Printing with Preview
D. P. Baird
October 18, 2000

The Printer unit provided with Delphi supports a basic set of printer functions. It does not provide access to all the Windows API functions. Neither does it support the preview of documents being printed. TPrintPreview utilizes the TPrinter capabilities together with additional Windows API functions and provides a document interface expressed in inches together with the option of previewing a document.

TPrintPreview borrows heavily from James "Woody" Woodward's TPrintObject implementation. It modifies the printing interface and extends the capabilities provided by Woody:

- To manage paper sizes,
- To add events to respond to an error in setting margins and to a change of printers using the print dialog from the preview screen, and
- To restructure some properties and methods to provide an interface more like that found in a word processor and to provide consistency with the terminology in the Windows API.

TPrintPreview provides a set of resources that can be used display and print a document. Some properties and methods are used to manage the resources while others are used in specifying a document. The occurrence of the methods Start and Finish in a program delimit the specification of a document. The methods used in specifying a document are termed commands.

When a program is specifying a document, TPrintPreview creates a data structure that is a list of pages, where each page contains a list reflecting the commands that are used to format and print the page. This data structure is used to both display and print the document.

TPrintPreview supports the following optional resources:

- A preview form.
- A progress bar form with an optional Cancel button that can be displayed while specifying a document and while printing it.
- A TPrintDialog that can be displayed after the document has been specified but before it is printed.
- Events that permit the application to specify "appendages" that can be called at particular points in the TPrintDialog code to provide special formatting that is not handles by commands.

These options can be chosen at design time or must be specified at run time as long as it is done before executing the Start method that marks the beginning of a document specification.

Printer Unit

Delphi's Printer Unit provides the interface to Window's printing *capabilities*. It provides the following:

1. The *TPrinter* object encapsulates the Windows API interface.
2. The *Printer* function returns a global instance of *TPrinter* to manage interaction with the printer. If an instance of *TPrinter* does not exist, one is created using the default printer.
3. The *SetPrinter* function returns current *TPrinter* object and makes the specified *TPrinter* object the current *TPrinter*. It is the caller's responsibility to free the old *TPrinter*, if appropriate. (This allows toggling between different *TPrinter* objects without destroying configuration settings.)
4. The *AssignPrn* procedure that is used for text printing with *Write* and *WriteLn* procedures.
5. The local variable *FPrinter* of type *TPrinter* provides the real interface to the Windows API. The functions *Printer* and *SetPrinter* set *FPrinter*.

While *TPrinter* exposes the parameters used to print, it does not expose all the parameters needed to layout a page. The *Windows* unit supports the *GetDeviceCaps* function in the Windows API that provides access to the following device-specific information:

1. PhysicalWidth – Physical width of the paper in pixels.
2. PhysicalHeight – Physical width of the paper in pixels.
3. PhysicalOffsetX – Physical left margin in pixels.
4. PhysicalOffsetY – Physical top margin in pixels.
5. LogPixelsX – Pixels per inch horizontal.
6. LogPixelsY – Pixels per inch vertically.

The *Printer* unit includes *GetPrinter* and *SetPrinter* methods that allow setting the paper size to both sizes supported by the printer driver as well as custom sizes supported by the driver.

Printing Dialogs

The standard dialogs are provided that support the printer function:

1. *TPrintDialog* – that can be used when sending jobs to the printer to select a printer and the pages to be printed.
2. *TPrinterSetupDialog* – used to select the printer and the paper size, source and orientation.
3. *TFontDialog* – used to select a font.

The first two dialogs use the *Printer* function directly get and store information. The third makes a font name, style and size specification available to the program. The programmer is responsible for the correct use of the information supplied through the use of the dialogs.

Warning: If both a *TPrinterSetupDialog* dialog and the *SetPageSize* method are used, it is possible that *SetPageSize* could define a page size for one printer that is not valid for a different printer available to the program. In some cases, the subsequent selection of that printer with a *TPrinterSetupDialog* dialog could generate an exception.

Progress Bar

An optional progress bar is available during document preparation and printing. An associated optional cancel button is also available. The published properties ShowProgress and ShowCancel control their availability. The default values are True.

The published properties ProgressBarMax and ProgressBarInc control the operation of the progress bar. ProgressBarMax specifies the upper limit of the range of possible positions with a default value of 100. ProgressBarInc specifies the amount to increment the progress bar using the method Increment with a default value of 1. The method IncBar can be used instead of Increment and specifies the amount of the increment to be used.

The Running property is set False when the Cancel button is chosen. Running is set True by the Start method. The following code could be inserted at the bottom of the loop used to print a report to either step the progress bar or terminate report preparation.

```
if Running then
  Increment
else
  Break;
{fi}
```

If ShowProgress is assigned a value in the program, this should be done before executing the Start method.

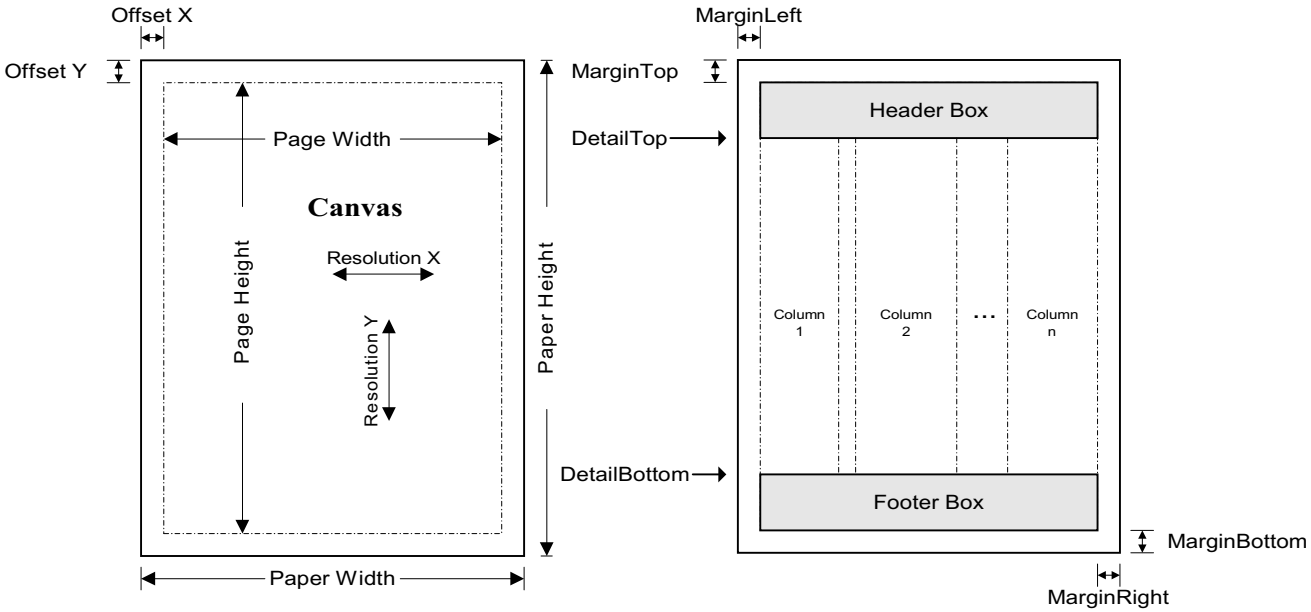
Design Approach

The overall approach is based on James "Woody" Woodward's *TPrintPreview*. According to Woody, the set of functions is based on an article from the Inprise web site however a specific reference is not given. I have customized the specific set of functions.

Managing Paper

Printing from *TPrinter* is based on *pixels*. However, to achieve "device independence" *inches* are used as the unit of measure. For some properties and methods definitions may be made using both pixels and inches.

The figure on the left below shows the parameters that are available through *TPrinter* or which can be obtained using *GetDeviceCaps.* The figure on the right shows the form layout parameters supported by this component.



The term *Canvas* is used in Delphi to refer to the area where text and graphics can be displayed or printed. The location for text and graphics is specified relative to the upper left-hand corner of the canvas. Text and graphic images that extend beyond the right or bottom edge of the canvas are not displayed or printed.

The mechanical structure of printers determines the offsets and page dimensions. The following table shows three examples for printers using letter size paper.

Printer	Offset X	Offset Y	Page Width	Page Height	Offset Right	Offset Bottom
Panasonic KX-P2130	0.25	0	8	11	0.25	0
HP LaserJet 5L	0.25	0.2	8	10.6	0.25	0.2
EPSON Stylus COLOR 640	0.1166… (7/60)	0.1166… (7/60)	8.2666… (8 16/60)	10.3333… (10 20/60)	0.1166… (7/60)	0.55 (33/60)

The Panasonic is a dot matrix printer capable of using continuous forms. The offsets for the LaserJet are symmetric. The EPSON Stylus has different offsets at the top and bottom.

The four *Margin* properties are used as reference points for positioning text and figures. The remaining items are associated with printer commands that are stored and interpreted when displaying or printing information. Initially, margins are set the offset values. Detail top and bottom are forced to be within the top and bottom margins.

In the figure above, the definition of parallel columns is illustrated. Overlapping columns can be defined where the program only uses a subset of non-overlapping columns at any one time.

Writing Lines

Several paradigms are supported:

1. Writing lines of text aligned to the left or right margins or centered between the margins.
2. Writing in up to 20 columns. A column can be positioned anywhere across the width of the page.
3. Writing text at a designated position on the page with a specified style.
4. Writing consecutive lines in the detail area with automatic paging.

Graphic support

Bitmaps (.bmp) and metafiles (.emf) files can be printed. Since virtual memory is used to store the contents of the document and its format, there is a limit on the space available to store images. The facility is oriented more towards adding logos to the report than printing a large number of photographs.

Reference

The *TPrintPreview* component provides two forms. The first is used to preview the report and the second to display a *TProgressBar* when preparing or printing a report. The *Preview* property controls the use of the preview form and the *ShowProgress* property controls the use of the progress bar.

TPrintPreview

Unit: PrintDrv — Palette: System

TObject•TPersistent•TComponent•TPrintPreview

TPrintPreview provides an interface to the printer that includes a print preview capability.

Description The methods *Start* and *Finish* are used to bracket a set of commands that specify the format and content of a document that is to be printed. A print preview option is available that permits the document to be viewed before printing.

Propertries

The following table lists the properties of *TPrintPreview*.

Property	Visibility	Default	Documented in
AutomaticPaging	published	False	TPrintPreview
CharPerLine	public, read-only		TPrintPreview
ComObject	public, read-only		TComponent
ComponentCount	public, read-only		TComponent
ComponentIndex	public		TComponent
Components	public, read-only		TComponent
ComponentState	public, read-only		TComponent
ComponentStyle	public, read-only		TComponent
DesignInfo	public		TComponent
CharPerLine	public, read-only		TPrintPreview
FontName	public, read-only		TPrintPreview
FontSize	public, read-only		TPrintPreview
FontStyle	public, read-only		TPrintPreview
HelpContext	published	0	TPrintPreview
LastXPosition	public		TPrintPreview
LineHeightInches	public, read-only		TPrintPreview

Property	Visibility	Default	Documented in
LineHeightPixels	public, read-only		TPrintPreview
LinesInDetailArea	public, read-only		TPrintPreview
LinesLeft	public, read-only		TPrintPreview
LinesPerPage	public, read-only		TPrintPreview
Name	published		TComponent
Orientation	published	poPortrait	TPrintPreview
Owner	public, read-only		TComponent
PageNumber	public, read_only		TPrintPreview
PaperHeight	public, read_only		TPrintPreview
PaperSize	published	PtrSU	TPrintPreview
PaperWidth	public, read_only		TPrintPreview
Preview	published	True	TPrintPreview
ProgressBarInc	published	1	TPrintPreview
ProgressBarMax	published	100	TPrintPreview
Running	public	True	TPrintPreview
ShowCancel	published	True	TPrintPreview
ShowPrintDialog	published	True	TPrintPreview
ShowProgress	published	True	TPrintPreview
StartPercent	published	P2_Fifty	TPrintPreview
Tag	published	0	TComponent
TheYPosition	public		TPrintPreview
Title	published		TPrintPreview
Transparent	published	True	TPrintPreview
VCLComObject	public		TComponent

AutomaticPaging

Indicates automatic paging mode

```
AutomaticPaging: Boolean
```

Description

Setting *AutomaticPaging* to *True* will force a new page if printing extends beyond the boundaries of the detail area. This setting is useful when printing many lines of text such as *TMemo* contents or *TStrings* and the number of lines of text is not know or doesn't matter.

When a text printing command is executed, it determines whether that text extends below the bottom of the detail area. If it does and *AutomaticPaging* is set to *True*, the page headers and footers are printed if they were set, the *OnNewPage* event is called, and a new page is issued.

To control all paging explicitly, set this property to *False*. The *GetLinesLeft* function can be used to determine if enough printable area remains to print in or if a new page should be issued manually using *NewPage*.

CharPerLine

CharPerLine returns “character” positions across a form.

```
CharPerLine: Integer;
```

Description

Computes the number of copies of the letter “B” that can be printed within the left and right margins. This is most useful with a fixed spaced font.

FontName

Returns the name of the font currently active.

FontName: **string**

Description The name of the font currently being used to print text is returned.

Note This property is read only.

FontSize

Returns the size of the font currently active.

FontSize: Word

Description The size of the font currently being used to print text is returned.

Note This property is read only.

FontStyle

Returns the style of the font currently active.

FontStyle: TFontStyle

Description The style for the font currently being used to print text is returned. See Delphi help for *TFontStyle*.

Note This property is read only.

HelpContext

Provides a context number for use in calling context-sensitive online Help while previewing

.

HelpContext: THelpContext

Description When the preview form is created, its HelpContext property is set to the value assigned.

See also *TWinControl.HelpContext* property.

LastXPosition

The X Position after the last write command

LastXPosition: Single

Description Returns the horizontal value, in inches, of the pen position after execution of one of the *Write* methods. It may also be set before the execution of *WriteText* with an *X* value of "wtNextX".

See also *WriteText* method.

LineHeightInches

Returns the line height of the current font

LineHeightInches: Single;

Description The return value is in inches and it reflects the line height using the current font size.

See also *LineHeightPixels* property

LineHeightPixels

Returns the line height of the current font

LineHeightPixels: Word;

Description The return value is in pixels and it reflects the line height using the current font size.

See also *LineHeightInches* method

LinesInDetailArea

Returns the number of lines in the detail area.

```
GetLinesInDetailArea: Word;
```

Description *LinesInDetailArea* calculates and returns the number of lines that can be printed within the detail area set by *SetDetailTopBottom*. The number of lines will vary based on the font in use.

See also *SetDetailTopBottom* method, *LinesLeft* property, *LinesPerPage* property

LinesLeft

Returns the number of lines remaining in the detail area.

```
LinesLeft: Word;
```

Description *LineLeft* returns the number of lines left that can be printed in the detail area given the current font size.

See also *SetDetailTopBottom* method, *LinesInDetailArea* property, *LinesPerPage* property

LinesPerPage

Returns the number of lines available on the page.

```
LinesPerPage: Integer;
```

Description *LinesPerPage* returns the total number of lines printable on the form between the top and bottom margins using the current font settings.

Note *LinesPerPage* together with *CharPerLine* indicate the capacity of the form to print using fixed space fonts.

See also *ColumnsPerLine* property

Orientation

Specifies the orientation printing on the paper.

```
Orientation: TPrinterOrientation
```

Description The Orientation property is used to tell the TPrintPreview to print in either Portrait (default) or Landscape mode. This setting can only be used to affect the entire print job. It cannot be used to set the page orientation for a particular page or pages.

Note The two values for Orientation are poPortrait and poLandscape, defined in the Delphi Printers unit.

PageNumber

Returns the current page number.

```
tPageNumber:Integer;
```

Description *PageNumber* returns the value of the current page number.

See also *SetHeaderInformation* method, *SetFooterInformation* method

PaperHeight

Returns the height of the paper currently specified for the printer.

```
PaperHeight: Single
```

Description Returns the height of the paper in inches.

Note This property is read-only.

PaperSize

Specifies the paper size to be used.

PaperSize: PaperSizes

Description When *PaperSize* is set to "Letter" or "Legal", the method *Start* sets the printer for 8½ x11 and 8½ x14 inch paper respectively.

When set to "PtrSU", the value assigned to the printer is used. The method *SetPaperSize* and the *TPrinterSetup* dialog are used to change the default paper size.

The values for *PaperSize* are:

Value	Meaning
PtrSU	The paper specified for the printer is used
Letter	Use 8½ x 11 inch paper.
Legal	Use 8½ x 14 inch paper.

See also *SetPaperSize* method.

PaperWidth

Returns the width of the paper currently specified for the printer.

PaperWidth: Single

Description Returns the width of the paper in inches.

Note This property is read-only.

Preview

Controls the display of print preview.

Preview: Boolean

Description The *Preview* property tells the *TPrintPreview* object whether or not you want to display a preview of the report is to be displayed or whether it is to go directly to the printer.

Setting this property to *True* will display the preview screen and allow the user to browse through the report for a visual inspection. The preview screen can scale the view from 40% to 100% in increments of 10%. The *StartPercentage* property specifies the scale factor to be used. There is a Print button on the preview form, which allows the user to print directly without having to cancel and run the report again.

A navigation control allows the user to cycle through the pages of the report or type in a particular page number from 1 to the total pages.

See also *StartPercentage* property

ProgressBarInc

Specifies the amount to increment the progress bar.

ProgressBarInc: Word

Description When using the *Increment* procedure to update the progress bar, the amount of increase is stored in the ProgressBarInc property.

See also *Increment* procedure, *ProgressBarMax* property.

ProgressBarMax

Specifies the upper limit of the range of possible positions

ProgressBarMax: Word

Description Normally, a progress indicator is used to show a percentage so the default value for ProgressBarMax is 100. However, you can choose the maximum setting for the progress bar can be choosen by supplying a different value in the ProgressBarMax property.

See also *Increment* procedure, *ProgressBarInc* property.

Running

Must be *True* to continue preview or printing.

Running: Boolean

Description When *ShowProgress* is *True* a progress indicator is displayed when preparing or printing a report. When *ShowCancel* is also *True*, the progress indicator includes a *Cancel* button. When the cancel button is chosen, *Running* is set *False*. Otherwise it is *True*. It can be tested by a program using TPrintPreview to cancel printing.

In the following, the text file "Readme.txt" is printed. The property *Running* is tested and, if *False*, the program breaks out of the do while loop

```
AssignFile(tFile,ExtractFilePath(ParamStr(0))+'Readme.txt');
Reset(tFile);
SetTopOfPage;
NextLine;
while not Eof(tFile) do begin
  if Running then begin
    Readln(tFile,sLine);
    WriteLine(wtLeft,0.5,wtNextY,sLine);
    Increment{Bar};
    end
  else
    Break;
  {fi}
  end;
{od while}
CloseFile(tFile);
```

See also *ShowProgress* property, *ShowCancel* property.

ShowCancel

Displays the cancel button with the progress indicator.

ShowCancel: Boolean

Description When *True*, a cancel button is visible along with the progress indicator when *ShowProgess* is also *True*. The default value is *True*.

See also S*howProgress* property.

ShowPrintDialog

Displays a *TPrintDialog* object when the print button is chosen on the preview screen.

ShowPrintDialog: Boolean

Description After storing the initial pass of the report being created, if the *ShowPrintDialog* property is *True*, a printer dialog box will allow the user to select a different printer and also gives them the option of printing the entire report of just a selection of consecutive pages.

See also *AfterPrinterChange* event.

ShowProgress

Displays a progress indicator.

ShowProgress: Boolean

Description If you want the *TPrintPreview* to display a progress indicator while preparing the reprt and during the printing cycle, then you must set the *ShowProgress* to *True*. If this value is *False*, then no indication is given to the user that printing is taking place.

StartPercent

Sets the initial zoom percentage for preview.

StartPercent: Percentages

Description Normally *StartPercent* is set at design time and specifies the scaling to be used in displaying the report. *StartPercent* is assigned one of the following values.

Value	Meaning
P1_Forty	Scale to 40% of paper size.
P2_Fifty	Scale to 50% of paper size.
P3_Sixty	Scale to 60% of paper size.
P4_Seventy	Scale to 70% of paper size.
P5_Eighty	Scale to 80% of paper size.
P6_Ninety	Scale to 90% of paper size.
P7_Full	Scale to 100% of paper size.
P8_Width	Scale to paper width.
P9_Page	Scale to paper size.

TheYPosition

The current vertical pen position.

TheYPosition: Single

Description Returns the current vertical pen position, in inches. When set, it establishes the vertical pen position. When assigning a value, there is no checking that it lies within the printable area of the canvas.

Title

The document identification that appears in the Print Queue.

Title: **string**

Description When printing in Windows, displaying the printer's status dialog shows a list of print jobs for that printer. If you want a special name to appear here when printing your report, set the text that you want to appear in the *Title* property.

Transparent

Facilitates printing “watermarks.”

Transparent: Boolean

Description In order to use a graphic as a background image such as a watermark, the text that is printed over it must not block out the graphic that is below it. Setting *Transparent* to *True* will cause the brush style to be set to bsClear. This will print the text over any underlying graphics in a transparent manner allowing it to show through.

Events

The following table lists the events in *TPrintPreview*.

Event	Visibility	Documented in
AfterNewPage	Published	TPrintPreview
AfterPrinterChange	Published	TPrintPreview
OnMarginError	Published	TPrintPreview
OnNewpage	Published	TPrintPreview

AfterNewPage

The event *AfterNewPage* occurs after advancing to the next page.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property AfterNewPage: TNotifyEvent
```

Description The *AfterNewPage* event is called after advancing to the next page. This allows a procedure assigned to it to be called for on all but the first page. *TheYPosition* is set to one line above the detail area before the event is called.

The parameter *TPrintPreview* is passed to the procedure assigned to *AfterNewPage*. This reference can be used to print any type of information that is desired on each subsequent page of the report. A good example of this is to define a continuation heading.

See also *OnNewPage* event, *NewPage* method, *AutomaticPaging* property

AfterPrinterChange

The event *AfterPrinterChange* occurs after the print dialog closes and the printer has been changed.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property AfterPrinterChange: TNotifyEvent
```

Description When the *ShowPrintDialog* property is *True*, choosing the *Print* button on the preview screen causes a print dialog box to be displayed. *AfterPrinterChange* is called after the print dialog box closes if the printer was changed.

The new printer could have different offsets and a different page size that were used when the report was originally formatted. The procedure assigned to *AfterPrinterChange* can test for the validity of the formatting and correct any problems found or give the operator the option of continuing or canceling printing. When Cancel is chosen, the parameter *Running* is set *False* to prevent processing.

See also *ShowPrintDialog* property, *TPrintDialog* dialog

OnMarginError

The event *OnMarginError* occurs when one or more of the parameters of *SetMargin* fall outside the printable area of the paper.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnMarginError: TNotifyEvent
```

Description The *OnMarginError* event is called from *SetMargins* when it detects that one or more of the margins are outside the printable area of the paper. This allows a procedure to be called that post messages, takes actions and, possibly, makes corrections.

If no procedure is assigned, a message dialog is displayed that allows the operator to signal whether or not a correction should be made or whether printing should be canceled.

See also *SetMargins* method

OnNewPage

The event *OnNewPage* occurs before going to the next page.

```
type TNotifyEvent = procedure(Sender: TObject) of object;

property OnNewPage: TNotifyEvent
```

Description The *OnNewPage* event is called before going to the next page. This allows a procedure assigned to it to be called for the first page and all subsequent pages. There is no need to call *NewPage* after printing the final page of your report because the *Finish* method will call *OnNewPage* as well as any other page specific commands needed when a page is completely printed.

The parameter *TPrintPreview* is passed to the procedure assigned to *OnNewPage*. This reference can be used to print any type of information that is desired on each page of the report. A good example of this is to place a company logo on each page.

See also *AfterNewPage* event, *NewPage* method, *AutomaticPaging* property

Methods

The following table lists the methods in *TPrintPreview*.

The methods *Start* and *Finish* delimit the section of the program that prepares a report. Certain methods should only executed while the *TPrintPreview* is in "prepare mode." (Internally, this is the period when *DryRun* is *True*.) On the table, the column labeled "Prep" indicates the methods that can be executed in this mode. A "C" indicates that the method is a print *command* that stores information for use while previewing and/or printing the report. The "X" indicates methods that can be meaningfully executed in this mode.

When the *Finish* method is executed, the *TPrintPreview* enters a "Preview/Print" mode that interprets the information stored by the commands. The events listed above occur while in this mode. The column Labeled "Pr/Pr" indicates methods that can be executed in the event handlers that may be defined. Where a method can be a *command* in prepare mode and also used in an event handler, care must be taken to insure that actions taken in an event handler are not cancelled out by something that was stored during prepare mode. For example, *SetHeaderInformation* could be executed in prepare mode or by the *OnNewPage* and *AfterNewPage* event handlers.

When the *Preview* property is *True* and in prepare or preview mode, commands reference the canvas in the preview form. When the *Preview* property is *False*, commands reference *Printer.Canvas*. It is anticipated that most programming will be done in terms of inches or lines. However, measurements can also be made in terms pixels. The program must be sensitive to the canvas being used and the vertical and horizontal resolution of screen or printer being referenced.

The column labeled "Any" indicates methods that can be executed at any point in a program where they are meaningful. These methods either are used to manage a *TPrintPreview* instance or reference the underlying printer.

Method	Visibility	Documented in	Any	Prep	Pr/Pr
Abort	public	TPrintPreview		X	X
AfterConstruction	public	TObject			
Assign	public	TPersistent			
BeforeDestruction	public	TComponent			
ClassInfo	public	TObject			
ClassName	public	TObject			

Method	Visibility	Documented in	Any	Prep	Pr/Pr
ClassNameIs	public	TObject			
ClassParent	public	TObject			
ClassType	public	TObject			
CleanupInstance	public	TObject			
Create	public	TPrintPreview	X		
CreateColumn	public	TPrintPreview		X	
DefaultHandler	public	TObject			
Destroy	public	TPrintPreview	X		
DestroyComponents	public	TComponent			
Destroying	public	TComponent			
Dispatch	public	TObject			
DrawBox	public	TPrintPreview		C	X
DrawBoxShaded	public	TPrintPreview		C	X
DrawLine	public	TPrintPreview		C	X
DrawTextBox	public	TPrintPreview		C	X
ExecuteAction	public	TComponent			
FieldAddress	public	TObject			
FindComponent	public	TComponent			
Finish	public	TPrintPreview		X	
FixMargins	public	TPrintPreview		X	X
Free	public	TObject			
FreeInstance	public	TObject			
FreeNotification	public	TComponent			
FreeOnRelease	public	TComponent			
GetColumnLeft	public	TPrintPreview		X	X
GetColumnRight	public	TPrintPreview		X	X
GetInterface	public	TObject			
GetInterfaceEntry	public	TObject			
GetInterfaceTable	public	TObject			
GetMargins	public	TPrintPreview		X	X
GetNamePath	public	TComponent			
GetPaperParms	public	TPrintPreview	X		
GetParentComponent	public	TComponent			
GetPixelsPerInch	public	TPrintPreview		X	X
GetPixelsPerPage	public	TPrintPreview		X	X
GetTextWidthInches	public	TPrintPreview		X	X
GetTextWidthPixels	public	TPrintPreview		X	X
HasParent	public	TComponent			
IncBar	public	TPrintPreview		X	X
InchesToPixelsX	public	TPrintPreview		X	X
InchesToPixelsY	public	TPrintPreview		X	X
Increment	public	TPrintPreview		X	X
InheritsFrom	public	TObject			
InitInstance	public	TObject			
InsertComponent	public	TComponent			
InstanceSize	public	TObject			
LinesToPixels	public	TPrintPreview		X	X
MethodAddress	public	TObject			
MethodName	public	TObject			

Method	Visibility	Documented in	Any	Prep	Pr/Pr
NewInstance	public	TObject			
NewLines	public	TPrintPreview		C	
NewPage	public	TPrintPreview		C	
NextLine	public	TPrintPreview		C	
PixelsToInchesX	public	TPrintPreview		X	X
PixelsToInchesY	public	TPrintPreview		X	X
PrintGraphic	public	TPrintPreview		C	
RemoveComponent	public	TComponent			
RemoveFreeNotification	public	TComponent			
RestoreCurrentFont	public	TPrintPreview		C	X
SafeCallException	public	TComponent			
SaveCurrentFont	public	TPrintPreview		X	X
SetDetailTopBottom	public	TPrintPreview		C	X
SetFontInformation	public	TPrintPreview		C	X
SetFontStyle	public	TPrintPreview		C	X
SetFooterDimensions	public	TPrintPreview		C	X
SetFooterInformation	public	TPrintPreview		C	X
SetHeaderDimensions	public	TPrintPreview		C	X
SetHeaderInformation	public	TPrintPreview		C	X
SetMargins	public	TPrintPreview		C	X
SetPaperSize	public	TPrintPreview	X		
SetTab	public	TPrintPreview		C	X
SetTopOfPage	public	TPrintPreview		C	X
Start	public	TPrintPreview	X		
TestMargins	public	TPrintPreview		X	X
UpdateAction	public	TComponent			
UpdateBar	public	TPrintPreview		X	
WriteColumn	public	TPrintPreview		C	X
WriteDecimal	public	TPrintPreview		C	
WriteLine	public	TPrintPreview		C	X
WriteText	public	TPrintPreview		C	X

Abort

Called to abort the print job

```
procedure Abort;
```

Description The *Abort* method is called when you want to abandon the printing process. If *Abort* is called before calling *Finish*, then the *TPrintPreview* information is cleared and readied for the next print job. You must then use the *Start* method again to initialize the printer settings.

See also *Start* method, *Finish* method

Create

Create object and initialize values.

```
constructor Create(AOwner: TComponent); override;
```

Description The *Create* procedure is used internally when placing a *TPrintPreview* component on a form at design time. It can also be called in order to create a *TPrintPreview* component at run-time. The *Create* constructor initializes the component properties and internal settings.

CreateColumn

Defines a column.

```
procedure CreateColumn( Number:Word; XPosition:Single; Length:Single );
```

Description Conventional column usage would be to define columns that are successive across the page. However, this does not have to be the case. Up to 20 columns can be defined at any point across the page and can overlap. In this manner, text can be printed over several lines that stagger the column information.

Number specifies the column being defined. *XPosition* is the horizontal starting position of the column in inches and *Length* is the horizontal length of the column in inches.

Note Defining columns on a page can be very useful in various situations. Even if you aren't printing a columnar style report, columns can be handy for keeping information lined up or for developing a table-like format.

For example, columns 1 through 5 can define 5 fields of information for one line, the next line may contain 3 columns. Simply set columns 1-5 to the appropriate position and columns 6-8 to their respective positions. Using the print method *WriteColumn*, you can then print several lines of text in the form of a specialized table of information.

See also *WriteColumn* method

Destroy

Destroys objects created by *TPrintPreview*.

```
destructor Destroy; override;
```

Description Do not call *Destroy* directly in an application. Instead, call *Free*. *Free* verifies that the print object is not already freed, and then only calls *Destroy*.

See also *Free* method

DrawBox

Places a rectangular box around the given coordinates.

```
procedure DrawBox( XTop,YTop,XBottom,YBottom:Single; LineWidth:Single );
```

Description *XTop*, *YTop*, *XBottom* and *YBottom* are values that represent the upper left and lower right corners of the box to be drawn and are in inches. *LineWidth* is the drawing width of the line around the perimeter of the box. *LineWidth* is measured in points.

The *DrawBox* method is used to put a rectangular box around given coordinates. The box is not filled in or shaded. To produce a shaded rectangle, use the DrawBoxShaded method.

Note Boxes can be drawn around graphics, blocks of text, or anything on the printed page. You can also use them effectively for creating good-looking headers for pages like invoices containing information to be shown at the top of each page such as customer information or order information.

See also *DrawBoxShaded* method, *DrawTextBox* method, *SetFooterDimensions* method, *SetHeaderDimensions* method

DrawBoxShaded

Places a shaded rectangular box around the given coordinates.

```
procedure DrawBoxShaded( XTop,YTop,XBottom,YBottom:Single;
        LineWidth:Single; Shading:Word );
```

Description *XTop*, *YTop*, *XBottom* and *YBottom* are values that represent the upper left and lower right corners of the box to be drawn and are in inches. *LineWidth* is the drawing width of the line around the perimeter of the box. *LineWidth* is measured in points. *Shading* is the gray scale color from 0-black to 255-white.

The *DrawBoxShaded* method is used to put a shaded rectangular box around given coordinates. To produce a rectangle without fill, use the *DrawBox* method.

This method is similar to the *DrawBox* method except that a value is supplied to set the fill color of the canvas brush. By using the *Shading* value in all three color places {RGB}, a gray scale ranging from black(0) to white(255) can be achieved.

Note A good value for most printers is 235. It produces a very light gray that highlights the box without overemphasis. It may be desirable to allow the user to set this value to select a value that works well with the printer being used.

See also *DrawBox* method, *DrawTextBox* method, *SetFooterDimensions* method, *SetHeaderDimensions* method

Draw Line

Draws a line.

```
procedure DrawLine( TopX,TopY,BottomX,BottomY:Single; LineWidth:Single );
```

Description *XTop*, *YTop*, *XBottom* and *YBottom* are values that represent the beginning and end points of the line to be drawn and are in inches. *LineWidth* is the drawing width of the line in points.

The *DrawLine* method draws a line anywhere on the page and at any angle.

Note This is useful for separator bars between report sections, can be used for line drawing between multiple areas of the page to connect and show a relationship, etc.

DrawTextBox

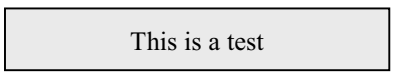
Draws a shaded box containing a line of text in a specified column.

```
procedure DrawTextBox( XTop,YTop,XBottom,YBottom:Single; LineWidth:Single;
    Shading:Word; Text:  string; Column: Integer; Alignment: Word );
```

Description *XTop*, *YTop*, *XBottom* and *YBottom* are values that represent the upper left and lower right corners of the box to be drawn and are in inches. *LineWidth* is the drawing width of the line around the perimeter of the box. *LineWidth* is measured in points. *Shading* is the gray scale color from 0-black to 255-white. *Text* is the line of text to be printed in the *Column* specified with the given *Alignment* within the column dimensions (0-1eft, 1-center, 2-right). The following constants are defined:

Constant	Value	Meaning
wtLeft	0	Left alignment in column
wtCenter	1	Center alignment in column
wtRight	2	Right alignment in column

Broadly, *DrawTextBox* combines the functionality of *DrawBoxShaded* and the *Write Column* command. The text is printed in the center vertically of the box rather than being separately specified. The text prints as a single line and if there is more text than will fit within the width of the box, it will be printed outside the box with a shaded background. The following commands print “This is a test.” in the center of 2 inch wide box as shown below.



```
CreateColumn(1,3.0,2.0);
DrawTextBox(GetColumnLeft(1),
            LastYPosition-0.5*LineHeightInches,
            GetColumnRight(1),
            LastYPosition+1.5*LineHeightInches,1,240,
            'This is a test.',1,1);
```

See also *DrawBoxShaded* method, *CreateColumn* method, *WriteColumn* method

Finish

Call *Finish* to complete the print job.

procedure Finish;

Description The *Finish* method must be called after all printing procedures have been completed in order to send the report to the printer or to show the preview screen if *Preview* is set to *True*. After the report is printed to the printer or the preview screen is closed, *Finish* releases all memory associated with the stored pages and images that were printed.

FixMargins

Call *FixMargins* to bring margins values within the printable area of the paper.

procedure FixMargins (**var** Top,Bottom,Left,Right:Single);

Description The *FixMargins* method is normally called when the function *TestMargins* detects that one or more values lie outside the printable area of the paper with the current printer. Any values that are too small are increased to the corresponding offset values.

See also *TestMargins* method, *SetMargins* method

GetColumnLeft

Returns the start of the column in inches.

function GetColumnLeft(Number:Word):

Description *Number* identifies the column number (1-20).

This function returns the leftmost coordinate of a given column. Used in conjunction with *GetColumnRight*, you can draw boxes around text that span the column information. You can also use the coordinates to position text so that everything lines up according to the column settings. The return value is in inches.

See also *CraeateColumn* method, *GetColumnRight* method

GetColumnRight

Returns the end of the column in inches.

function GetColumnRight(Number:Word):

Description *Number* identifies the column number (1-20).

This function returns the leftmost coordinate of a given column. Used in conjunction with *GetColumnLeft*, you can draw boxes around text that span the column information. You can also use the coordinates to position text so that everything lines up according to the column settings. The return value is in inches.

See also *CraeateColumn* method, *GetColumnLeft* method

GetMargins

Returns the margin settings.

procedure GetMargins(**var** Top,Bottom,Left,Right: Single);

Description The variables *Top*, *Bottom*, *Left* and *Right* are set to the values of the corresponding margins in inches.

Note In executing *SetMargins*, if an attempt is made to set margins that are smaller than the offsets for the current printer, the operator is given the option of the values to the value of the corresponding offset. *GetMargins* can be used to retrieve the values set.

See also *SetMargins* method, *TestMargins* method, *FixMargins* method, *OnMarginError* event

GetPaperParms

Returns information about the paper being used by the printer.

```
procedure GetPaperParms(var Top,Bottom,Left,Right,Height,Width: Single);
```

Description The variables *Top*, *Bottom*, *Left* and *Right* are set to the value of the corresponding offsets for the current printer. The variables *Height* and *Width* are set to the dimensions of the paper being used by the printer. All values are in inches.

See also *SetPaperSize* method

GetPixelsPerInch

Returns the resolution of the current canvas.

```
procedure GetPixelsPerInch( var X:Word; var Y:Word );
```

Description The variable *X* is set to the horizontal resolution of the current canvas and the variable *Y* is set to the vertical resolution.

GetPixelsPerPage

Returns size of page.

```
procedure GetPixelsPerPage( var X:Word; var Y:Word );
```

Description The variable *X* is set to the width of the page in pixels and the variable *Y* is set to the height.

GetTextWidthInches

Returns the width occupied by text.

```
function GetTextWidthInches( Text: string ): Single;
```

Description Returns the width in inches of *Text*.

See also *GetTextWidthPixels* method

GetTextWidthPixels

Returns the width occupied by text.

```
function GetTextWidthPixels( Text: string ): Integer;
```

Description Returns the width in pixels of *Text*.

See also *GetTextWidthInches* method

IncBar

Increments the progress bar.

```
procedure IncBar(X: Integer);
```

Description The progress bar position is incremented by the amount specified by *X*.

See also *Increment* method, *ProgressBarInc* property, *ProgressBarMax* property

InchesToPixelsX

Converts a measurement using the horizontal resolution

```
function InchesToPixelsX( Inches: Single ): Integer;
```

Description *InchesToPixelsX* converts the measurement *Inches* to pixels using the horizontal resolution of the current canvas.

See also *InchesToPixelsY* method, *PixelsToInchesX* method, *PixelsToInchesY* method

InchesToPixelsY

Converts a measurement using the vertical resolution

```
function InchesToPixelsY( Inches: Single ): Integer;
```

Description *InchesToPixelsY* converts the measurement *Inches* to pixels using the vertical resolution of the current canvas.

See also InchesToPixelsX method, *PixelsToInchesY* method, *PixelsToInchesX* method

Increment

Increments the progress bar.

```
procedure Increment;
```

Description The progress bar position is incremented by the amount specified by the *ProgressBarInc* property.

See also *IncBar* method, *ProgressBarInc* property, *ProgressBarMax* property

LinesToPixels

Returns the height of one or more lines.

```
function LinesToPixels( Line: Integer ): Integer;
```

Description *LinesToPixels* returns the height in vertical pixels of *Line* lines using the current font and canvas.

NewLines

Advances the current print position by one or more lines.

```
procedure NewLines( Number: Word );
```

Description Advances the current vertical print position by *Number* lines. The horizontal print position is set to the left margin. If *AutomaticPaging* is *True*, a test is made to determine whether any of the lines should be placed on a new page.

See also *NextLine* method, *GetLinesLeft* method, *AutomaticPaging* property

NewPage

Advances the report to a new page.

```
procedure NewPage;
```

Description The header, footer and page number information, if specified, is written to the current page. If a method for the *OnPageChange* event is specified, it is executed. The report is advanced to the next page and, if a method for the *OnNewPage* event is specified, it is executed. The print position is set at the left margin, one line above the detail area.

SeeAlso *OnPageChange* event, *OnNewPage* event, *SetFooterInformation* method, *SetHeaderInformation* method, *SetTopOfPage* method

NextLine

Advances the current print position by one line.

```
procedure NextLine;
```

Description Advances the current vertical print position by *one* lines. The horizontal print position is set to the left margin. If *AutomaticPaging* is *True*, a test is made to determine whether the line should be placed on a new page.

See also *NewLines* method, *GetLinesLeft* method, *AutomaticPaging* property

PixelsToInchesX

Converts a measurement using the horizontal resolution

```
function PixelsToInchsX( Pixels: Integer ): Single;
```

Description *PixelsToInchesX* converts the measurement *Pixels* to inches using the horizontal resolution of the current canvas.

See also *PixelsToInchesY* method, *InchesToPixelsX* method, *InchesToPixelsY* method

PixelsToInchesY

Converts a measurement using the vertical resolution

```
function PixelsToInchsY( Pixels: Integer ): Single;
```

Description *PixelsToInchesY* converts the measurement *Pixels* to inches using the vertical resolution of the current canvas.

See also *PixelsToInchesX* method, *InchesToPixelsY* method, *InchesToPixelsX* method

PrintGraphic

Prints a graphic on the current page of the report.

```
procedure PrintGraphic(X,Y,W,H: Single; thePicture: TImage);
```

Description *X* and *Y* specify the position, in inches, of the upper left-hand corner of *thePicture* on the printed page. *W* and *H* specify the width and height, in inches, of the printed image. (The *StrechDraw* method is used.)

RestoreCurrentFont

Restores the font previously saved.

```
procedure RestoreCurrentFont;
```

Description A stack, ten deep, is provided to save the currently defined font. *RestoreCurrentFont* uses the top element in the stack, if any, to set the font information and pops the stack.

See also *SaveCurrentFont* method, *SetFontInformation* method.

SaveCurrentFont

Saves the current font.

```
procedure SaveCurrentFont;
```

Description A stack, ten deep, is provided to save the currently defined font. *SaveCurrentFont* pushes the stack and stores the current font information.

See also *RestoreCurrentFont* method, *SetFontInformation* method.

SetDetailTopBottom

Specifies the area to be used for detail printing.

```
procedure SetDetailTopBottom( Top: Single; Bottom: Single );
```

Description The *Top* and *Bottom* of the detail area are specified in inches. A test is made to determine whether the detail area falls between the top and bottom margins. If not, the top and/or bottom margin is used instead of the values given.

When *AutomaticPaging* is *True*, an attempt to print below the bottom of the detail area forces a new page and sets *LastYPosition* (the current print position) to one line above the top of the detail area. The following sequence prints a previously opened text file *F* using a *TPrintPreview* component *Prn*:

```
Prn.AutomaticPaging := True;
Prn.SetDetailTopBottom(atop,aBottom);
while not EOF(F) do begin
  ReadLn(F,Temp);
  Prn.WriteLine(wtLeft,wtNextY,Temp);
  end;
{od while}
```

The *SetTopOfPage* method also sets *LastYPosition* to one line above the detail area.

See also *AutomaticPaging* property, *LastYPosition* property, *SetTopOfPage* method

SetFontInformation

Specifies a new font.

```
procedure SetFontInformation( Name: string; Size:Word; Style: TFontStyles );
```

Description Sets *Name* as the current font with *Size* and S*tyle*. Use *Size* to specify the point size of the font. If the value is negative, the internal leading that appears at the top of each line of text is included. If the value is positive, Size represents the height of the characters but not the internal leading. The following table shows the values for *Style*.

Value	Meaning
fsBold	The font is boldfaced.
fsItalic	The font is italicized.
fsUnderline	The font is underlined.
fsStrikeout	The font is displayed with a horizontal line through it.

See also *SaveCurrentFont* method, *SetFontStyle* method, *RestoreCurrentFont* method

SetFontStyle

Changes the style of the current font.

```
procedure SetFontStyle(Style: TFontStyles );
```

Description *Style* has the definition used in *SetFontInformation*.

See also *SaveCurrentFont* method, *SetFontInformation* method, *RestoreCurrentFont* method

SetFooterDimensions

Used to draw a box on each new page that, by convention, contains footer text.

```
procedure SetFooterDimensions( XTop,YTop,XBottom,YBottom:Single;
        Boxed: Boolean; LineWidth:Single; Shading:Word );
```

Description If *Boxed* is *True*, then a box with the upper left corner at (*XTop*,*YTop*) and lower right corner at (*XBottom*,*YBottom*) is drawn. Y*Top* and Y*Bottom* are distances above the bottom margin. If

Shading is greater than zero, then the box is shaded gray (1-black to 255-white) using the value given otherwise a box is drawn with the specified *LineWidth*. The coordinates are in inches and the *LineWidth* in points.

SeeAlso *SetFooterInformation* method, *DrawBox* method, *DrawBoxShaded* method, *SetHeaderDimensions* method

SetFooterInformation

Specifies text to be printed on each new page. By convention, this is footer text/

```
procedure SetFooterInformation( Line:Integer; Alignment:Word;
        YPosition: Single; Text: string; FontName: string; FontSize: Word;
        FontStyle: TFontStyles );
```

Description A TPrintPreview object has 10 footer lines available for use. *Line* selects one of these lines. *Alignment* and *Text* correspond to the similar parameters in *WriteLine*. *YPosition* is the distance the bottom of the footer line is above the bottom margin. *FontName*, *FontSize* and *FontStyle* correspond to the similar parameters in *SetFontInformation*.

By including the format specifier "%u" in *Text*, the page number is inserted when viewing or printing the document. If a second "%u" is included, then number of pages is inserted.

The following instruction prints "Page n of m Pages" at the bottom margin where n is a value of 1 to 10:

```
SetFooterInformation(n,wtRight,0.0,'Page %u of %u Pages',
                     'Times New Roman',10,[]);
```

If only a page number is desired, then only one format specifier is included. For example the Text value '- %u -' in the example above would print "- 1 -" at the bottom of the page.

Remarks Each footer line has it's own positioning so they may overlap if needed. For instance, one line could be the report type left justified and another line at the same *YPosition* could contain the date using right justified text. This would make them appear on the same line as if it were one print command.

See also *WriteLine* method, *SetFontInformation* method, *SetFooterDimensions* method, *SetHeaderInformation* method

SetHeaderDimensions

Used to draw a box on each new page that, by convention, contains header text.

```
procedure SetHeaderDimensions( XTop,YTop,XBottom,YBottom:Single; Boxed:
        Boolean; LineWidth:Single; Shading:Word );
```

Description When *Boxed* is *True*, a box with the upper left corner at (*XTop*,*YTop*) and lower right corner at (*XBottom*,*YBottom*) is drawn. *YTop* and *YBottom* are distances below the top margin. If *Shading* is greater than zero, then the boxes is shaded gray (1-black to 255-white) using the value given otherwise a box is drawn with the specified *LineWidth*. The coordinates are in inches and the *LineWidth* in points.

SeeAlso *SetHeaderInformation* method, *DrawBox* method, *DrawBoxShaded* method, *SetFooterDimensions* method

SetHeaderInformation

Specifies text to be printed on each new page. By convention, this is header text.

procedure SetHeaderInformation(Line:Integer; Alignment:Word;
YPosition: Single; Text: **string**; FontName: **string**; FontSize: Word;
FontStyle:TFontStyles);

Description A TPrintPreview object has 10 header lines available for use. *Line* selects one of these lines. *Alignment* and *Text* correspond to the similar parameters in *WriteLine*. *YPosition* is the distance the bottom of the footer line is below the top margin. *FontName*, *FontSize* and *FontStyle* correspond to the similar parameters in *SetFontInformation*.

By including the format specifier "%u" in Text, the page number is inserted when viewing or printing the document.

The following instruction prints "Page n of m Pages" at the top margin where n is a value of 1 to 10:

```
SetHeaderInformation(n,wtRight,0.0,'Page %u of %u Pages',
                     'Times New Roman',10,[]);
```

If only a page number is desired, then only one format specifier is included. For example the Text value '- %u -' in the example above would print "- 1 -" at the top of the page.

Remarks Each header line has it's own positioning so they may overlap if needed. For instance, one line could be the title centered and another line at the same *YPosition* could contain a reference "tab" using right justified text. This would make them appear on the same line as if it were one print command.

See also *WriteLine* method, *SetFontInformation* method, *SetHeaderDimensions* method, *SetfooterInformation* method

SetMargins

Sets properties to define print area.

procedure SetMargins(Top,Bottom,Left,Right:Single);

Description The parameters *Top*, *Bottom*, *Left* and *Right* specify the distance in inches from the corresponding edge of the paper.

A test is made to determine whether the defined print area falls inside the page (printable area) for the current printer. If it does not and an *OnMarginError* event is defined, then control is passed to that procedure that can take corrective action. If the event is not defined, then the operator is given the choice of letting the program increase the margin in error to the corresponding *Offset* value, ignoring the error or canceling printing.

See also *FixMargins* method, *GetMargins* method, *TestMargins* method, *OnMarginError* event

SetPaperSize

Specfies the size of the paper to be used.

function SetPaperSize(aSize: Integer; aHeight, aWidth: Single): Boolean;

Description If *aSize* is zero, then *aHeight* and *aWidth* specify the size of the paper to be used. *aSize* can also be assigned one of the paper selection constants (starting with "`DMPAPER_`") listed in `windows.pas`. If the size is set, *True* is returned; otherwise, *False* is returned.

The published property *PaperSize* must be set to "PtrSU" and *SetPaperSize* must be called before calling the method *Start*.

Remarks The published property *PaperSize* can be used to select "Letter" (8½x11) or "Legal" (8½x14). When *PaperSize* is set to "PtrSU", a *TPrinterSetupDialog* object can be used to select both the printer and paper size to be used.

Setting custom paper sizes with *SetPaperSize* must be done with an understanding of restrictions imposed by the printer drivers to be used. This method was included to support label printing on dot matrix printers.

See also *PaperSize* property, `windows.pas` source code or *DEVMODE* topic in the Microsoft win32 help file for *aSize* values.

SetTab

Specifies the indentation of a line

```
procedure SetTab(Inches: Single);
```

Description The parameter *Inches* specifies the indention to be used with *WriteLine* when the constant *wtIndent* is used to specify the alignment.

See also *WriteLine* method

SetTopOfPage

Positions the print position for detail printing.

```
procedure SetTopOfPage;
```

Description SetTopOfPage sets *LastYPosition* to one line above the detail area.

See also *SetDetailTopBottom* method, *NewPage* method

TestMargins

Tests whether the print area falls within the page defined to the printer.

```
function TestMargins( Top,Bottom,Left,Right: Single ): Boolean;
```

Description The parameters *Top*, *Bottom*, *Left* and *Right* specify the distance in inches from the corresponding edge of the paper.

A test is made to determine whether the defined print area falls inside the page for the current printer. If they do, *True* is returned; otherwise, *False* is returned.

Start

Initializes a *TPrintPreview* object to print a report.

```
procedure Start;
```

Description *Start* must be executed before executing any printing commands. The printing of headers, footers and page numbers is reset. The following *TPrintPreview* properties are initialized.

Property	Visibility	Default	Initialized by Start
AutomaticPaging	published	False	
Detail Top Bottom	private		printer offsets
FontName	public, read_only		**Note:** The application should set initial font before issuing printer commands.
FontSize	public, read_only		
FontStyle	public, read_only		
Footer definition	private		cleared
Header definition	private		cleared
Margins	private		printer offsets
Orientation	published	poPortrait	
Page number text	private		null string

Property	Visibility	Default	Initialized by Start
PaperHeight	public, read_only		
PaperSize	published	PtrSU	uses “Letter” or “Legal”
PaperWidth	public, read_only		
Preview	published	False	True
Print position	private		top left corner of paper
ProgressBarInc	published	1	
ProgressBarMax	published	100	
Running	public		True
ShowCancel	published	True	
ShowPrintDialog	published	True	
ShowProgress	published	True	
StartPercent	published	P2_Fifty	
Title	published		
Transparent	published	True	

SeeAlso *Finish* method, *Abort* method

UpdateBar

Sets progress bar.

```
procedure UpdateBar(X: Integer);
```

Description Sets the progress bar to the percentage of *ProgressBarMax* represented by *X*.

See also *Increment* method, *ProgressBarMax* property.

WriteColumn

Prints text in a column.

```
procedure WriteColumn(ColumnNumber:Word; Alignment: Word; Y:Single;
        Text: string);
```

Description The *Text* is printed in column *ColumnNumber* with the specified *Alignment* at the vertical position on the paper specified in inches by *Y*.

Alignment is a value 0, 1 or 2. The following table shows the meaning assigned to these values and constants that have been defined for the values.

Constant	Value	Meaning
wtLeft	0	Left alignment in column
wtCenter	1	Center alignment in column
wtRight	2	Right alignment in column

The values –1.0 and –2.0 have special meaning when assigned to *Y*. The following table shows the meanings assigned to these values and the constants that have been defined.

Constant	Value	Meaning
wtNextY	-1.0	Use next line
wtThisY	-2.0	Use this line

See also WriteDecimal method, *WriteLine* method, *WriteText* method, *CreateColumn* method

WriteDecimal

Prints text with decimal alignment.

```
procedure WriteDecimal( X:Single; Y:Single; Text: string );
```

Description The *Text* is normally the text representation of a number including a decimal point. It is printed with the decimal point (period) in the horizontal position on the paper specified by *X* at the vertical position on the paper specified in inches by *Y*. If *Text* does not contain a "period" then it is printed right aligned at *X*.

The values –1.0 and –2.0 have special meaning when assigned to *Y*. The following table shows the meanings assigned to these values and the constants that have been defined.

Constant	Value	Meaning
wtNextY	-1.0	Use next line
wtThisY	-2.0	Use this line

See Also *WriteColumn* method, WriteLine method, WriteText method

WriteLine

Prints a line of text.

```
procedure WriteLine( Alignment: Word; Y:Single; Text: string );
```

Description The *Text* is printed with the specified *Alignment* at the vertical position on the paper specified in inches by *Y*.

Alignment is a value 0, 1, 2 or 3. The following table shows the meaning assigned to these values and constants that have been defined for the values.

Constant	Value	Meaning
wtLeft	0	Left alignment in column
wtCenter	1	Center alignment in column
wtRight	2	Right alignment in column
wtIndent	3	Indent by current tab amount

The values –1.0 and –2.0 have special meaning when assigned to *Y*. The following table shows the meanings assigned to these values and the constants that have been defined.

Constant	Value	Meaning
wtNextY	-1.0	Use next line
wtThisY	-2.0	Use this line

See Also *WriteColumn* method, WriteDecimal method, WriteText method

WriteText

Prints a text string at an arbitrary position.

```
procedure WriteText(X:Single; Y:Single; Text: string; Style:TFontStyles );
```

Description The *Text* is printed at the horizontal position on the paper specified in inches by X and the vertical position on the paper specified in inches by *Y* with the font style *Style*. See Delphi help for *TFontStyle*.

The values –1.0 and –2.0 have special meaning when assigned to *X* or *Y*. The following table shows the meanings assigned to these values and the constants that have been defined.

Constant	Value	Meaning
wtNextX	-1.0	Use current horizontal position
wtLeftX	-2.0	Align to left margin
wtNextY	-1.0	Use next line
wtThisY	-2.0	Use this line

See Also *WriteColumn* method, WriteDecimal method, *WriteLine* method, *TFontStyle* type