically 11.0 inches, or other units), and the PaperHeight to the width of the paper (typically 8.5 inches, or other units). Landscape must be set prior to calling CreatePage.

Data Type: True/False (BOOL)

## HideButton Property

HideButton hides the PrintWorks control button on your form. Set HideButton to true if you do not want your code to respond to the ClickIn event. The ClickIn event will still be fired if the user clicks in the hidden button's area, so make sure you do not place any code in the ClickIn event unless you want to respond.

Data Type: True/False (BOOL)

# LineWeightR Property

LineWeight and LineWeightR specify the thickness of lines to be drawn. The LineWeight property specifies the thickness in pixels, while the LineWeightR property specifies the thickness in units (usually inches or cm). When using metric or other units than inches, it is recommended that LineWeightR be used to ensure line thicknesses will scale properly. Using a LineWeight (or, LineWeightR) of zero, draws a one pixel thick line, no matter what the display or printer resolution.

Data Type: Real (float)

## **PrintEventOnly Property**

PrintEventOnly, when True, does not invoke PrintWorks' printing logic when the Print ToolBar Button is clicked. It only fires the PrintEvent custom event. This gives you the opportunity to design your own printing logic.

Data Type: True/False (BOOL)

## DisplayPageNum Property

DisplayPageNum determines what page in your document is displayed, and how the control is displayed. If DisplayPageNum is zero, the control is a push button. If DisplayPageNum is greater than zero, the control becomes like a picture control, displaying the page within it. The page number must not be greater than the number of pages in the document. The page is actually displayed when the DrawNow action property is called. Typically, you would create the document and pages in your code at the appropriate time, and call DrawNow to display it.

Data Type: Integer (int)

# DrawNow Property

DrawNow causes the page specified by DisplayPageNum to be drawn in the control.

Data Type: Action - True/False (BOOL)

# DeskColor Property

DeskColor is the color of the preview window behind the page being previewed. Its default value is the Windows system desktop color, but you can set it to any color you like.

Data Type: RGBCOLOR (long)



# PictureScale Property

The PictureScale property determines the size of the image when the control is being used as a picture control. The default scale is 1.0, where the entire image is displayed. Increasing the scale, zooms in on the image. Scroll bars automatically appear when zoomed in (i.e. PictureScale > 1.0). If a PictureScale of less than 1.0 is used, the image will become smaller, and will be placed in the upper left portion of the picture control. This property can easily be hooked up to a scroll bar, or other type of slide control to enable the end user to adjust the scale. The demo program shows how to use control buttons to adjust the zoom level.

# Overview

PrintWorks is a Visual Basic custom control that greatly simplifies creating, previewing and printing documents of any kind. Documents are created as a group of one or more pages that may be viewed on the screen, or printed directly. PrintWorks provides custom properties and events for creating the content of the pages in the document, controlling the appearance of the preview window, and managing printing. If you are using [Visual Forms for Windows](#), PrintWorks adds the additional functionality of allowing your pages to have a base form, including a template for filling in the form with data. Properties are provided for creating stunning reports containing formatted text, graphics, and charts and graphs.

PrintWorks provides a built-in, highly functional [Preview](#) window. One line of code launches this window from within your applications, allowing users to scroll through the pages, zoom in or out, and print some or all of the pages. Documents may contain pages of different sizes and orientations. Print jobs can have mixed orientations. PrintWorks works directly with the Windows API printing functions, so you can print to any type of Windows printer, including a FAX. You may also preview individual pages in the control itself (similar to a picture control).

PrintWorks creates output by actually drawing it, not by using graphics controls or executing a form print. The latter two methods, which are available from within Visual Basic, drain resources, are difficult to create, display slowly, and produce poor quality output. PrintWorks uses the inherent graphics capabilities of the Windows API to create scalable metafiles to represent each page. A metafile is Windows internal device independent graphics file format. Output stored in this manner can be previewed or printed in the most efficient and highest quality possible. You can save each metafile to disk for later use, as an archive, or for other applications. It is an efficient way to save data.

PrintWorks uses a powerful Windows Dynamic Link Library (DOCMAN.DLL), written in C++, to process your documents. This results in very fast displaying and printing, because it communicates directly with the Windows API. In addition to managing documents, this DLL also contains a graphics library so you can easily create your output. The graphics and text action properties are modeled after traditional BASIC commands. In addition, the handle to the metafile device context is always available during creation of a page. This permits you to execute your own graphics functions, or access the Windows API directly from your code. Thus, PrintWorks is inherently extensible.

The PrintWorks can appear and function like a standard command button. When clicked, it fires the [ClickIn](#) event, where you place your code to construct your document, preview it, and/or print it. If you wish to connect your code to another event, command button, or sub-routine, you can hide the button using the [HideButton](#) property. This approach makes it possible to handle all use of PrintWorks through one, hidden control. The control can also appear like a picture control, and the page can be drawn directly in it.

## Documentation Conventions

PrintWorks is controlled by setting its various properties. Some of these properties are quantitative, such as [Resolution](#); and, other are action properties, such as [Preview](#). In this documentation, the action properties are often referred to as calls, because they act similar to procedure or function calls. In reality, that is what they are. Setting an action property to True, causes a function in the DLL to be called to perform the action. When you see the phrase "call [DeleteDocument](#)", for example, it means the same as "set the

DeleteDocument property to True".

All examples are given in Visual Basic code. The reason is this code is both correct for VB programmers, and a good pseudo code for Visual C++ (and other C\C++) programmers. C\C++ programmers should refer to the section Support For Visual C++ for details on using PrintWorks in other programming environments.

## **Demo Programs**

A Visual Basic demo program TEST1.EXE is provided to demonstrate the features of PrintWorks. The source code is provided in the file: TEXT1.FRM to show how the various programming tasks are accomplished. The demo program has a number of PrintWorks buttons that demonstrate different aspects of the control. The code attached to each control is commented and logically arranged to make it easy to understand. If you prefer, you can cut and paste code from these examples to save time. Each action property that is used to accomplish a particular task is immediately preceded by code that sets the quantitative properties that it depends on. Source comments are provided to explain what each section of code does. The source code for the demo is considered an integral part of the documentation for PrintWorks, and it is recommended you review it. In addition, it is recommended, you read the entire Introduction portion of the on-line help.

The demo program consists of 8 controls (at this writing). The names of the controls and the particular portion of PrintWorks that they demonstrate are as follows:

- DMWin1 - Line Objects, Direct API Call
- DMWin2 - Three-D Bar, Bitmaps, starting Preview close up
- DMWin3 - Three-D Pie
- DMWin4 - Filled Objects
- DMWin5 - Fonts & Text
- DMWin6 - Rotated Text, PrintEvent and PreviewClosing custom events
- DMWin7 - Multiple Pages, Forms, Templates, Data Fields, Metric Units, Landscape Orientation
- DMWin8 - Invisible Control (code is actually executed in the Command2 control's click event)

A second demo program TEST2.EXE demonstrates using the control as a picture box. Command buttons show how to zoom in and out, and how the scroll bars automatically appear when needed. This demo also shows how the PrintDocument property can be used with the control as a picture box to create your own custom print preview window.

Additional demo programs may be available. They will follow the naming convention of TESTx.EXE.

# Document Management

PrintWorks has a specific architecture for managing documents. It is both flexible and convenient. A document consists of one or more pages, which are created and composed in sequential order before printing or previewing. You use the following steps:

1. Create the document with the [CreateDocument](#) property
2. Create a page with the [CreatePage](#) property
3. Compose the page by adding text and/or graphics
4. Specify a form and/or template to attach to the page
5. Add data to the template's fields
6. Display the document
7. Delete the document using [DeleteDocument](#) (done automatically if the document is [Previewed](#))

Steps 2 through 5 may be repeated to add additional pages. Steps 4 and 5 are optional, and are only available if you are using Visual Forms for Windows to create background forms and/or data templates.

When a page is rendered, it is done so in three layers. The first, or bottommost layer is the page itself. This page is blank when the page is created in your code. It can ultimately contain whatever runtime text or graphics you decide to add. In addition, the data supplied for fields in a data template are added to the page layer at runtime using the [FillFieldID](#) and/or [FillFieldNumber](#) properties. When the page layer is drawn, it fills the entire logical page that you specify with the [PaperWidth](#) and [PaperHeight](#) properties prior to calling the [CreatePage](#) property.

The second layer is the form layer, which is a static metafile on disk. Typically you would create the form using [Visual Forms for Windows](#). However, it could be any valid Windows metafile. When the form layer is drawn, it also fills the entire logical page, and can be thought of as a full page overlay. If you create this layer using [Visual Forms for Windows](#), make sure the paper dimensions are consistent, so it will scale properly. If you create this metafile by some other means, keep in mind it will be scaled to fill the entire page.

The third layer is another disk based metafile, but it does not have to fill the entire page. You specify the position and size using the appropriate properties, and then call the [MergeMetaFile](#) property. Typically you would add graphics in the form of bitmaps; however, sometimes it is convenient to use a metafile produced by some other software, such as a graphing package, or possibly a scalable logo. PrintWorks automatically manages loading, merging and deleting from memory this metafile. You may add additional metafiles, using the handle to the metafile device context provided by the [hMetaDC](#) property. If you add a metafile, or any object in this way, you are responsible for cleaning up the memory afterward.

Pages are created in sequence. Creating a new page, closes the previous one. If you wish to add text or graphics to a page, do so while the page is open. Also, the [hMetaDC](#) property always applies to the currently open page. The last page in your document is closed automatically when you use the [Preview](#), [PrintDocument](#), or [DeleteDocument](#) properties. When the [Preview](#) is activated, it takes control of your application until it is closed. During this time you cannot modify any properties or pages in your document.

Automatic memory clean up is performed when [DeleteDocument](#) is called.

DeleteDocument is called automatically when the Preview window closes. If you don't call Preview, you must call DeleteDocument.

Pages are always displayed (on both the screen and printer) with the width parallel to the top of the screen or physical page. If you wish to display a landscape oriented page on the screen set the width to 11.0 and the height to 8.5 (for letter size in inches). Also, set Landscape equal to True, so the printer will rotate it when it is printed. This methodology makes it easy to create landscape documents without having to convert coordinates. Coordinates always are referenced to the upper left corner of the page, increasing from left to right, and top to bottom. If you mix orientations on the same page, make sure you compute your reference points (XPos, YPos, X2 and Y2) relative to the upper left corner of the page.

# Filling In Forms

PrintWorks provides properties for easily filling in forms that were created with Visual Forms for Windows. Template files (usually with the TF extension) contain specifications for placement of data fields on a form. PrintWorks takes these specifications and adds the text you specify to compose the data field.

The first step is to connect the template to the page. This is done by setting the TemplateFile equal to the file name of the template file prior to calling CreatePage. The template file may or may not contain text for the data fields (depending on whether you entered text in Visual Forms for Windows). If some data fields contain text, you may clear all the fields of data with the ClearFields property. Also, whenever new data is assigned to a field, the old text or data is first cleared. Using ClearFields will alert you to fields that you forgot to update in your code as they will be blank on the page.

The second step is to assign current data to each of the data fields you wish to display. You first identify the field with either the FieldNumber or FieldID property. Set the FieldData property equal to the text string you wish to print; and, use either FillFieldNumber or FillFieldID to actually place the data in the field. You may reference and fill a data field by either its numerical number or its ID or name. Make sure you use the proper pair (i.e. either FieldNumber and FillFieldNumber OR FieldID and FillFieldID). If you are unsure of the field's name and/or number, refer to the TXT file created with your template for a listing. FillFieldNumber is more efficient and faster, particularly if there are a lot of data fields.

PrintWorks keeps a list of all the data fields for the page, and their data. When the page is closed, the data is actually drawn on the page layer for later display. You may change the contents of the fields any number of times prior to closing the page.

# Drawing Text

Drawing text involves two basic operations: (1) defining and selecting a font; and, (2) calling the Paragraph property. PrintWorks font selection and text rendering only support TrueType fonts. If you wish to use other Windows fonts, use the hMetaDC property and the Windows API directly. The orientation, or angle that text is printed (portrait, landscape, etc.), is a function of the font, not the text. If you change angles or orientations, you must select a new font. To maximize the efficiency of your code, print all text for a single font at one time, rather than repeatedly changing fonts.

NOTE: It is not necessary to select a font for filling in data fields in a data template. The data template file contains all the information for the required font.

Text is always printed in fields. A field is an invisible bounding rectangle that contains the text. The lower left corner of the field is defined by XPos and YPos, and the width of the field is FieldWidth. The field always proceeds from left to right (as text is normally written) in the orientation of the text.

The Paragraph property handles all printing of text, even a single line, or character. Keep in mind that the five values available for the Justify property only work for text in portrait, landscape, reverse portrait or reverse landscape orientations. For text printed at any other angle (for example 45 degrees), always set the Justify property to zero (left justification). Also keep in mind that when Justify = 4 (decimal alignment), the decimal place is set at the XPos position.

# Line Drawing

The line drawing properties use XPos, YPos, X2 and Y2 to define the location of the object. The characteristics of the line being drawn are defined by the LineWeight (or, LineWeightR) and LineColor properties. The line weight, or thickness, can be expressed in one of two ways. The LineWeight property is expressed in pixels, with higher values resulting in heavier weight lines. When a line is rendered, the actual thickness is the number of pixels defined by LineWeight. The LineWeightR property is expressed in units (inches, cm, etc.), with higher values also resulting in heavier weight lines. When a line is rendered, the actual thickness is computed using the page's Resolution. LineWeightR is more versatile because it is resolution independent. If you are using metric units, use LineWeightR to get a properly weighted line. In either case, a line weight value of zero results in a one pixel thick line.

Colors can be especially tricky when dealing with lines, particularly thin lines. Some colors require dithering (mixing of colors, including in some cases, the white background). It is possible for a thin, dithered line to not show up at all because it merely blends with the background. Try to select only line colors that are solid, to avoid this pitfall.



# Area Fills

Area fills are enclosed figures of various shapes. They are defined by their geometric properties, XPos YPos, X2, Y2, Radius, Aspect, etc. and by their colors and pattern. All area fills have a FaceColor, a BackgroundColor, an EdgeColor and a Pattern. The face of the figure using the FaceColor and the Pattern (either solid, or one of 6 pre-defined patterns). If the background mode is transparent (from a call to MakeTransparent), the BackgroundColor is ignored. If the background mode is opaque (from a call to MakeOpaque), the background color is used to fill in between the lines in the pattern.

The EdgeColor is used to draw a border around the figure. The thickness of the line is determined by LineWeight (or, LineWeightR).

# Printing

Printing may be handled in one of several ways. The Preview window has printing support built in. Simply click on the Print Toolbar Button. You may also print directly from your code without invoking the Preview window. You may also do a combination of both. A custom event, called PrintEvent, is fired when the Print Toolbar Button in the Preview window is clicked. You may add code to this event to perform whatever pre-printing task you like, including doing something totally un-related to printing. You can use the PrintAll, FirstPrintPage and LastPrintPage properties to set up the Print Options dialog. You may even bypass the built in printing logic altogether.

# Previewing

Previewing is done via the Preview window. Simply call the action property Preview after composing your document. The Preview window permits scrolling through the pages in your document, zooming in and out, and printing. It provides considerable functionality with just one line of code. Properties are provided to customize the appearance of the Preview window.

A page may also be displayed in the control itself, rather than the Preview window, by setting DisplayPageNum equal to the page number, and calling DrawNow.

# Object Sizing & Positioning

These properties control the size and position of graphical objects and text fields.

# Three-D Objects

These objects are useful for constructing three dimensional charts and graphs.

# Object Appearance

These properties control the colors, patterns and line properties of objects.

# Adding Graphics

Graphics may be added in two ways. Each page may have one Windows metafile placed at any point on the page. In addition, you may add any number of Windows device independent bitmaps (usually having the DIB or BMP extension). These may be positioned and sized independently. Bitmaps are good for displaying logos, graphs, charts, etc. A number of graphics programs and controls provide output in the form of bitmaps, that can be easily incorporated into PrintWorks.

NOTE: It is technically possible to save a device dependent bitmap to disk, and some software may do this. Only device independent bitmaps are recognized and displayed by PrintWorks. This is to ensure that the pages in the document are completely device independent, so they will display and print correctly on all devices.

## File Distribution

PrintWorks requires two files to operate in runtime mode: (1) the VBX file: DMWIN.VBX; and, (2) the DLL: PWDLL.DLL. Both of these files should be distributed with your application. An additional license file: DMWIN.LIC, is required for design time support (Visual Basic, Visual C++, and other development environments). This file identifies you as a PrintWorks licensee, and MUST NOT be distributed.

All other files supplied on the distribution disk, including form files, template files, font lists, metafiles, bitmaps and this help file may not be distributed, unless specifically authorized. It is suggested the VBX and DLL files be installed on your user's \WINDOWS\SYSTEM directory. The LIC file should be installed on the local directory for your development environment, or on your \WINDOWS\SYSTEM directory. If you get a message when loading PrintWorks that the license file was not found, make sure it is in a directory where your development environment can find it.



# Integrating With Visual Forms for Windows

Visual Forms for Windows (VFW) is an interactive form design program also marketed by Bytech Business Systems, Inc. This program is useful for rapidly designing base forms for documents. PrintWorks is designed to support the metafiles produced by VFW. You must simply set the FormFile property to the file name of the form's metafile, and set the MergeForm action property to True while constructing a page. The form will then be printed with that page of your document. You must distribute the metafile for the form (WMF file, NOT the VFW file) with your application as well as the font list (TFL file) that pertains to the form. If you plan to have many different forms in a project, it is recommended that you use a common font list file to minimize the number of files to be managed. You must also be sure your users have the TrueType fonts required by your forms and documents. If some fonts are not found, Windows will make a substitution. Experience has shown these substitutions are usually not satisfactory.

VFW also permits the creation of Template Files (usually with the TF extension). A template file is a subset of the objects in the form, including only text fields that were defined in VFW as data fields. The text in the data field text objects are not included in the form's metafile, in order that the contents of that field may be determined at runtime. PrintWorks provides properties for specifying a template file and dynamically setting the text data at runtime.

# Using Metric and Other Units

PrintWorks is based on standard Windows metafiles, which are scalable. As such, they are somewhat independent of units of measure, since their size depends on the number of pixels available in the display device. When the documents are rendered on a display device (either a screen or printer) it is scaled to fill the available pixels. Since different displays can have a wide range of resolutions (and, therefore available pixels) PrintWorks uses inches (or other units you choose) for the properties that relate to position and size of the various graphics and text objects, to allow the documents to be device independent. The units of measure are related to pixels through the resolution property that is specified when creating a page.

Throughout this help file, by convention, these dimensions are referred to as inches. A typical 8.5 x 11.0 inch page, with a 1/4 inch border, at 300 dots/inch resolution, has 2400 x 3150 pixels available. The number of pixels is constant since it is a property of the device. To use different units, the resolution property for each page should be adjusted accordingly. For example, if centimeters are used, the page height and width would be 2.54 times the value in inches. Therefore, it would be necessary to divide the resolution property by 2.54 to maintain the proper number of pixels.

In addition to adjusting the resolution property, there is one other area to be aware of: the size of a font. Font size (or, height) is always represented in Points, whatever the units of measure. One point is 1/72 of an inch; or, there are 72 points per inch. If you are not sure how to determine the point size of a font, it is recommended you experiment, starting with 10 to 12 points for normal text.

Example:

Inches:

Width, inches: 8.0  
Height, inches: 10.5  
Resolution: 300  
Width, pixels: 2400  
Height, pixels: 3150  
12 Point font size: 12

Centimeters:

Width, cm: 20.32  
Height, cm: 26.67  
Resolution: 118  
Width, pixels: 2400  
Height, pixels: 3150  
12 Point font size: 30.5

# Support for Visual C++

PrintWorks supports Visual Basic Level 1 compatibility. That means that it is fully supported by Visual C++ and other programming environments that support Level 1 VBX's. To access the design time properties of PrintWorks, use the "Styles" section of the properties dialog in App Studio. To access the runtime properties, use the CVBControl class of the Microsoft Foundation Class library. The example below shows how to use the CVBControl class to set or retrieve the resolution property.

Example:

```
CVBControl*    pPrintWorks1 = GetDlgItem(IDC_PRINTWORKS1);

    pPrintWorks1->SetNumProperty("Resolution", 300L);    // This function expects
a "long"

    int    reso = (int) pPrintWorks1->GetNumProperty("Resolution"); // This function
returns a "long"
```

None of the properties supported by PrintWorks are arrays, so the array parameter in SetNumProperty and GetNumProperty may be ignored.

## **Close ToolBar Button**

Click the Close (or Exit) ToolBar button or open the system menu to exit the Preview window. The PreviewClosing event is fired when the Close button is clicked.

## **Print ToolBar Button**

Click the Print ToolBar button to invoke the Preview windows printing logic. As soon as the button is clicked, the PrintEvent event is fired. If PrintEventOnly is True, control is returned to the Preview window immediately after returning from the PrintEvent handler. If PrintDialog is True, the Print Options Dialog is displayed after returning from the PrintEvent event handler.

## **Zoom Scale ToolBar Button**

Enter a Zoom In value in the edit box to set the scale for zooming in on a page. The scale is in percent and can be in the range of 10 - 100 (a pixel for pixel representation, based on the Resolution property). If you select a scale that will display a page smaller than full page, the scale will default to 100 percent.

## Full Page ToolBar Button

Click the Full Page button to view the page in Full Page View. Full Page View can also be activated by clicking the left mouse button in the screen's client area when Zoomed In.

## **Zoom In ToolBar Button**

Click the Zoom In ToolBar button to view the page close up. Clicking the left mouse button in the preview window during Full Page view, also activates the zoom in button.



## **PageUp ToolBar Button**

Click the PageUp ToolBar button to view the previous page in the document. If the document has only one page, this button is disabled.

## **PageDown ToolBar Button**

Click the PageDown ToolBar button to view the next page in the document. If the document has only one page, this button is disabled.

## **Ok Print Option**

Printing commences when this button is clicked.

# Cancel Print Option

Printing is cancelled when this button is clicked.

## ClickIn Event

The ClickIn event is fired when a left mouse click is made over the PrintWorks button. This event is fired even if the HideButton property is True. You would typically place code for creating documents in the ClickIn event handler.

# PrintEvent Event

The PrintEvent event is fired whenever the Print ToolBar Button is clicked from the Preview window. You may modify the Preview window's printing logic by placing your own custom code in the PrintEvent event handler.

# PreviewClosing Event

The PreviewClosing event is fired when the Preview window is about to close. If you wish to execute code at this time, such as special memory clean up (or destroying objects in C++), add that code to the PreviewClosing event handler.

# Visual Forms for Windows

Visual Forms for Windows is a graphical program for interactively designing forms for reports. It works similar to many paint or draw programs, but is specifically designed for developing custom forms and base sheets for reports. Visual Forms for Windows has an integrated data field template generator, so you can graphically add data fields to your forms or reports.

Visual Forms for Windows is totally supported by PrintWorks, and makes it possible to save many hours and lines of code when developing the static (base form) portion of your documents, and their data templates.



# Print Copies

Enter the number of copies of each page to print.

**Z**

