VPE+ Application Reporting Interface

Version 1.0.3

User Guide & Notes

Revision: April 2016

www.onlycode.nz

VPE+ Application Reporting Interface

Contents

VPE+ Application Reporting Interface

Contents

VPE+ Application Reporting Interface

Introduction

VPE Plus (VPE+) is a free Delphi VCL code-based reporting tool. It is built as an add-on to Virtual Print Engine (VPE) which is independently licensed.

The Code-Based Approach

As a code-based tool, reports generated with VPE+ are implemented entirely in standard Delphi code, so:

> you are not tied to the constraints of automation often encountered in a visual designer
> there are no constraints on access to data sources, no need to "pipe" or "link" data into the report environment
> there are no constraints on access to code resources
> there are no constraints on how Delphi language features can be applied during report generation

This hands-on flexibility means you can pretty much code whatever is required to get the job done!

VPE+ Reporting Style

Generally, VPE+ adopts a banded style of reporting, although you are by no means limited to using bands at all. Fixed and dynamic bands (page headers and footers, group headers and footers, report body bands etc etc) are implemented within "frame" components which provide a flexible structure and flow to the report generation process.

Conceptually, a positional (x, y) cursor moves in synchrony with the various frames, bands and output functions so that sequential text elements can be automatically and easily placed without explicit reference to positional coordinates. By the same token, x and/or y coordinates can be specified with print functions if necessary, irrespective of the current cursor position. In addition, the cursor can be readily manipulated to precisely and easily align text with respect to another object or the font metric of another text element anywhere on the page. Positionally, VPE+ is very precise.

Text is output either as single line (potentially clipped) strings, or as multi-line wrapping blocks. Horizontal tabs can be defined for single line columnar style reports with optional surrounding boxes and background shading. For wrapping text blocks, text can be pre-rendered (without actual output) to assess space requirements, and then incrementally output around other objects or wrapped across multiple pages. Rich text (RTF) can be wrapped in a similar fashion.

Application Interface Features

More than just code-based report generation, however, VPE+ also provides customisable application integration features to manage the user interface with support for:

> easy presentation of report setup parameters and options
> previewing, printing, filing, emailing of reports
> batching of multiple reports
> bulk report runs
> overridable setup, status and preview forms

and much more...

What is Virtual Print Engine (VPE)?

Virtual Print Engine is the underlying core of VPE+. It is a comprehensive cross platform Report Engine and PDF Library licensed and supported by Ideal Software. To use VPE+ as a developer, you require a license for VPE Professional Edition, or you can use the free 30 day trial version of this edition.

VPE+ is simply a wrapper for VPE, implementing it's own style of code-based, banded reporting with the Delphi VCL.

Note, in particular, that there are many more properties, methods and features in VPE beyond those directly wrapped or introduced by VPE+. It is thus essential to become familiar with VPE to take full advantage of it when using VPE+.

Refer to the Ideal Software site (www.idealsoftware.com) for detailed documentation on VPE and its sister product dycodoc.

IMPORTANT TO REMEMBER:

Take particular care when mixing methods from VPE and VPE+ involving measurement units. The former uses centimetres as a default, the latter uses millimetres as a default. VPE does not support millimetre units as such, so unless you synchronise the respective units to a common value, you may need to use the unit conversion functions (AsVPEUnits, AsReportUnits, ConvertUnits) to integrate the two.

VPE+ Application Reporting Interface

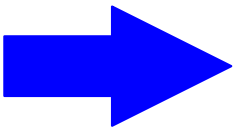
Introduction

SUPPORT

The VPE+ add-on is neither developed nor directly supported by Ideal Software. Please do NOT ask VPE+ support questions of Ideal Software, or use their forums for this purpose!

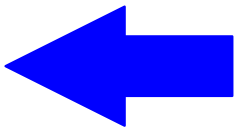
VPE+ has been developed by Brent Rose. You can contact me via the "Contact" page at www.onlycode.nz.

While I will endeavour to answer any questions, please bear in mind that this is free software and I cannot guarantee the time to respond immediately or to everyone at this stage.



If you do use VPE+, or are considering licensing a copy of VPE in order to use VPE+, please let me know (use the "Contact" page at www.onlycode.nz to introduce yourself).

How much interest there is in VPE/VPE+ in this context is likely to determine the future development course of VPE+, and may also impact on VPE. Your feedback is welcome, and your contact details will NOT be used to send you unsolicited mail, or for any other purpose.



YOUR IMPROVEMENTS

If you discover any issues with VPE+, resolve any bugs, make your own improvements, or have some constructive suggestions, please notify me (via the "Contact" page at www.onlycode.nz) so that the project can benefit from your efforts. Thanks for trying VPE and VPE+.

THIS USER GUIDE:

This document is intended as a "brief reference guide" only. It is generated in code using VPE+ and grew as an exercise in testing its' functionality, but is otherwise probably a somewhat clumsy way of going about compiling a manual! Nevertheless, there is a lot of useful information here and browsing through it is a good way to gain familiarity with the scope and features of VPE+.

Another useful resource is the VPE+ demonstration application (VPEPlusDemo.exe) and its' source code. There are examples of many of the key features of VPE+ included here. Obtain these resources and others from the website www.onlycode.nz.

VPE+ Application Reporting Interface

Introduction

Installing VPE+:

Prerequisite: Install VPE Professional from Ideal Software. A 30 day trial is available.
Be sure the "<VPE Source Folder>\delphi" folder is included in the IDE Library Path.

To keep installation of VPE+ as versatile as possible, and allow you to readily make modifications to the code, only the source code files are provided. You must build and install a component package in your IDE... but this is fairly straight forward. At this time, the installation has not been tried with Delphi versions prior to XE2.

1. Extract the VPE+ source files to a component folder of your choice. eg \VPEPlus in your component installation folder.

   The following (26) files should be present:

CONTACT.txt	some contact details
GNU General Public License.html	copy of freeware license file in html format
GNU General Public License.txt	copy of freeware license file in txt format
HISTORY_VPEPLUS.txt	notes on version and change history
VPEPlusRegister.pas	component registration unit
VPEPlusRegister.res	resource file for VPE+ components
VPEfPlusBatchOutput.dfm	form file for batched report selection and output form
VPEfPlusBatchOutput.pas	code file for batched report selection and output form
VPEfPlusBatchSetup.dfm	form file for report batch print device selection form
VPEfPlusBatchSetup.pas	code file for report batch print device selection form
VPEfPlusPageSelection.dfm	form file for page range selection form
VPEfPlusPageSelection.pas	code file for page range selection form
VPEfPlusPaperBinPrompt.dfm	form file for paper bin selection form
VPEfPlusPaperBinPrompt.pas	code file for paper bin selection form
VPEfPlusPreview.dfm	form file for default report preview form
VPEfPlusPreview.pas	code file for default report preview form
VPEfPlusSetup.dfm	form file for default report setup form
VPEfPlusSetup.pas	code form file for default report setup form
VPEfPlusStatus.dfm	form file for default report status form
VPEfPlusStatus.pas	code file for default report status form
VPEuPlusColumnFrame.pas	code file to output to "newspaper columns"
VPEuPlusDevice.pas	code file to manage report print devices & options
VPEuPlusFrame.pas	code file for banded report frame components
VPEuPlusLabelFrame.pas	code file for the label printing frame
VPEuPlusReporting.pas	code file for core ReportInterface component and functionality
VPEuPlusTextBlock.pas	code file implementing the "text block" for plain text and rtf

2. Add the VPE+ component folder to your IDE Library Path.

3. From the IDE menu, select "Component, Install Component".

   For "Units" select all (14) .pas files from the VPE+ base folder.

   Check the radio button to "Install into a new package".

   Click <Next>.

4. Define the new package.

   Use Package Name = "VPEPlus", selecting a folder of your choice

   Use Description = "VPE Plus Interface"

   Click <Finish>.

5. Include additional files as prompted:

   When asked to enable the "Visual Component Library" framework, click <Yes>.

   When asked to add "vclimg" and "dbrtl", click <OK>.

   There may be some variation with installs in different IDEs.

   If asked to add VpevclXe2 (or later version) as well, click <OK>.

   To remove warnings that VPE_VCL and VPEngine have been "implicitly imported", find and add these (.dcu) files.
   (Their location may vary from installation to installation.)

6. Compile error with "Debug Information" setting:

   You may get an error with this setting when compiling the demo application under earlier IDEs where the "Debug information" setting is a boolean value rather than the more recent enumerated type. See "Project Options, Delphi Compiler, Compiling" in the Debugging group. Remove the numeric representation and reset the boolean value (as True). The compile error will look something like:

   [MSBuild Error] "2" is an invalid value for the "DebugInformation" parameter of the "DCC" task. The "DebugInformation" parameter is of type "System.Boolean".

VPE+ Application Reporting Interface

Introduction

SIXTY SECONDS TO YOUR FIRST REPORT: (Well, maybe 3 minutes or so.)

Creating the following VCL project demonstrates how to quickly generate your first VPE+ report with a standard user interface setup form.

1. On the main form of a new VCL project place:
    a) a TReportInterface component (default name ReportInterface1)
    b) a TReportWriter component (default name ReportWriter1)
    c) a TButton component (default name Button1)
2. Execute the ReportWriter via the ReportInterface by adding to Button1.OnClick:

```
ReportInterface1.ExecuteReport(ReportWriter1);
```

3. In ReportWriter1.OnConfigure, give the report a setup title & subtitle, disable email output, and turn off PageTitles (not used for now):

```
ReportWriter.ReportTitle := 'A Sample Report';
ReportWriter.ReportSubTitle := 'Testing';
ReportWriter.UsePageTitles := False;
ReportWriter.OptionDisallow(roCanEmail);
```

4. Add an output statement to ReportWriter1.OnGenerate:

```
ReportInterface.PrintPos('Hello World');
```

5. Run the application and click the button. Click the <Preview> button to preview the report.

As you can see from this simple application, you can also (in addition to previewing the report) select a print device and send the report to it, or save the report to a file. By default, the only file output format available is PDF. We turned off the email output option because we have not implemented the means to send a report by email in this case (per TReportWriter.OnSendEmail).

Note also that the report is overstamped with "Demo Version". This will be automatically printed until valid VPE license keys are entered in the interface (see properties VPEInterface1.VPELicenseKey1 and VPELicenseKey2)

EXTENDING YOUR FIRST REPORT TO USE A BANDED PAGE FRAME:

From here, we can embellish the report a little by using a page frame to add a banded structure. We'll utilise the "system header/footer" page title features which allow you to (optionally) implement a generic header and footer style for your reports.

1. On the main form, add:
    a TPageFrame component (default name PageFrame1)
2. Alter the configuration in ReportWriter1.OnConfigure, turning page titles ON:

```
ReportWriter.UsePageTitles := True;
```

    (Or delete this statement since "True" is the default state.)
3. Add default page title values in ReportWriter1.OnConfigure (these will show up in the setup form):

```
ReportWriter.PageHeaderTitle := 'My First Report';
ReportWriter.PageHeaderSubTitle := 'Quick Test';
ReportWriter.PageFooterTitle := 'Footer';
```

4. Rather than print "Hello World" in ReportWriter1.OnGenerate (delete this statement), we'll execute the PageFrame1 instead:

```
PageFrame1.Execute(ReportWriter);
```

5. Print "Hello World" in PageFrame1.OnRow, and end the frame loop by invalidating it:

```
ReportInterface.PrintPos('Hello World');
Valid := False;
```

6. Execute the system page header in PageFrame1.OnPageHeader:

```
ReportInterface.PrintSystemPageHeader;
```

7. Execute the system page footer in PageFrame1.OnPageFooter:

```
ReportInterface.PrintSystemPageFooter;
```

8. Implement the "system page header" output in ReportInterface1.OnSystemPageHeader (we'll just centre the title and subtitle):

```
with ReportInterface, ReportWriter do
begin
    PrintPos(PageHeaderTitle, jCentre, BandCentreXPos, tsB);
    NewLine;
    PrintPos(PageHeaderSubTitle, jCentre, BandCentreXPos, tsB);
end
```

Introduction

9. Implement the "system page footer" output in ReportInterface1.OnSystemPageFooter (we'll centre the title and add a page number):

```
with ReportInterface, ReportWriter do
begin
    PrintPos(PageFooterTitle, jCentre, BandCentreXPos);
    PrintPos(Format('Page %d', [CurrentPage]), jRight, BandRight);
end
```

10. Run the application and click the button. Click the <Preview> button to preview the report.

WHERE TO FROM HERE?

There are, of course, many more features of VPE+ beyond this brief introduction!

As mentioned, check out the demonstration application which goes much further in showing the extent to which the default VPE+ setup form can be customised, including presenting almost any set of user input parameters. You can even generate your own setup form to better suit your purposes.

There is a "Categorical Index" of events, methods and properties at the end of this guide which serves as a brief indication and reminder of available features. When manipulating fonts, the "Font Metrics" diagramme is useful (see the "Line & Font Metrics" section). When using Page, Master, Detail, Label or Column Frames, the respective structure diagrammes in the "Frames" section are useful.

Information and code examples will continue to be added to the website, so check the site from time to time.

Remember, also, to refer to the VPE documentation from Ideal Software describing the full scope of the Report Engine from document management to printing, layout, drawing and text functions, to bar codes, charts, and clickable objects etc...

VPE+ Application Reporting Interface

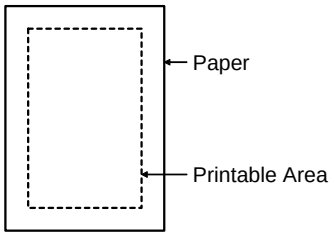
Overview

General Overview

The ReportInterface component provides reporting infrastructure and interface between an application and the Virtual Print Engine (VPE). Typically, a single ReportInterface component is used in an application, placed in a common DataModule.

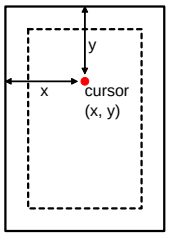
A ReportWriter component (a descendant of the VPE TVPEngine) is placed on a form as the basis of a given report (or reports).

ReportInterface describes a "page" (or piece of paper) as the target of report output, within which is a printable area:



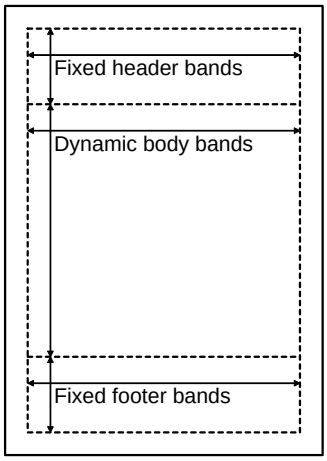
Paper Boundaries & Printable Area

The printable area constitutes a "default band" in which you can output report elements (but does not strictly confine you to the printable area). A virtual (XPos, YPos) cursor marks the default output position within the band, and moves according to report output or programmatic control. However, the cursor is only a convenience, and in no way restricts output to the page using ReportInterface or VPE output methods.



Output Cursor Position

Optionally, ReportInterface can then logically structure this default band using a "framework" which imposes more detailed bands on the page space and a cyclic control flow to progress through a report. The primary framework component is a PageFrame which introduces fixed header and footer bands, and dynamic report body bands in between.



Band Structure

In addition, further frames in the form of MasterFrame or DetailFrame components can be added and optionally nested to elaborate on the band structure. A specialised frame, LabelFrame, generates mailing or other labels. Another specialised frame, ColumnFrame, facilitates output to newspaper style columns.

Frames and bands and the properties that define and control their behaviour are described later in this guide.

More than just reporting structure, however, ReportInterface provides a great deal of versatile application functionality to support report setup, preview, and output, including the ability to dynamically present custom report parameters to the user, batch reports, and manage bulk report runs. Should requirements challenge this default interface, all key forms (setup, status, preview) can be overridden with your own custom forms. Any number of such custom forms may be utilised in an application.

VPE+ Application Reporting Interface

VPE+ Report Interface (TReportInterface) General

ReportInterface is a component providing reporting infrastructure and interface between an application and the Virtual Print Engine (VPE). Typically, a single ReportInterface component is used in the application, placed in a common DataModule. A ReportWriter component (a descendant of the VPE TVPEngine) is placed on a form as the basis of a given report or reports.

Default Interface Setup & Properties

property ActiveReportFolder: string;
Default folder to receive user report files.
By order of preference, this is taken from property DefaultReportFolder (see below).
If DefaultReportFolder is invalid or not specified, the folder is set in event OnGetReportFolder.
If no OnGetReportFolder event handler is assigned, the EXE folder is used.
NOTE: The ActiveReportFolder is calculated ONCE, and that value is retained.
Force a re-calculation by assigning any value (including blank) to DefaultReportFolder.

property ActiveTempFolder: string;
Default folder to receive temporary report files.
By order of preference, this is taken from property DefaultTempFolder (see below).
If DefaultTempFolder is invalid or not specified, the folder is set in event OnGetTempFolder.
If no OnGetTempFolder event handler is assigned, the TEMP environment variable is used.
NOTE: The ActiveTempFolder is calculated ONCE, and that value is retained.
Force a re-calculation by assigning any value (including blank) to DefaultTempFolder.

property DefaultFixedBandEnabled
Default fixed band enable states to be applied on report execution.
(The ReportWriter.FixedBandEnabled state, if not bsDefault, overrides these defaults.)

- LetterfootEnabled: TDefaultBandState; — default for UseLetterfoot state (default bsDisabled)
- LetterheadEnabled: TDefaultBandState; — default for UseLetterhead state (default bsDisabled)
- PageFooterEnabled: TDefaultBandState; — default for UsePageFooter state (default bsEnabled)
- PageHeaderEnabled: TDefaultBandState; — default for UsePageHeader state (default bsEnabled)

NOTE: Remittance band is disabled by default - enable it using EnableRemittance.
NOTE: FixedBand "Use" states can be overridden in ReportWriter.OnConfigure.

TDefaultBandState
band states (subset of type TBandState)

- bsDisabled — band is disabled
- bsEnabled — band is enabled

property DefaultFixedBandHeights
Default fixed band heights to be applied on report execution.

- LetterfootHeight: Double; — default for UseLetterfootHeight (default 10 mm)
- LetterheadHeight: Double; — default for UseLetterheadHeight (default 20 mm)
- PageFooterHeight: Double; — default for UsePageFooterHeight (default 10 mm)
- PageHeaderHeight: Double; — default for UsePageHeaderHeight (default 25 mm)
- RemittanceHeight: Double; — default for UseRemittanceHeight (default 20 mm)

NOTE: Override default band heights with the respective ReportWriter Band properties.

property DefaultFonts
Defines a default font in 5 sizes for general use (optionally applied as saved fonts 1..5).
(overridden by ReportWriter.DefaultFonts where these are defined)

- FontName: string; — name of font to use (default = Arial)
- Size1 to Size5: Integer; — font sizes to use: defaults = Size1 (16), Size2 (14), Size3 (12), Size4 (10), Size5 (8)

call ***procedure SetDefaultFonts;*** to save as fonts 1..5 (eg in ReportWriter.OnConfigure)

property DefaultPageFooterStamp: string;
optional "stamp" for use in the page footer - eg a company name (default none)
NB this global default is applied when TReportWriter.PageFooterStamp is not defined.

property DefaultPaperMargins
Defines the default paper margins (override by calling SetPaperMargins)

- Bottom: Double; — sets bottom paper margin (default 10 mm)
- Left: Double; — sets left paper margin (default 15 mm)
- Right: Double; — sets right paper margin (default 15 mm)
- Top: Double; — sets Top paper margin (default 10 mm)

property DefaultReportDescription: string;
Default report description (default = "Report") applied where ReportWriter.ReportDescription is not provided. See ReportWriter.OnDescribeReport event for dynamic description changes.
The report description is used in setup and as the print spool job description.

property DefaultReportFolder: string;
Default folder to receive user report files. See property ActiveReportFolder above.
Leave DefaultReportFolder blank to defer to the event OnGetReportFolder.
If no OnGetReportFolder event handler is assigned, the EXE path is used.

property DefaultTempFolder: string;
Default folder to receive temporary report files. See property ActiveTempFolder above.
Leave DefaultTempFolder blank to defer to the event OnGetTempFolder.
If no OnGetTempFolder event handler is assigned, the TEMP environment variable is used.

property DefaultTitleSetup: string;
default setup form caption (default = "Report Setup", overridden by ReportWriter.TitleSetup)

property DefaultTitleStatus: string;
default status form caption (default = "Report Status", overridden by ReportWriter.TitleStatus)

property TagPreview: Integer;
optional tag passed to a custom (override) preview form (eg for formatting or control options)

property TagSetup: Integer;
optional tag passed to a custom (override) setup form (eg for formatting or control options)

property TagStatus: Integer;
optional tag passed to a custom (override) status form (eg for formatting or control options)

property TitleSystem: string;
system title (default = "Report System") - used for prompt titles and default report title

VPE+ Application Reporting Interface

VPE+ Report Interface (TReportInterface) General

property Units: TUnits;	VPE+ report system units (uMM or uCM or uInch). Default is uMM. NB This setting may differ from the units set for VPE (uCM or uInch). *See the "Unit Conversion" topic below.*
property UseEmbeddedFlagParser: Boolean;	if True, a leading "[..]" in string output contains VPE format codes (default False)
property VPELicenseKey1: string;	VPE License Key #1 as provided by Ideal Software
property VPELicenseKey2: string;	VPE License Key #2 as provided by Ideal Software
property VPEUnits: TVPEUnits;	Units used by the underlying VPE system (uCM or UInch). Default is uCM. NB This setting may differ from the units set for VPE+ (uMM or uCM or uInch). *See the "Unit Conversion" topic below.*

System Page (Fixed) Band Output

Four system print procedures can be used to execute generic fixed band output for letterheads, page headers, page footers and letterfoots. These methods can be called from any report. An optional OptionTag allows instance specific formatting information to be passed to modify the output accordingly.

The print procedures call matching Interface event handlers in which the output is implemented (see below).

procedure PrintSystemLetterhead(OptionTag: Integer);	fires OnSystemLetterhead to print a system letterhead an optional tag to pass formatting options
procedure PrintSystemPageHeader(OptionTag: Integer);	> fires OnSystemPageHeader to print a system page header an optional tag to pass formatting information
procedure PrintSystemPageFooter(OptionTag: Integer);	> fires OnSystemPageFooter to print a system page footer an optional tag to pass formatting information
procedure PrintSystemLetterfoot(OptionTag: Integer);	> fires OnSystemLetterfoot to print a system letterfoot an optional tag to pass formatting information
procedure OnSystemLetterhead(ReportInterface: TReportInterface; ReportWriter: TReportWriter; OptionTag: Integer);	implements output for the generic system letterhead ReportInterface component managing the reporting process ReportWriter generating the report an optional tag to pass formatting information
procedure OnSystemPageHeader(ReportInterface: TReportInterface; ReportWriter: TReportWriter; OptionTag: Integer);	implements output for the generic system page header ReportInterface component managing the reporting process ReportWriter generating the report an optional tag to pass formatting information
procedure OnSystemPageFooter(ReportInterface: TReportInterface; ReportWriter: TReportWriter; OptionTag: Integer);	implements output for the generic system page footer ReportInterface component managing the reporting process ReportWriter generating the report an optional tag to pass formatting information
procedure OnSystemLetterfoot(ReportInterface: TReportInterface; ReportWriter: TReportWriter; OptionTag: Integer);	implements output for the generic system letterfoot ReportInterface component managing the reporting process ReportWriter generating the report an optional tag to pass formatting information

System Setup Form

The default system setup form can be customised to handle a wide range of report options (or overridden altogether with another custom form - see ReportWriter.OverrideSetup event).

Various report titles, headers and footers can be automatically displayed in the default setup form and edited by the user. Alternatively, your own header/footer controls can be placed in a TGroupBox and passed to the form by assigning the box to property ReportWriter.ReportTitleGroupBox.

Controls for any report parameters can similarly be placed in a TGroupBox and passed to the setup form by assigning the box to property ReportWriter.ReportOptionGroupBox.

In both cases, the respective TGroupBox can be hidden (set Visible := False) on the local TForm in which the report code is located. The default setup form will temporarily parent and show the TGroupBox allowing the user to manipulate all parameters.

See ReportWriter notes for further details.

property OverrideSetup(ReportInterface: TReportInterface; ReportWriter: TReportWriter; OverrideState: TOverrideFormState; var OverrideForm: TForm; OptionTag: Integer);	Allows an alternative setup form to be used during the report process. ReportInterface component managing the reporting process ReportWriter generating the report indicates form state to implement: ofsFree, ofsCreate, ofsShow, ofsHide the override form instance as created and returned when OverrideState = ofsCreate. an optional tag to be used for setup options

VPE+ Report Interface (TReportInterface) General

System Preview Form

The default system preview form implements a full range of VPE preview functionality. It is also capable of handling report batches (loading, previewing, and outputting multiple reports collectively as a batch).

Standard VPE preview settings can be adjusted to control the appearance of the preview window. Set the ReportWriter.PreviewWindow properties to control the basic layout of the preview window. The ReportWriter.OnPreviewConfigure event then allows further configuration as required.

See ReportWriter notes for further details.

The preview form can be overridden altogether with another custom form using the ReportInterface.OverridePreview event:

property OverridePreview(	Allows an alternative preview form to be used during the report process.
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
ReportWriter: TReportWriter;	ReportWriter generating the report
OverrideState: TOverrideFormState;	indicates form state to implement: ofsFree, ofsCreate, ofsShow, ofsHide
var OverrideForm: TForm;	the override form instance as created and returned when OverrideState = ofsCreate.
OptionTag: Integer);	an optional tag to be used for preview options

System Status Form

The default system status form is a simple display panel allowing progress and status messages to be displayed during report generation. Use the procedures below to utilise the form.

The status form can be overridden altogether with another custom form using the ReportInterface.OverrideStatus event:

property OverrideStatus(	Allows an alternative status form to be used during the report process.
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
ReportWriter: TReportWriter;	ReportWriter generating the report
OverrideState: TOverrideFormState;	indicates form state to implement: ofsFree, ofsCreate, ofsShow, ofsHide
var OverrideForm: TForm;	the override form instance as created and returned when OverrideState = ofsCreate.
OptionTag: Integer);	an optional tag to be used for status options
property StatusLabel: TLabel;	target for ReportInterface status messages (per procedure DisplayStatus) assigned in default status forms constructor, or can be assigned in an override status forms constructor, or can be assigned in ReportWriter.OnExecuted
procedure DisplayStatus(	displays StatusString in the status form
StatusString: string);	eg call in OnGenerateStart for a single report-wide status message.
procedure HideStatusForm;	hides the status form
procedure ShowStatusForm;	shows the status form

Paper Layout/Orientation

Paper layout can be changed by setting ReportWriter.PaperLayout, which reads or writes to VPE.PageOrientation.

Layout can be set in advance in PageFrame.OnReportBefore, or changed in ReportWriter.OnPageStart (or elsewhere).

VPE+ Report Interface (TReportInterface) General

Page Numbering

If using the "Page x of y" style of page numbering where the total number of pages must be known, or inserting/deleting pages during report generation, page numbering may need to be applied AFTER the document is otherwise complete. By allowing you to maintain a list of page number positions (XPos, YPos and justification) for each page generated, ReportInterface can retrospectively apply page numbering (eg in TPageFrame.OnReportAfter).

Using the procedures listed below, it is your responsibility to synchronise the list with any changes made to the page structure. You can also manipulate the final list before applying page numbers (eg to force alternate left/right justification).

NB Set ReportWriter.CurrentPage to go to a page, or read it to get the current page.

The following procedures are available to manage page numbering:

procedure AddPageNoPos(	adds a page number position to the page number list
X, Y: Double;	the (X, Y) cursor position of the page number
AJustify: TJustify = jRight);	justification for page number output
procedure AddPageNoPosVoid;	adds a page number flag indicating no page number should be output for this page
procedure DeletePageNoPos(	removes a page number position from the page number list
APageNo: Integer);	number of the page for which the page number position is to be deleted
procedure InsertPageNoPos(	inserts a page number position in the page number list
APageNo: Integer;	page number to which the page number position applies
X, Y: Double;	the (X, Y) cursor position of the page number
AJustify: TJustify = jRight);	justification for page number output
procedure InsertPageNoPosVoid(	inserts a page number flag indicating no page number should be output for this page
APageNo: Integer);	page number to which the void page number position applies
procedure NumberPages(	numbers document pages at positions defined by page number list
AFormatString: string;	formatting string used by the Format function to format the numbering string can reference two parameters: the page number and/or total number of pages eg "Page %d of %d", or "Pg %d"
AFontIndex): Integer;	indicates the saved font to use for numbering string output
function RetrievePageNoPos(	retrieves page number position details, returning True if record exists
APageNo: Integer;	page number of the page number position record to retrieve (1..PageCount)
var X, Y: Double;	returns the (X, Y) cursor position of the page number position
var AJustify: TJustify): Boolean;	returns the justification for the page numbering string

Setting Fonts (Name and Size) and Saving/Restoring Fonts

To allow font metrics to be correctly managed, avoid using the native VPE font setting procedures (SetFont, SetFontName, SetFontSize) as these procedures effectively reset the font "behind the back" of the ReportInterface. Instead, use:

procedure FontSet(	sets the cursor font and font size
AFontName: string;	the new font name
AFontSize: Integer);	the new font size
procedure FontSetName(	cursor font changes, but current font size is retained
AFontName: string;	the new font name
procedure FontSetSize(	cursor font size changes, but current font is retained
AFontSize: Integer);	the new font size
procedure PushFont;	saves font name and size to an (unlimited) LIFO stack
procedure PushFont(	saves font name and size to an indexed location
AFontIndex: Integer);	indexed location to save to (1..10)
procedure PopFont(	applies the last pushed font name and size from the LIFO stack
ResetLine: Boolean = False);	set True to make the popped font the new line font (by default, current line metrics are retained)
procedure PopFont(	applies font name and size from an indexed location
AFontIndex: Integer;	indexed location to retrieve from (1..10)
ResetLine: Boolean = False);	set True to make the popped font the new line font (by default, current line metrics are retained)

VPE+ Application Reporting Interface

VPE+ Report Interface (TReportInterface) General

Saving and Restoring Cursor Positions

Cursor positions are defined by an X and Y coordinate. The X coordinate indicates the current horizontal print position, and is advanced as text is output. The Y coordinate indicates the current vertical print position, and represents the top of the current line.

procedure PushPos;	saves the cursor position to an (unlimited) LIFO stack
procedure PushPos(ACursorIndex: Integer);	saves the cursor position to an indexed location indexed location to save to (1..10)
procedure PopPos;	applies the last pushed cursor position from the LIFO stack
procedure PopPos(ACursorIndex: Integer);	applies the cursor position from an indexed location indexed location to retrieve from (1..10)
function SavedXPos(ACursorIndex: Integer): Double;	returns the X coordinate of a cursor position previously saved by PushPos(ACursorIndex) indexed location of saved cursor (1..10)
function SavedYPos(ACursorIndex: Integer): Double;	returns the Y coordinate of a cursor position previously saved by PushPos(ACursorIndex) indexed location of saved cursor (1..10)
function MinSavedXPos(ACursorIndexSet: TCursorIndexSet): Double;	returns the least X coordinate from cursor positions saved by PushPos(ACursorIndex) the set of saved cursor indexes (1..10) to include (default [] for ANY cursor)
function MinSavedYPos(ACursorIndexSet: TCursorIndexSet): Double;	returns the least Y coordinate from cursor positions saved by PushPos(ACursorIndex) the set of saved cursor indexes (1..10) to include (default [] for ANY cursor)
function MaxSavedXPos(ACursorIndexSet: TCursorIndexSet): Double;	returns the greatest X coordinate from cursor positions saved by PushPos(ACursorIndex) the set of saved cursor indexes (1..10) to include (default [] for ANY cursor)
function MaxSavedYPos(ACursorIndexSet: TCursorIndexSet): Double;	returns the greatest Y coordinate from cursor positions saved by PushPos(ACursorIndex) the set of saved cursor indexes (1..10) to include (default [] for ANY cursor)

Unit Conversion

VPE+ supports measurement units in millimetres (default), centimetres, or inches. This is set in property TReportInterface.Units (uMM, uCM, or uInch). The underlying VPE may be set to use either centimetres or inches in property TReportInterface.VPEUnits (uCM or uInch). VPE does not support millimetre units.

Thus, unit conversion is necessary where the unit settings of VPE and VPE+ do not match. Take care passing VPE+ units to VPE functions and procedures, and reading VPE measurements back to VPE+. Use the following functions to convert units:

TUnits	available measurement units
uMM	millimetres (default)
uCM	centimetres
uInch	inches
function AsReportUnits(Value: Double): Double;	converts VPE units to VPE+ "report" units. value to convert
function AsVPEUnits(Value: Double): Double;	converts VPE+ units to VPE "native" units. value to convert
function ConvertUnits(Value: Double; FromUnits, ToUnits: TUnits): Double;	converts from one unit type to another. value to convert source and target unit type, TUnits = (uMM, uCM, uInch);

VPE+ Report Interface (TReportInterface) General

Lines and Boxes

procedure DrawHLine(	prints a horizontal line
FromXPos, ToXPos: Double;	specifies the left and right extent of the line
LineYPos: Double = NA;	vertical position (YPos) of line, default = NA = current YPos
APenWidth: Double = NA;	pen width to use (default = NA = pwNormal = 0.3 mm)
APenColour: TColor = NA;	pen colour to use (default = NA = clBlack)
APenStyle: TVPEPenStyle = psSolid);	pen style to use: psSolid (default), psDash, psDot, psDashDot, psDashDotDot
procedure DrawVLine(	prints a vertical line
FromYPos, ToYPos: Double;	specifies the top and bottom extent of the line
LineXPos: Double = NA;	horizontal position (XPos) of line, default = NA = current XPos
APenWidth: Double = NA;	pen width to use (default = NA = pwNormal = 0.3 mm)
APenColour: TColor = NA;	pen colour to use (default = NA = clBlack)
APenStyle: TVPEPenStyle = psSolid);	pen style to use: psSolid (default), psDash, psDot, psDashDot, psDashDotDot
procedure DrawLine(	prints a line between any two points
FromXPos, FromYPos,	specifies the (x, y) from-point of the line
ToXPos, ToYPos: Double;	specifies the (x, y) to-point of the line
APenWidth: Double = NA;	pen width to use (default = NA = pwNormal = 0.3 mm)
APenColour: TColor = NA;	pen colour to use (default = NA = clBlack)
APenStyle: TVPEPenStyle = psSolid);	pen style to use: psSolid (default), psDash, psDot, psDashDot, psDashDotDot
procedure DrawBox(	draws a box using the specified pen and brush, with rounded corners
ALeft, ATop, ARight, ABottom: Double;	defines the box corners
ACornerRadius: Double = NA;	defines the radius of corner rounding (default = NA, or 0 = no rounding)
APenWidth: Double = NA;	pen width for box lines (default = NA = pwNormal = 0.3 mm)
APenColour: TColor = NA;	pen colour for box lines (default = NA = clBlack)
APenStyle: TVPEPenStyle = psSolid;	pen style to use: psSolid (default), psDash, psDot, psDashDot, psDashDotDot
ABrushColour: TColor = clNone);	background brush colour (default = clNone = transparent)
procedure DrawEllipse(	draws an ellipse using the specified pen and brush
ALeft, ATop, ARight, ABottom: Double;	defines the box corners bounding the ellipse
APenWidth: Double = NA;	pen width for ellipse line (default = NA = pwNormal = 0.3 mm)
APenColour: TColor = NA;	pen colour for ellipse line (default = NA = clBlack)
APenStyle: TVPEPenStyle = psSolid;	pen style to use: psSolid (default), psDash, psDot, psDashDot, psDashDotDot
ABrushColour: TColor = clNone);	background brush colour (default = clNone = transparent)

VPE+ Application Reporting Interface

VPE+ Report Interface (TReportInterface) General

Image Management

Images can be output from files or from streams. Streams will use more memory, being stored in the application as well as copied into an open VPE document. Thus, for large images or many images, a file based approach may be more efficient.

ReportInterface helps manage image files by allowing you to register images in a file list. Image registration allows you to re-use the image, and also computes image height and width (where available) for scaling purposes. Images are output using the DrawImage method. Overloaded versions allow you to output an image directly from a file or stream, or to output (and re-use) an image from the maintained image file list.

To add an image to a list, call an appropriate overloaded version of RegisterImage which returns the index of the image in the list.

The image list is automatically cleared and any temporary files deleted as appropriate after each report has completed UNLESS you call the method RetainImageList some time during report execution. Doing so allows you to re-use the same image list for a subsequent report. If you retain an image list, it is your responsibility to free the list by calling ClearImageList. Ensure this is done outside the scope of report execution to avoid the risk of deleting image files BEFORE a report is fully written to the relevant output stream.

You can permanently retain one or more images at the start of the image list by calling LockImageList once those images are registered. Doing so prevents the currently registered images being removed when ClearImageList is called, but allows subsequent images to be added and removed as usual. Effectively, these images are thus available as "global" images, so you can retain a repeatedly used report logo image, for example, across any report generated in the application. To remove the lock, call UnlockImageList which then allows ClearImageList to remove ALL registered images.

NOTE: ClearImageList is otherwise only called automatically when the ReportInterface is destroyed, in which case UnlockImageList is automatically called first.

NOTE: VPE also maintains an internal image cache which allows images to be efficiently re-used within the same report or across multiple reports as required. In VPE+, this cache is flushed by default after each report. If you wish to retain the cache, call the method RetainImageCache some time during report execution. To manually flush the cache, call FlushImageCache outside the scope of report execution.

function RegisterImage(	registers an existing image file in the image list
AFileName: string;	full name of an existing image file
AKind: TImageKind;	the kind of image contained in the file (see TImageKind below)
ATempFile: Boolean;	True if the image file is temporary and should thus be deleted afterwards
APixelHeight: Integer = 0;	the height in pixels of the image (optional - can be used to scale images)
APixelWidth: Integer = 0): Integer;	the width in pixels of the image (optional - can be used to scale images)
	returns the index of the image in the list
function RegisterImage(	registers an image from a database field in the image list
AField: TField;	the database field containing the image
AKind: TImageKind): Integer;	the kind of image contained in the field (see TImageKind below)
	returns the index of the image in the list
	NB Only image types for which Delphi has a specific "image object" are recognised (these are ikBMP, ikEMF, ikGIF, ikICO, ikJPG, ikPNG and ikWMF)
function RegisterImage(	registers an image from an application resource file
AResourceName: string);	name of resource
AKind: TImageKind): Integer;	the kind of image contained in the field (see TImageKind below)
	returns the index of the image in the list
	NB Height & width is read from image types for which Delphi has a specific "image object". (these are ikBMP, ikEMF, ikGIF, ikICO, ikJPG, ikPNG and ikWMF)
	Other resource types are streamed directly to file without height and width details.
TImageKind	ikAuto - used when file extension is not indicative of the image type (must read from file header) ikBMP, ikJPG, ikWMF, ikEMF, ikTIFF, ikGIF, ikPCX, ikPNG, ikICO, ikJNG, ikKOALA, ikIFF, ikMNG, ikPBM, ikPBM_RAW, ikPCD, ikPGM, ikPGM_RAW, ikPPM, ikPPM_RAW, ikRAS, ikTARGA, ikWBMP, ikPSD, ikCUT, ikXBM, ikDDS, ikHDR, ikFAX_G3, ikSGI
procedure ClearImageList;	clears all images from the image list (deleting temporary files as appropriate)
	only call this procedure OUTSIDE the report execution process
procedure LockImageList;	locks currently registered images, preventing them from being removed by ClearImageList.
	use this feature to retain "global" images used repeatedly
	subsequently registered images are not locked unless LockImageList is called again
procedure RetainImageList;	stops the ReportInterface image list being cleared after a given report.
	call any time during the report execution process
procedure FlushImageCache;	flushes the VPE image cache
	only call this procedure OUTSIDE the report execution process

VPE+ Application Reporting Interface

VPE+ Report Interface (TReportInterface) General

procedure RetainImageCache;	stops the VPE image cache from being flushed after a given report. call any time during the report execution process
procedure UnlockImageList;	unlocks the image list, allowing ClearImageList to remove ALL images.
function ImageAtIndex(ImageIndex: Integer): TImageMarkerPtr;	returns a pointer to the image marker (TImageMarker) for the specified image index of required image in image list
TImageMarker = **record** ImageFileName: string; IsTemporary: Boolean; ImageKind: TImageKind; PixelHeight: Integer; PixelWidth: Integer; **end;**	structure marking each image registered in the list full name of an existing image file True if the image file is temporary and should thus be deleted afterwards the kind of image contained in the file (see TImageKind above) the height in pixels of the image (optional - can be used to scale images) the width in pixels of the image (optional - can be used to scale images)
function ImageAspect(ImageIndex: Integer; ASize: Double; ScaleAspect: TImageAspect): Double;	given a height or width (ASize), returns the width or height according to the aspect ratio the index of the image in the image list the size by which to scale the image (either height or width) the aspect required as the result (saReturnHeight or saReturnWidth) only works if a valid TImageMarker height and width are specified for the image
function CreateVPEStream(AField: TField): TVPEStream;	creates and returns a TVPEStream derived from AField the field source for the image
function CreateVPEStream(AStream: TStream): TVPEStream;	creates and returns a TVPEStream derived from AStream the (non-VPE) stream source for the image
procedure CreateImageFile(AField: TField; AKind: TImageKind; AFileName: string; out ImageHeight, ImageWidth: Integer);	creates an image file from a AField the field source for the image the kind of image to be saved a filename for the image the image height and width where this can be determined
procedure DrawImage(ImageIndex: Integer; ALeft, ATop, ARight, ABottom: Double; AFramePenWidth: Double = 0);	draws the image indicated by ImageIndex in the Rect specified the index of the image in the image list the image bounds negative ARight or ABottom values define an actual width or height "NA" causes ARight or ABottom to be calculated according to the aspect ratio. pass a non-zero pen width to draw a frame around the image
procedure DrawImage(AFileName: string; AKind: TImageKind; ALeft, ATop, ARight, ABottom: Double; AFramePenWidth: Double = 0);	draws the image from AFileName in the Rect specified the image filename the kind of image contained in the file the image bounds negative ARight or ABottom values define an actual width or height "NA" causes ARight or ABottom to be calculated according to the aspect ratio. pass a non-zero pen width to draw a frame around the image
procedure DrawImage(AField: TField; AKind: TImageKind; ALeft, ATop, ARight, ABottom: Double; AFramePenWidth: Double = 0);	draws the image from AField in the Rect specified the image field name the kind of image contained in the file the image bounds negative ARight or ABottom values define an actual width or height "NA" causes ARight or ABottom to be calculated according to the aspect ratio. pass a non-zero pen width to draw a frame around the image
procedure DrawImage(AStream: TStream; AKind: TImageKind; ALeft, ATop, ARight, ABottom: Double; AFramePenWidth: Double = 0;	draws the image from AStream in the Rect specified the image stream the kind of image contained in the file the image bounds negative ARight or ABottom values define an actual width or height "NA" causes ARight or ABottom to be calculated according to the aspect ratio. pass a non-zero pen width to draw a frame around the image

NOTE: DrawImage for AField and AStream are useful for once-only streamed image output. They create and free a TVPEStream which cannot therefore be re-used in the VPE image cache. To persist and re-use image streams, create your own VPEStream using CreateVPEStream, and output them with overloaded DrawImage for CPEStream. If using multiple image streams, be aware that large amounts of memory may be used. Consider using file based images instead.

NOTE: If the VPE property PictureBestFit is True (default is False) and all four coordinates are specified, the image will be scaled to the maximum size fitting within the defined Rect according to the aspect ratio. Otherwise, the VPE property PictureKeepAspect (default True) controls whether the aspect ratio is honoured, or the image is stretched.

VPE+ Report Interface (TReportInterface) Line & Font Metrics

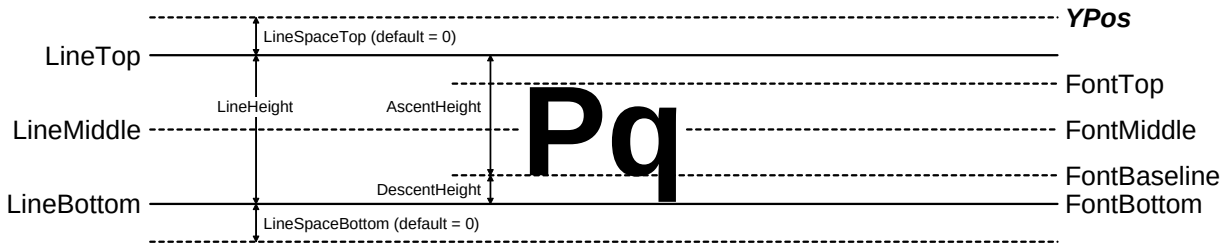
ReportInterface maintains a line cursor position while printing, located at point (XPos, YPos).

Print functions generally advance the horizontal cursor position (XPos) by the width of the text printed, but maintain the same vertical position (YPos) used for output. With multi-line output (TextBlocks), the final YPos will reflect the last line output, and whether the block is set to finish on a new line or not.

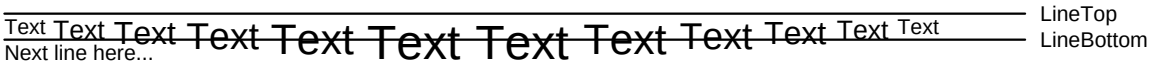
By default, YPos corresponds to LineTop, but increasing line spacing by setting LineSpaceTop will displace LineTop downward by that value (and similarly the other positional line metrics) while leaving the cursor YPos unchanged. Setting LineSpaceBottom has no impact on these values for the current line, but displaces the next line further downward.

Note that the value of LineHeight depends only on the applied FONT - it does NOT change with increased line spacing (which is extra space applied above or below the "font line"). A call to NewLine, however, will advance YPos by the sum of LineSpaceTop + LineHeight + LineSpaceBottom.

For the purpose of the examples below, we are assuming that default line spacing is applied (ie both LineSpaceTop and LineSpaceBottom are zero). The line and font metrics are:

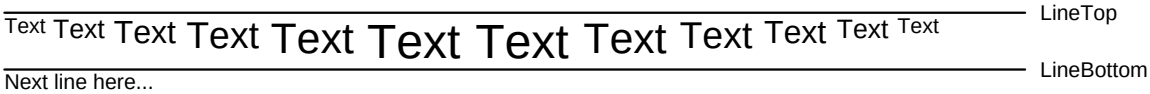


The font applied at the start of a line (the "line font") determines how far YPos is advanced with a call to NewLine, rather than any subsequent font that may be applied at the cursor (the "cursor font").
This means that if a change is made to a larger font, say, that font will overrun the extent of the line currently recognised by ReportInterface as shown here:



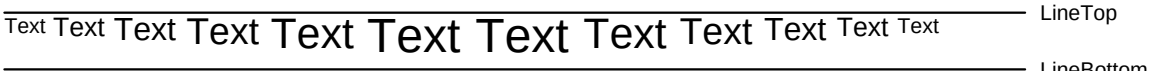
Next line here...

To correctly encompass the full extent of the largest font used in this line, call ResetLineHeight while that font is applied. The current LineHeight will then reflect the LineHeight of the current cursor font. Alternatively, call ResetLineFont to change the font marked as the "line font" (and hence the LineHeight as well). The result is:



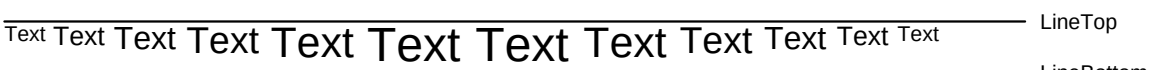
Next line here...

Notice that each "Text" word, although aligned correctly to a common YPos with respect to its own line metrics, is not aligned evenly with adjacent "Text" words (ie the top of each word varies from font to font). To align the words by, say, FontTop, mark FontTop with the first font, and then reset FontTop to this value for each new font assigned giving:



Next line here...

Finally, because the "Text" words in larger fonts have effectively been raised in position to align FontTop with the original line font, there is an extra gap between the baseline of the largest words and the bottom of the line. To close this gap, mark the LineBottom position after printing the largest words (so the gap will be appropriate to this font and its adjusted position), then re-instate it at the end of the line when the font is correctly set for the next line (eg set LineBottom to a "MarkedLineBottom"). Now, before calling NewLine, call ResetLineHeight to ensure a LineHeight feed suitable for the new font. The next line will now be positioned appropriately below the largest font used:



Next line here...

NOTE: Font and line metrics are fairly logical and predictable, but manipulating text placement can get confusing, especially when some positional function calls force changes in these metrics that may not have been anticipated. Generally, this becomes an exercise in being aware of what cursor position, font and line height are active at any given time, and the impact of your code on line metrics.

VPE+ Report Interface (TReportInterface) Line & Font Metrics

Manipulating Cursor Position:

procedure AdvanceXPos(Advance: Double);	advances the X (horizontal) position of the cursor distance to advance the cursor X position
procedure AdvanceYPos(Advance: Double);	advances the Y (vertical) position of the cursor distance to advance the cursor Y position
procedure CursorHome;	restores the cursor position to the top left of a band
procedure CursorLeft;	restores the cursor position to the left of a band
procedure CursorTop;	restores the cursor position to the top of a band
procedure CursorTo(X, Y: Double);	resets the cursor position new cursor X position (XPos) new cursor Y position (YPos)
procedure NewLine(LineCount: Integer = 1);	advances the cursor position to the beginning of a subsequent line number of lines to advance the cursor (default = 1)
procedure NewPage;	begins a new page
property XPos;	sets or returns horizontal position of cursor
property YPos;	sets or returns vertical position of cursor (equates to LineTop)

Manipulating Lines and Font Alignment:

TLineMetric	represents the available font metrics (refer to font metric diagramme above) lmFontTop, lmFontMiddle, lmFontBaseline, lmFontBottom, lmLineTop, lmLineMiddle, lmLineBottom
function AscentHeight: Double;	returns the ascent height for the cursor font
function AscentHeight(AFontIndex: Integer): Double;	returns the ascent height of a saved font index of saved font (1..10)
function CapitalHeight: Double;	returns the capital height for the cursor font
function CapitalHeight(AFontIndex: Integer): Double;	returns the capital height of a saved font index of saved font (1..10)
function DescentHeight: Double;	returns the descent height for the cursor font
function DescentHeight(AFontIndex: Integer): Double;	returns the descent height of a saved font index of saved font (1..10)
function LineHeight: Double;	returns the line height for the cursor font
function LineHeight(AFontIndex: Integer): Double;	returns the line height of a saved font index of saved font (1..10)
function RenderedLineHeight: Double;	renders a cursor font test character and returns its line height (used internally - use LineHeight to return the already calculated value)
function TextWidth(Text: string; TextStyle: TTextStyle = tsNormal): Double;	returns the width of a string using the current font the text to measure font style to apply (default = tsNormal)
procedure ResetLineHeight;	forces LineHeight to reflect height of currently applied (cursor) font (rather than that of the "line font" applied at the beginning of the line)
procedure ResetLineFont;	forces the currently applied (cursor) font to be regarded as the line font (calls ResetLineHeight)
property FontTop;	returns vertical position of fonts top extent
property FontMiddle;	returns vertical position of fonts middle point
property FontBaseline;	returns vertical position of fonts baseline
property FontBottom;	returns vertical position of fonts bottom extent (equals LineBottom)
property LineTop;	returns vertical position of lines top extent (equals YPos)
property LineMiddle;	returns vertical position of lines mid-point
property LineBottom;	returns vertical position of lines bottom extent (equals FontBottom)

VPE+ Application Reporting Interface

VPE+ Report Interface (TReportInterface) Line & Font Metrics

Manipulating Font Size:

function FontSizeFitHeight(	returns a font size to fit a font to a given AscentHeight
AFontName: string;	the font to test
AFontFit: TFontFit;	the fittng mechanism to apply
ffReduceToFit	if larger, reduce font size until fits (no increasing)
ffIncreaseToFit	if smaller, increase font size to largest fit (no decreasing)
ffLargestFit	increase or decrease font size to largest fit
StartFontSize: Integer;	the initial font size to test with
AFitHeight: Double;	the AscentHeight to fit to
RejectOddFontSize: Boolean): Integer;	True to only test/return even font sizes (default= True)
function FontSizeFitWidth(	returns a font size to fit a string to a given width
AFontName: string;	the font to test
AFontFit: TFontFit;	the fittng mechanism to apply
ffReduceToFit	if larger, reduce font size until fits (no increasing)
ffIncreaseToFit	if smaller, increase font size to largest fit (no decreasing)
ffLargestFit	increase or decrease font size to largest fit
StartFontSize: Integer;	the initial font size to test with
PrintStr: string;	the text to fit
AFitWidth: Double;	the width to fit the text to
RejectOddFontSize: Boolean): Integer;	True to only test/return even font sizes (default= True)

VPE+ Report Interface (TReportInterface) Line & Font Metrics

Re-Aligning the Cursor Font:

The cursor position can be adjusted to place text relative to other objects or text elements. However, care is sometimes needed when using these manipulations to avoid unexpected results.

In particular, remember that font line metrics (LineTop, LineMiddle, LineBottom, FontTop, FontMiddle, FontBaseline, FontBottom) are relative to a given vertical position. Usually this is the current YPos (default), although you can specify any reference position when re-aligning text (BaseYPos in the methods below). Resetting YPos or calls to methods like NewLine and AdvanceYPos, will, of course, directly move the YPos, but so too will calls to the "AlignTo" methods. The PrintPos method does NOT change the YPos. Given the same YPos, changing the font or font size changes the relative line metrics, while setting LineSpaceTop displaces them all accordingly. There can be quite some potential for confusion!

These procedures realign the *cursor* font, moving the cursor position accordingly:

procedure AlignToCursorFont(	aligns a cursor font metric to another cursor font metric
AlignBy,	cursor font metric to align by
AlignTo: TLineMetric;	cursor font metric to align to
BaseYPos: Double = NA): Double;	vertical reference position or origin (default = NA = current YPos)
procedure AlignToLineFont(	aligns a cursor font metric to a font metric of the line font
AlignBy,	cursor font metric to align by
AlignTo: TLineMetric;	line font metric to align to
BaseYPos: Double = NA): Double;	vertical reference position or origin (default = NA = current YPos)
procedure AlignToSavedFont(	aligns a cursor font metric to the font metric of a saved font
AFontIndex: Integer;	index of saved font (1..10) to align to
AlignBy,	cursor font metric to align by
AlignTo: TLineMetric;	saved font metric to align to
BaseYPos: Double = NA): Double;	vertical reference position or origin (default = NA = current YPos)
procedure AlignToYPos(	to align a cursor font metric at a given vertical position
AlignBy: TLineMetric;	cursor font metric to align by
YAlignTo: Double);	vertical position (YPos) to align to

These functions return the YPos required to realign the *cursor* font (without moving the cursor position):

function YPosAlignToCursorFont(	returns the YPos that aligns a cursor font metric to a metric of the cursor font
AlignBy,	cursor font metric to align by
AlignTo: TLineMetric;	cursor font metric to align to
BaseYPos: Double = NA): Double;	vertical reference position or origin (default = NA = current YPos)
function YPosAlignToLineFont(	returns the YPos that aligns a cursor font metric to a metric of the line font
AlignBy,	cursor font metric to align by
AlignTo: TLineMetric;	line font metric to align to
BaseYPos: Double = NA): Double;	vertical reference position or origin (default = NA = current YPos)
function YPosAlignToSavedFont(	returns the YPos that aligns a cursor font metric to a metric of a saved font
AFontIndex: Integer;	index of saved font (1..10) to align to
AlignBy,	cursor font metric to align by
AlignTo: TLineMetric;	saved font metric to align to
BaseYPos: Double = NA): Double;	vertical reference position or origin (default = NA = current YPos)
function YPosAlignToYPos(	returns the YPos that aligns a cursor font metric to a given vertical position
AlignBy: TLineMetric;	metric to align by
YAlignTo: Double): Double;	vertical position (YPos) to align to

VPE+ Application Reporting Interface

VPE+ Report Interface (TReportInterface) Report Execution

Executing Reports

A given report is based on a locally placed ReportWriter component which is passed to the ReportInterface procedure ExecuteReport together with a TSetupMode. Reports may also be generated as a batch, or existing reports added to a batch and then previewed or output collectively.

procedure ExecuteReport(

AReportWriter: TReportWriter;	ReportWriter component used to generate the report
ASetupMode: TReportSetupMode = smSetup;	the setup mode controlling how the report is generated (default = smSetup)
AFormatIndex: Integer = 0;	index to identify a custom format when ASetupMode = smCustomFile (default = 0)
AReportTag: Integer = 0;	optional tag to identify a given report (default = 0)

TReportSetupMode	to specify how to go about generating the report
smSetupRepeat	display setup form prompt prior to generating report, repeat cycle until user cancels
smSetup	display setup form prompt prior to generating report (default)
smPDFFile	direct output to PDF file
smHTMLFile	direct output to HTML file
smXMLFile	direct output to XML file
smODTFile	direct output to ODT file
smVPEFile	direct output to VPE file
smCustomFile	direct output to custom file (identified by AFormatIndex)
smEmailPDFFile	direct output to PDF file email
smEmailHTMLFile	direct output to HTML email
smEmailXMLFile	direct output to XML email
smEmailODTFile	direct output to ODT email
smEmailVPEFile	direct output to VPE email
smEmailCustomFile	direct output to custom file (identified by AFormatIndex) email
smPreview	show the report in preview
smPrinter	send the report directly to the printer or output device

Report Generation Process:

The general sequence of events for generatign a report is:

1. System Configuration
   Configure and adjust ReportInterface or ReportWriter settings prior to report generation in ReportWriter event **OnConfigure;**
   NB The VPE document is NOT open at this point, and cannot be referenced until OnGenerateStart fires. It is closed after OnGenerateEnd.

2. Status Form Display
   The status form (if required) is created so that status messages may be displayed per ReportInterface.StatusLabel.

3. Setup Form Display
   The setup form is processed (if required)
   ReportWriter events OnSetupBefore, OnSetupValidate, OnSetupAfter allow setup configuration, validation, and acceptance/rejection.
   Read the setup state from properties SelectedSetupAction, SelectedSetupMode, SelectedFormat and SelectedFormatIndex.

4. Generate Report
   ReportWriter events OnGenerateStart, OnGenerateEnd demarcate the generation process (VPE document is open during this phase).
   ReportWriter event OnCustomFormatGenerate handles custom format generation.
   ReportWriter event OnGenerate handles VPE generation - report code started here.

   NOTE: Use the generate Start/End events to start & end data transactions as they still fire on report exceptions..

5. Cycling Report Setup
   If smSetupRepeat mode is used, report setup and generation is repeated until the user cancels.

VPE+ Application Reporting Interface

VPE+ Report Interface (TReportInterface) Batched Report Execution

Executing Batched Reports

A report batch allows you to manage multiple reports collectively. For example, you can allow the user to multi-select individual reports for output or emailing, change the format of a given report, and have reports of different formats within the same batch.

Any report may be generated as part of a batch, or existing reports added to a batch. The end user can add reports to a batch by opening existing files while in Preview. Reports may be added in code by using the ReportBatch.Add method.

Only reports using the native VPE file format can actually be previewed in the default preview form, but any other file format (text, HTML, PDF etc) may still be included in a batch. A non-VPE format will simply show in preview with an "unable to preview" message. Of course, it is quite possible to design your own custom preview form which DOES display other formats if you wish (eg by allowing a text file to be displayed in a memo component, or integrating a third party PDF viewing component etc).

Multiple reports are generated by calling ExecuteBatchReport for each report. Initially, a VPE report is always sent to a temporary file. By passing a SetupMode specifying a file-format to this method (eg smPDFFile), you set the default target format should the report be subsequently exported. With any other SetupMode, the ReportWriter.OutputDefaultFormat will be applied instead. For a custom format, pass smCustomFile with the appropriate FormatIndex.

An initial "batch setup only" prompt (using the standard setup form) can be presented by calling BatchSetup. This allows the user to provide report parameters and output selection. NO actual output is performed - it is a prompt only. Read the users response via the properties SelectedSetupAction, SelectedSetupMode, SelectedFormat and SelectedFormatIndex. Reports may then be generated accordingly with calls to ExecuteBatchReport.

Having generated a report batch, use BatchOutput to handle collective output. Custom report files can be filed or emailed, but not previewed or printed unless you provide a suitable means to do so. BatchOutputPrompt can be used to prompt the user for (and execute) an output option for any selection of reports included in the batch.

IMPORTANT: All batches must be closed with a call to BatchClose to clean up temporary files and allocated batch resources.

procedure ExecuteBatchReport(

AReportWriter: TReportWriter;	ReportWriter component used to generate the report(s)
ASetupMode: TBatchSetupMode;	the setup mode controlling how the batched reports are generated
smVoid	no immediate VPE export format output (ie output to be handled later, collectively)
smPDFFile	export to PDF file
smHTMLFile	export to HTML file
smXMLFile	export to XML file
smODTFile	export to ODT file
smVPEFile	export to VPE file
smCustomFile	generate custom format indicated by AFormatIndex
AFormatIndex: Integer = 0);	the format index associated with SetupMode smCustomFile (default = 0)

Other Batch Functions, Procedures & Properties

function BatchOutput(	outputs a report batch, returning True if output handled
AReportWriter: TReportWriter;	ReportWriter managing output
ASetupAction: TOutputSetupAction;	specifies the output action to take
saPreview	preview
saPrint	print directly to output device as set (no prompt)
saFile	save to selected file format
saEmail	save to selected file format and present result for emailing
AllowConfirmDone: Boolean = True;	set False to suppress the output confirmation prompt
ForceFileNamePrompt: Boolean = False): Boolean;	set True to force a filename prompt
function BatchOutputPrompt(	prompts user for a batch output option (returns saCancelled if cancelled)
AReportWriter: TReportWriter;	ReportWriter managing output
ASetupActions: TOutputSetupActions	set of batch output actions to include in the prompt
saPreview	allow preview output
saPrint	allow print output
saFile	allow file output
saEmail	allow email output
): TSetupAction;	returns the selected setup action (saCancelled, or one of the listed TOutputSetupActions)

VPE+ Report Interface (TReportInterface) Batched Report Execution

Other Batch Functions, Procedures & Properties *continued*

function SystemBatchFileName(	Returns a validated file name for the indexed report = "FileNameStub[Succ(ABatchIndex)]"
ABatchIndex: Integer;	index of batched report
ASystemOutputFormat: TSystemFormat;	required target output format (dictates file extension)
ofPDFFile	to PDF file
ofODTFile	to penDocument Text file
ofHTMLFile	to HTML file
ofXMLFile	to XML file
ofVPEFile	to VPE (native) file
AFileNameStub: string): string;	report filename stub - if not provided, default is "Report"
function UserBatchFileName(	Returns a validated file name based on AFileName
ABatchIndex: Integer;	index of batched report
AFormatIndex: Integer;	custom format index indicating which custom format extension to use
AFileName: string);	custom report filename; if not provided = "Report[Succ(ABatchIndex)]"
procedure BatchClose;	closes all reports in a batch. Must be called to clean up temporary files and batch resources.
procedure BatchItemClose(	closes a specific batch report
ABatchIndex: Integer);	index of batch report to close, 0..Pred(ReportBatch.Count)
procedure BatchSetup(	presents a pre-output batch setup prompt (obtain output selection using ReadBatchSetupResult
AReportWriter: TReportWriter);	ReportWriter managing output
property BatchIndex: Integer;	reflects the index of the current report in ReportBatch
property Batching;	True when batching reports (as a result of calling ExecuteBatchReport) Non-batched reports cannot be executed while batching is in progress
property ReportBatch: TList;	gives access to the list of batched reports (TBatchItem's)

Adding Existing Reports to a Batch

Populate a TBatchItem record and call ReportBatch.Add(ABatchItemPtr);
ReportBatch will handle disposal of the BatchItem.

TBatchItem = **record**	
SourceVPEFileName: string;	filename of source report (usually VPE format)
SourceVPEFileType: TSourceVPEFileType;	sftVoid for custom formats; sftTempVPE to delete source VPE file; else sftKeepVPE
TargetFormat: TTargetFormat;	format of output file to be generated (if needed)
TargetFileName: string;	name of output file to be generated (if needed)
TargetIsTemp: Boolean;	True to delete the target file after processing
UserSelected: Boolean;	True if flagged as user-selected by default
OverwriteOK: Boolean;	True if user has OK'ed report file overwrite
UserOptionTag: Integer;	optional tag assigned to report
FormatIndex: Integer;	0 = standard formats, else users FormatIndex (1..CustomFormatCount)
FilterIndex: Integer;	this marks save dialogue FilterIndex (1-based)
Description: string;	report description (optional, default = TitleSystem)
end;	

Setting Descriptions for Batched Reports

A report description can be specified in property ReportWriter.ReportDescription (eg set it in OnConfigure) or left to default to ReportInterface.DefaultReportDescription (default = "Report"). As a report executes, the description can also be further customised in ReportWriter.OnDescribeReport.

VPE+ Report Interface (TReportInterface) Report Run Execution

Bulk Report Runs

Multiple (bulk) reports can be generated as a "report run". You might do this to generate a series of subscription reports for your customer base, for example. A report run is managed within the scope of a single (override) status form instance. You can set this form up any way you like. It can show progress details and current report information etc.

To engineer the run, begin by creating an instance of your status form (eg MyStatusForm). Pass it a parameterless callback procedure in the constructor, or by assigning the callback procedure to a form property after creation. You code this procedure yourself to generate the run reports. By calling it in response to a button click or in the OnActivate event, for example, the status form starts the actual run.

A suitable constructor might be declared as:

```
constructor CreateModalStatus(VPEInterface: TReportInterface; ReportRunProcedure: TReportRunCallback = nil);

type TReportRunCallback = procedure() of object;
```

To execute the run, call the ReportInterface method ExecuteReportRun(MyStatusForm). It will show MyStatusForm modally.

A label on MyStatusForm can be assigned to VPEInterface.StatusLabel so that VPEInterface.DisplayStatus messages are automatically displayed. Otherwise, you can update the (modal) status form however you wish.

```
function ExecuteReportRun(                    executes a "report run" using AStatusForm to display status details
    AStatusForm: TForm): TModalResult;
```

VPE+ Report Interface (TReportInterface) Output Devices

Device Management

The ReportInterface manages report output devices through its (property) DeviceManager which maintains an independent list of available devices (DeviceList) including basic device details stored as TDevice definitions.

The DeviceList is automatically initialised on demand (eg when the first report is executed) and is simply a string list matching the system printer list with associated TDevice objects. The index of the default device is noted in DefaultDeviceIndex. Until other devices are referenced, their TDevice records are not populated.

A device may be selected prior to ExecuteReport using procedures SelectDevice or SwitchDevice. However, device properties (DeviceCopies, DeviceCollate, DeviceDuplex) are defaulted on ExecuteReport, so must be set once the report is actually underway.

The following properties and procedures are available through the ReportInterface.DeviceManager property:

procedure AssignBinList(ADeviceIndex: Integer; AStringList: TStrings);	assigns a list of bins from the specified device index of the device in Device list target string list to assign to
procedure AssignDeviceList(AStringList: TStrings);	assigns a list of devices target string list to assign to
procedure DefaultDeviceSettings;	defaults the active devices settings (1 copy, collation on, duplexing off)
procedure FinaliseDeviceList;	clears the Device list of "device markers" (list will be re-initialised on demand)
procedure InitialiseDeviceList;	populates the Device list if not already done (call FinaliseDeviceList first to force a re-initialisatio
function CanCollate: Boolean;	True if CollateMethod is CollateManual, or active device supports collation
function DeviceIndex: Integer;	index of the active device
function DeviceIndex(ADeviceName: string; ExactName: Boolean = False): Integer;	returns the Device index for the named device name of device to find unless True, returns first device containing ADeviceName string
function DeviceName: string;	returns name of active device
function DeviceName(ADeviceIndex: Integer): string;	returns name of indexed device index of device for which to return a name
function Device: TDevice;	returns the TDevice object representing the active device
function Device(ADeviceIndex: Integer): TDevice;	returns the TDevice object representing the indexed device index of device for which to return a TDevice
function Device(ADeviceName: string; ExactName: Boolean = False): TDevice;	returns the TDevice object representing the named device name of device to find unless True, returns first device containing ADeviceName string
function ReportDevicesExist: Boolean;	True if any device is defined; False if no devices are available
function SelectDevice(ADeviceName: string): Boolean; ExactName: Boolean = False): TDevice;	returns True if the named Device is made "active" name of device to find unless True, activates the first device containing ADeviceName string
function SelectDevice(ADeviceIndex: Integer): Boolean;	returns True if the indexed Device is made "active" index of device to activate
function SelectDevice: Boolean;	returns True if the default Device is made "active"
function SwitchDevice: Boolean;	prompts for a device to activate, returning True if successful
property CollateMethod: TCollateMethod;	PrinterCollation for printer dependent collation (if capable) ManualCollation to manually implement collation printing 1 copy at a time
property DefaultDeviceIndex: Integer;	returns the index of the (Windows) default device
property DeviceCollate: Boolean;	sets collation state (see CollationMethod)
property DeviceCopies: Integer;	sets number of copies to output
property DeviceCount: Integer;	returns number of devices available
property DeviceDuplex: TDuplex;	sets duplex state (dupSimplex, dupHorizontal, dupVertical)
property DeviceList: TStrings;	returns a list of devices

VPE+ Report Interface (TReportInterface) Output Devices

Device Objects Maintained by DeviceManager

As each available device is accessed, a TDevice object is created to represent it.
These objects can be referenced using the "Device" properties of ReportInterface.DeviceManager.

procedure AssignBinList(AStringList: TStrings);	assigns a list of bins from the specified device target string list to assign to
function ActiveBinID: Integer;	Device ID of the active bin for this device
function ActiveBinIndex: Integer;	index of the active bin in the bin list for this device
function BinNameByIndex(ABinIndex: Integer): string;	returns name of indexed bin index of bin to be named
function BinNameByID(ABinID: Integer): string;	returns name of ID'ed bin device bin ID of bin to be named
function BinIDByIndex(ABinIndex: Integer): Integer;	returns device bin ID for indexed bin index of bin to be ID'ed
function BinIndexByID(ABinID: Integer): Integer;	returns index of ID'ed bin device bin ID of bin to be indexed
function GetBinID(var ABinID: Integer): Boolean;	returns True if identifies ID of devices bin variable to hold returned device bin ID
property CopyLimit: LongInt;	returns the devices copy limit
property DeviceIndex: Integer;	returns the devices index in the device list
property DeviceName: string;	returns the devices name
property SupportCollate: Boolean;	returns True if the device supports collation
property SupportDuplex: Boolean;	returns True if the device supports duplexing
property SupportOrientation: Boolean;	returns True if the device supports paper orientation

VPE+ Report Interface (TReportInterface) Error Control

Aborting Reports and Error Control

property AbortReason: string;	Core error message which gets set internally to: - exception message for exceptions generated during the generate process - "ReasonSetupRejected" string returned from OnSetupAfter if setup rejected - "Reason" string returned via procedure AbortReport
property ReportStatus: TReportStatus;	Reflects the "completion status" of a report following generation.
TReportStatus	
rsReportOK	report completed normally
rsSetupCancelled	user clicked <Cancel> in report setup (report never generated)
rsSetupRejected	no filename or OnSetupAfter not Accepted
rsOverwriteRejected	user rejected file overwrite
rsReportCancelled	user cancelled report (eg per status form <Cancel>)
rsReportAborted	user aborted during generation (eg per status form <Abort>)
rsReportError	failed to generate report
function AbortReasonMessage: string;	Returns a compound error message string reflecting: - the report completion status set by procedure AbortReport - the string returned by property AbortReason
function ReportAborted(	returns True if the report has been aborted
ShowReason: Boolean = False): Boolean;	Set True to display the AbortReasonMessage dialogue
procedure AbortReport(	call AbortReport to halt a report
AbortType: TReportAbort;	assign a general type to the abort call (see below)
Reason: string = ');	optionally add more detail for the reason
TReportAbort	subrange of TReportStatus reflecting report abort states
rsSetupCancelled..rsReportError	(see TReportStatus details above)

Exceptions raised during report execution can be trapped in the event TReportWriter.OnGenerateException. Either handle the exception and set event parameter ReRaise to False, or allow the exception to be raised by leaving ReRaise True.

Note also that executing a report clears and residual report status and error information relating to any previous report.

Error Control With Report Runs

The above methods relate to errors that might occur while generating a single report. When executing a report run, however, we must be concerned with errors that might occur between multiple reports, and, in particular, a request to STOP a report run. This requires a set of "run status flags" separate from those described above because report status flags pertain to a single report and are "cleared" with each new report, whereas a report run spans multiple reports.

The failure of a given report to generate does not necessarily mean a report run should stop. Unless another run error state is already set, a report error will automatically set a run state of "rsRunReportError". If you want the run to continue, you can call the method ClearRunError which will set the state "rsRunWithErrors" instead.

A call to AbortReportRun does not directly stop a run, but rather sets an abort state which you can periodically test during the run. The easiest way to do this is to poll the run status before each report is generated. Call Application.ProcessMessages first to allow any state changes in the message queue to be processed, then test RunAborted for the run state. A specific status can be determined by referencing the property RunStatus.

property RunStatus: TRunStatus;	Reflects the status of a "report run".
TRunStatus	
rsRunOK	run completed normally
rsRunWithErrors	run completed, but with errors in one or more reports
rsRunCancelled	user clicked <Cancel> in setup before run started
rsRunAborted	user aborted run (eg per status form)
rsRunReportError	an error occurred generating a given report
function RunAborted: Boolean;	returns True if the run has been aborted
procedure AbortReportRun(	call AbortRun to halt a run
AbortType: TRunAbort;	assign a general type to the abort call (see below)
Reason: string = ');	optionally add more detail for the reason
TRunAbort	subrange of TRunStatus reflecting run abort states
rsRunCancelled..rsRunReportError	(see TRunStatus details above)
procedure ClearRunError;	resets a run error state to rsRunWithErrors (so a run can continue)

VPE+ Report Interface (TReportInterface) Print Functions

Use PrintPos for single line text output. PrintPos will advance the cursor XPos to the end of the output text, but will NOT change the cursor YPos.

```
procedure PrintPos(
    Text: string;                          text to print
    Justify: TPrintJustify;                text justification (jLeft, jRight, jCentre), default = jLeft
    X: Double;                             horizontal XPos reference point about which to justify text
    Y: Double;                             vertical YPos at which to output text
    LeftLimit: Double;                     left text margin or limit (NA = default = BandLeft)
    RightLimit: Double;                    right text margin or limit (NA = default = BandRight)
    TextStyle: TTextStyle = tsNormal;      font style to use for text output (Bold, Underline, Italic, Strikeout)
                                           else tsB, tsU, tsI, tsS, tsBU, tsBI, tsBS, tsIU, tsSU, tsIS, tsBIU, tsBSU, tsBIS, tsISU, tsBISU
    TruncateMode: TStringCutMode = scmChar);   truncation method (scmWord or scmChar) - by whole word, or by character
```

Additional overloaded options:

```
procedure PrintPos(
    Text: string;
    TextStyle: TTextStyle = tsNormal;
```

```
procedure PrintPos(
    Text: string;
    Justify: TPrintJustify;
    X: Double;
    TextStyle: TTextStyle = tsNormal;
    TruncateMode: TStringCutMode = scmChar);
```

```
procedure PrintPos(
    Text: string;
    Justify: TPrintJustify;
    X, Y: Double;
    TextStyle: TTextStyle = tsNormal;
    TruncateMode: TStringCutMode = scmChar);
```

For printing by character alignment (eg a decimal point) - justifies 1st occurrence of JustifyChar at X:

```
procedure PrintPos(
    Text: string;
    JustifyChar: Char;
    X: Double;
    TextStyle: TTextStyle = tsNormal;
```

For single line text output followed by NewLine;

```
procedure PrintLine(
    Text: string;
    Justify: TPrintJustify = jLeft;
    TextStyle: TTextStyle = tsNormal);
```

```
procedure PrintLine(
    Text: string;
    Justify: TPrintJustify = jLeft;
    X: Double;
    TextStyle: TTextStyle = tsNormal;
    TruncateMode: TStringCutMode = scmChar);
```

```
procedure PrintLine(
    Text: string;
    Justify: TPrintJustify = jLeft;
    X, Y: Double;
    TextStyle: TTextStyle = tsNormal;
    TruncateMode: TStringCutMode = scmChar);
```

```
procedure PrintLine(
    Text: string;
    Justify: TPrintJustify = jLeft;
    X, Y, LeftLimit, RightLimit: Double;
    TextStyle: TTextStyle = tsNormal;
    TruncateMode: TStringCutMode = scmChar);
```

VPE+ Report Interface (TReportInterface) Print Functions

Line Spacing

Line spacing can be increased by adding space above and/or below the font line. By default, there is no such extra space. Use the method SetLineSpacing, or set properties LineSpaceTop and LineSpaceBottom to apply spacing. Set these properties to zero, or call ClearLineSpacing to remove extra spacing.

Note that the value of LineHeight does not change with increased line spacing. This is because LineHeight reflects the size of the FONT applied in all cases, and not any extra spacing between lines. Positional font metrics LineTop, LineMiddle, LineBottom, FontTop, FontMiddle, FontBaseline and FontBottom will, however, be displaced downward by the value of LineSpaceTop. They are not affected by changes to LineSpaceBottom.

The vertical cursor position, YPos, always refers to the top-most position above LineSpaceTop.

When using TabBoxes, the drawn box is extended to cover any additonal line spacing.

property LineSpaceTop: Double;	value of extra top line spacing (default = 0).
property LineSpaceBottom: Double;	value of extra bottom line spacing (default = 0).
procedure SetLineSpacing(	applies extra line spacing (space above or below the font line).
ALineSpaceTop,	extra space to include above the font line (NA for no change).
ALineSpaceBottom: Double);	extra space to include below the font line (NA for no change).
procedure ClearLineSpacing;	clears any extra line spacing (both LineSpaceTop and LineSpaceBottom).

VPE+ Report Interface (TReportInterface) Line Tabs

Horizontal line tabs are a useful means of implementing single line columnar output. Text is printed between defined text margins within the tab using PrintTab, and can be enhanced with tab boxes and background shading.

Tabs are sequentially defined to an "active list" using the method SetLineTab. Such a list can be saved to an indexed location or to a stack using PushTabList, and retrieved and re-applied using PopTabList. Optionally modify tab presentation using SetTabBox and SetTabBoxes (for tab box lines and shading) or the various "bump" features: BumpTabBox, BumpTabBrush, BumpTabJustify, BumpTabMargins, BumpTabPenColour, BumpTabPenStyle.

Justified text is output to a tab between left and right text margins using the method PrintTab. A variety of other methods and properties allow you to manipulate and exploit tabs in a report:

Setting / Defining Line Tabs

function SetLineTab(	adds a sequential tab setting, returning index of added tab (1-based)
ALeftPos,	start (left) position of tab (use NA to follow previous tab)
AWidth: Double;	width of tab
AJustify: TPrintJustify;	text justification (jLeft, jRight, jCentre)
ALeftMargin: Double = NA;): Integer;	margin between left tab border and start of text (NA = default = 0.5 mm)
ARightMargin: Double = NA): Integer;	margin between right tab border and end of text (NA = default = 0.5 mm)

NOTE: Use a text block (TTextBlock) to justify text jBlock or jBlockFull.

procedure SetTabBox(	defines a tab box for a given tab
ATabIndex: Integer;	index of tab to set (1..Count)
ALeftPenWidth,	pen width for left box side (NA = as is; pwNone = 0 = no line)
ATopPenWidth,	pen width for top box side (NA = as is; pwNone = 0 = no line)
ARightPenWidth,	pen width for right box side (NA = as is; pwNone = 0 = no line)
ABottomPenWidth: Double;	pen width for bottom box side (NA = as is; pwNone = 0 = no line)
ABrushColour: TColor = clNone); overload;	background shading colour to apply (NA = as is; default = clNone = transparent)
procedure SetTabBox(	defines a tab box for a given tab with the same PenWidth for all box lines
ATabIndex: Integer;	index of tab to set (1..Count)
APenWidth: Double = NA;	pen width for all box sides (default = NA = as is; pwNone = 0 = no line)
ABrushColour: TColor = clNone); overload;	background shading colour to apply (NA = as is; default = clNone = transparent)
procedure SetTabBox(	defines a tab box for the last defined tab
ALeftPenWidth,	pen width for left box side (NA = as is; pwNone = 0 = no line)
ATopPenWidth,	pen width for top box side (NA = as is; pwNone = 0 = no line)
ARightPenWidth,	pen width for right box side (NA = as is; pwNone = 0 = no line)
ABottomPenWidth: Double;	pen width for bottom box side (NA = as is; pwNone = 0 = no line)
ABrushColour: TColor = clNone); overload;	background shading colour to apply (NA = as is; default = clNone = transparent)
procedure SetTabBox(	defines a tab box for the last defined tab with the same PenWidth for all box lines
APenWidth: Double = NA;	pen width for all box sides (NA = default = as is; pwNone = 0 = no line)
ABrushColour: TColor = clNone); overload;	background shading colour to apply (NA = as is; default = clNone = transparent)
procedure SetTabBoxes(	defines tab boxes for all tabs
ALeftPenWidth,	pen width for left box side (NA = as is; pwNone = 0 = no line)
ATopPenWidth,	pen width for top box side (NA = as is; pwNone = 0 = no line)
ARightPenWidth,	pen width for right box side (NA = as is; pwNone = 0 = no line)
ABottomPenWidth: Double;	pen width for bottom box side (NA = as is; pwNone = 0 = no line)
ABrushColour: TColor = clNone); overload;	background shading colour to apply (NA = as is; default = clNone = transparent)
procedure SetTabBoxes(	defines tab boxes for all tabs with the same PenWidth for all box lines
APenWidth: Double = NA;	pen width for all box sides (NA = as is; pwNone = 0 = no line)
ABrushColour: TColor = clNone); overload;	background shading colour to apply (NA = as is; default = clNone = transparent)
property TabBoxLineColour: TColor;	colour to use for box lines (default = clBlack)
property TabBoxLineStyle: TVPEPenStyle;	pen style to use for box lines: psSolid (default), psDash, psDot, psDashDot, psDashDotDot

VPE+ Report Interface (TReportInterface) Line Tabs

Output to Line Tabs

procedure PrintTab(	prints to the current line tab, indexed 1..Count, then moves to next sequential tab
Text: string;	text to print
TextStyle: TTextStyle = tsNormal;	font style to use for text output (Bold, Underline, Italic, Strikeout), default = tsNormal else tsB, tsU, tsI, tsS, tsBU, tsBI, tsBS, tsUI, tsUS, tsIS, tsBUI, tsBUS, tsBIS, tsUIS, tsBUIS
DrawBox: Boolean = True;	True (default) to print the tab box lines (if set), else no lines
ShadeBox: Boolean = True); overload;	True (default) to shade the tab box (if set), else no background shading
procedure PrintTab(	prints to a specified line tab, but does NOT change the active tab
ATabIndex: Integer;	index of line tab to output to (1..Count)
Text: string;	text to print
TextStyle: TTextStyle = tsNormal;	font style to use for text output (Bold, Underline, Italic, Strikeout), default = tsNormal else tsB, tsU, tsI, tsS, tsBU, tsBI, tsBS, tsUI, tsUS, tsIS, tsBUI, tsBUS, tsBIS, tsUIS, tsBUIS
DrawBox: Boolean = True;	True (default) to print the tab box lines (if set), else no lines
ShadeBox: Boolean = True); overload;	True (default) to shade the tab box (if set), else no background shading
procedure PrintTabSet(	prints an array of strings to sequential line tabs (from current tab)
TextSet: array of string;	text strings to print
TextStyle: TTextStyle = tsNormal;	font style to use for text output (Bold, Underline, Italic, Strikeout), default = tsNormal else tsB, tsU, tsI, tsS, tsBU, tsBI, tsBS, tsUI, tsUS, tsIS, tsBUI, tsBUS, tsBIS, tsUIS, tsBUIS
DrawBox: Boolean = True;	True (default) to print the tab box lines (if set), else no lines
ShadeBox: Boolean = True);	True (default) to shade the tab box (if set), else no background shading
procedure SkipTab(	skips the next line tab(s)
TabCount: Integer = 1;	number of tabs to skip (default = 1)
DrawBox: Boolean = True;	True (default) to print the tab box lines (if set), else no lines
ShadeBox: Boolean = True);	True (default) to shade the tab box (if set), else no background shading
procedure DrawTabBox(	draws a tab box (no text output)
ALineTab: TLineTab;	line tab for which the box is drawn
DrawBox: Boolean = True;	True (default) to print the tab box lines (if set), else no lines
ShadeBox: Boolean = True);	True (default) to shade the tab box (if set), else no background shading
procedure DrawTabBox(	draws a tab box (no text output)
ATabIndex: Integer;	index of tab (1..Count) for which the box is drawn
DrawBox: Boolean = True;	True (default) to print the tab box lines (if set), else no lines
ShadeBox: Boolean = True);	True (default) to shade the tab box (if set), else no background shading
procedure DrawTabBoxes(	draws all tab boxes (no text output)
DrawBox: Boolean = True;	True (default) to print the tab box lines (if set), else no lines
ShadeBox: Boolean = True);	True (default) to shade the tab box (if set), else no background shading

Note: DrawTabBox & DrawTabBoxes do NOT clear tab bumps when called - use ClearTabBumps instead.

procedure FinishTabBoxes(	draws the top or bottom line of all line tab boxes (at YPos)
PenWidth: Double);	width of pen to use (eg pwNormal = 0.3 mm)

Saving, Retrieving and Clearing Line Tabs

procedure PushTabList; overload;	push active line tab list onto stack (LIFO)
procedure PushTabList(	save active line tab list to an indexed position
ATabListIndex: Integer); overload;	index position to save to (1..10)
procedure PopTabList; overload;	make last pushed line tab list active, and remove it from the stack
procedure PopTabList(	restore saved line tab list from indexed position
ATabListIndex: Integer); overload;	index position to restore from (1..10)
procedure ClearLineTabs;	clears active line tabs
procedure ClearTabListStack;	clears tab list stack (called automatically after a report)
procedure ClearSavedTabLists;	clears all saved tab lists (called automatically after a report)

VPE+ Report Interface (TReportInterface) Line Tabs

Tab Bumpers

For 1-off overrides (or "bumps") against the next tab. Bumps are cleared when the tab is printed.

procedure BumpTabBox(	bumps tab box line widths
ALeftPenWidth,	alternative left box line width (NA = no bump; pwNone = 0 = no line)
ATopPenWidth,	alternative top box line width (NA = no bump; pwNone = 0 = no line)
ARightPenWidth,	alternative right box line width (NA = no bump; pwNone = 0 = no line)
ABottomPenWidth: Double);	alternative bottom box line width (NA = no bump; pwNone = 0 = no line)
procedure BumpTabBrush(	bumps background shading colour
ABrushColour: TColor);	alternative brush colour to apply (clNone for no shading)
procedure BumpTabJustify(	bumps text justification
Justify: TPrintJustify);	alternative justification to apply (jLeft, jCentre, jRight)
procedure BumpTabMargins(	bumps tab margins
MarginLeft	alternative left tab margin (NA = no bump, negative value to widen tab)
MarginRight: Double);	alternative right tab margin (NA = no bump, negative value to widen tab)
procedure BumpTabPenColour(	bumps line pen colour
APenColour: TColor);	alternative pen colour to use
procedure BumpTabPenStyle(	bumps line pen style
APenStyle: TVPEPenStyle);	alternative pen style to use
procedure ClearTabBoxBumps;	clears tab box pen, colour and brush shading bumps
procedure ClearTabBrushBump;	clears tab brush shading bump
procedure ClearTabBumps;	clears tab bumper & cancels HoldTabBumps
procedure ClearTabJustifyBump;	clears current tab text justification bump
procedure ClearTabMarginBump;	clears current tab margin bumps
procedure ClearTabTextBumps;	clears tab justification and margin bumps
procedure FreeTabBumper;	frees tab bumper object & cancels HoldTabBumps (called automatically after a report)
procedure HoldTabBumps;	prevents automatic clearing of tab bumper (call ClearTabBumps instead)

Tab Metrics & Miscellaneous

Note: Tab Index = 1..Count

function TabStart(	XPos for tab left border
ATabIndex: Integer): Double;	index of tab
function TabCentre(	XPos for tab centre position (centred between borders)
ATabIndex: Integer): Double;	index of tab
function TabEnd(	XPos for tab right border
ATabIndex: Integer): Double;	index of tab
function TabLeftMargin(	the tabs left text margin
ATabIndex: Integer): Double;	index of tab
function TabRightMargin(	the tabs right text margin
ATabIndex: Integer): Double;	index of tab
function TabTextStart(	XPos for start of tab text (accounts for left margin)
ATabIndex: Integer): Double;	index of tab
function TabTextCentre(	XPos for tab centre position (centred between text margins)
ATabIndex: Integer): Double;	index of tab
function TabTextEnd(	XPos for end of tab text (accounts for right margin)
ATabIndex: Integer): Double;	index of tab
function TabWidth(	tab width (between left and right borders)
ATabIndex: Integer): Double;	index of tab
function TabTextWidth(	text width, accounting for left and right text margins
ATabIndex: Integer): Double;	index of tab
function GetTab(	returns tab object from tab line given by Index
ATabIndex: Integer): TLineTab;	index of tab
function GetTabList(	returns tab line object given by Index from saved lists
ATabListIndex: Integer): TLineTabList;	index of tab list
procedure ResetTabLine;	restores first tab as active tab and returns cursor to BandLeft
property ActiveTabIndex: Integer;	set or get the active line tab index (1..Count). Cursor position is not changed.
property ActiveTabList: TLineTabList;	current ("active") line tab list

Text Block (TTextBlock) Wrapping Text Blocks

TTextBlock encapsulates the VPE text block object (TVPETextBlock) and allows for controlled output of multi-line text blocks. Either drop a TTextBlock onto a form and link it to the ReportWriter generating the report (by setting its ReportWriter property), or create one on the fly using constructor Create(ReportWriter). Call the OpenBlock method (or OpenBlockFromFile or OpenBlockFromStream) to pass text to the block and render it (internally) using the currently set margins and font settings. You can also pass block margins and text margins via this method. The PrintLines and PrintHeight methods output text by line count or height constraint. As output proceeds, a "text cursor" is advanced through the text block so only remaining text is output with subsequent calls to these methods.

To manage text output with varying line widths (eg to wrap text around another object), or varying font settings etc, use the matching RenderLines or RenderHeight methods which "output" text in the same manner, but without adding it to the document. This allows you to assess vertical space requirements for the text. As changes are made to the boundaries, justification or font settings, the remaining text must be re-rendered by calling RenderBlock before reading line count or height properties etc.

Note that properties such as BlockLineCount and BlockHeight will reflect the full combined line count and height of all the segments included in a block. Call ResetBlock to return the "text cursor" to the beginning at any time, particularly after a block has been "Rendered" for height assessment and must be re-started for final output.

You may re-use the same TTextBlock for a new block of text by simply calling OpenBlock again. Call CloseBlock to release the underlying TVPETextBlock and free the memory associated with it while a VPE document is still open (this happens automatically when the document is closed). CloseBlock is called automatically if you call OpenBlock or free a TTextBlock.

constructor Create(	returns the TTextBlock object created
AOwner: TComponent);	AOwner should be a ReportWriter, although the writer can be set later via the ReportWriter prop
procedure OpenBlock(	
ATextBlock: string;	text with which to initialise the block
ABlockLeft: Double = NA;	optional left print boundary (otherwise set per BlockLeft)
ABlockRight: Double = NA;	optional right print boundary (otherwise set per BlockRight)
ATextLeftMargin: Double = NA;	optional left text margin (otherwise set per TextLeftMargin)
ATextRightMargin: Double = NA);	optional right text margin (otherwise set per TextRightMargin)
procedure OpenBlockFromFile(	
AFileName: string;	source file for text with which to initialise the block
ABlockLeft: Double = NA;	optional left print boundary (otherwise set per BlockLeft)
ABlockRight: Double = NA;	optional right print boundary (otherwise set per BlockRight)
ATextLeftMargin: Double = NA;	optional left text margin (otherwise set per TextLeftMargin)
ATextRightMargin: Double = NA);	optional right text margin (otherwise set per TextRightMargin)
procedure OpenBlockFromStream(	
AStream: TStream;	source stream for text with which to initialise the block
ABlockLeft: Double = NA;	optional left print boundary (otherwise set per BlockLeft)
ABlockRight: Double = NA;	optional right print boundary (otherwise set per BlockRight)
ATextLeftMargin: Double = NA;	optional left text margin (otherwise set per TextLeftMargin)
ATextRightMargin: Double = NA);	optional right text margin (otherwise set per TextRightMargin)
procedure CloseBlock;	frees memory associated with the underlying VPE TVPETextBlock.
procedure RenderBlock;	re-renders remaining text with current width and font settings
procedure ResetBlock;	re-winds the block to the beginning of the text (as after OpenBlock call)

Note that calls to TTextBlock output functions are only valid during report execution (between OnGenerateStart and OnGenerateEnd).

Text Block Definition:

Set these properties to define the block (and call RenderBlock to re-render after any changes):

property BlockLeft: Double;	defines left block boundary
property BlockRight: Double;	defines right block boundary
property ReportWriter: TReportWriter;	the TReportWriter generating the report (passed in the constructor, or set in the component at design time)
property TextLeftMargin: Double;	defines left margin between BlockLeft and text
property TextRightMargin: Double;	defines right margin between BlockRight and text (default left and right text margins = 0 mm) jCentre justified text is centred between the margins, so, as a rule, they should be even for centred text.
property Justify: TJustify;	set text justification (jLeft, jRight, jCentre, jBlock, jBlockFull)
property TextStyle: TTextStyle;	font style to use for text output (Bold, Underline, Italic, Strikeout), default = tsNormal else tsB, tsU, tsI, tsS, tsBU, tsBI, tsBS, tsUI, tsUS, tsIS, tsBUI, tsBUS, tsBIS, tsUIS, tsBUIS

Text Block (TTextBlock) Wrapping Text Blocks

Text Block Output:

procedure PrintHeight(	outputs block text to a vertical height constraint
HeightLimit: Double;	print height constraint; stops with respect to HeightMode
HeightMode: THeightMode = hmStopBefore;	
	hmStopBefore = last line must fit WITHIN HeightLimit (default)
	hmStopBeyond = accepts line that partially fits as well (useful when wrapping around objects)
EndOnNewLine: Boolean = True);	True (default) to finish block with NewLine (else cursor finishes at BlockRight)
procedure PrintLines(	outputs block text to a line count constraint
LineCountLimit: Integer = 0;	line count constraint (default = 0 = all lines)
EndOnNewLine: Boolean = True);	True (default) to finish block with NewLine (else cursor finishes at BlockRight)

Pre-Rendering TextBlocks:

In order to assess the overall line count or height of a multi-segmented TextBlock, it is necessary to pre-render (without actual output) each segment. Use procedure RenderHeight or RenderLines to achieve this. As you change block width or font settings etc, it is still necessary to call RenderBlock to re-render the remaining text before calling either of these procedures.

BlockLineCount and BlockHeight will reflect the combined line count and height of all rendered block segments.

To subsequently output the text block, call ResetBlock to re-initialise the block text, and use procedure PrintLines or PrintHeight as appropriate.

procedure RenderHeight(	renders block text to a vertical height constraint (no actual output)
HeightLimit: Double;	print height constraint; stops with respect to HeightMode
HeightMode: THeightMode = hmStopBefore;	
	hmStopBefore = last line must fit WITHIN HeightLimit (default)
	hmStopBeyond = accepts last partial line as well (useful when wrapping around objects)
EndOnNewLine: Boolean = True);	True (default) to finish block with NewLine (else cursor finishes at BlockRight)
procedure RenderLines(	renders block text to a line count constraint (no actual output)
LineCountLimit: Integer = 0);	number of lines to render (default = 0, for all lines)

Other TTextBlock Properties and Methods:

function BlockHeight: Double;	full height of block (all rendered segments)
function BlockHeightLeft: Double;	height of remaining (unprinted) lines
function BlockLineCount: Integer;	total number of lines rendered (including any already output lines)
function BlockLineCountLeft: Integer;	number of lines remaining for output
function BlockRendered: Boolean;	True if block has been rendered (else call to OpenBlock or RenderBlock needed)
function BlockTextWidth: Double;	returns width of text (between left and right text margins, + "eyeball tolerance" of 0.15 mm)
function BlockWidth: Double;	returns block width (between BlockLeft and BlockRight)
function EnoughTextWidth(	returns True if block will accommodate a given width of text
ATextWidth: Double = 0): Boolean;	the text width to accommodate (default = 0 for minimum allowed width)
	NB the minimum allowed text width is arbitrarily set to 10mm
function IsEmpty: Boolean;	true if there is no rendered lines in the block
function IsFinished: Boolean;	true if there is nothing left to output
function TextLeft: Double;	left text margin
function TextRight: Double;	right text margin
property CurrentLine: Integer;	index of the current line (to be output next) in the block (1st line = 1)

Synchronising Text Blocks to Tab Settings

To output text to match line tab settings, text block borders and margins can be synchronised by calling:

procedure SynchToLineTab(	sets text block borders and margins to match a line tab, and calls RenderBlock
ATabIndex: Integer); overload;	index of line tab to synchronise with
procedure SynchToLineTab(	sets text block borders and margins to match line tabs, and calls RenderBlock
FromLineTabIndex,	index of line tab to synchronise left block borders and margins with
ToLineTabIndex: Integer); overload;	index of line tab to synchronise right block borders and margins with

RTF Block (TRTFBlock) Wrapping RTF Blocks

TRTFBlock encapsulates the RTF version of the VPE TVPETextBlock and allows for controlled output of multi-line RTF blocks. Either drop a TRTFBlock onto a form and link it to the ReportWriter generating the report (by setting its ReportWriter property), or create one on the fly using TRTFBlock.Create(ReportWriter).

Unlike the plain text block component, TRTFBlock does not have Justify or TextStyle properties as these features are encoded in the RTF instead. Also, because varying font sizes may be involved, line heights may vary. To return the height of a given line, use the function RTFLineHeight.

Otherwise, in general, an RTFBlock behaves in the same manner as a TextBlock, and the same methods and properties apply.

function RTFLineHeight(	returns the height of an RTF line
LineNumber: Integer): Double;	number of the line to return height for (1..n) NB The line number refers to unrendered or unprinted lines only, so line #1 is always the next line to be rendered or printed. Already rendered or printed lines are ignored.

RTF On-the-Fly

An RTFBlock can also be used to output "RTF on-the-fly". This involves enriching plain text with RTF coding at the time of printing to enhance output. A syntactically complete RTF document as such is not required. Conventional header details with font and colour tables are not required. You could, for example, do no more than place bold tokens around a given word and achieve the expected result.

VPE uses default (internal) font and colour tables which circumvents their need in an RTF string, and makes compiling RTF easy. You can, of course, readily alter or replace these tables as required. Refer to the VPE documentation for full details of supported coding and usage, and the methods available to manipulate RTF.

Several additional VPE+ utility methods are provided to assist with compilation of an RTF string by applying specific coding for you. These methods are entirely optional, simply providing a shortcut to some common RTF syntax. You can encode your own RTF, and exploit more RTF features than they actually cover.

NOTE: RTF strings with a leading "{" encoding character which is NOT the standard "{\rtf" control code grouping may be misinterpreted by VPE. In these cases, use the RTFEnclose function to enclose the string within a "{\rtf ...}" grouping.

function RTFCodeGroup(	returns a string enclosed (grouped) by braces, eg "{AText}"
AText: string): string;	the text to be enclosed in braces
function RTFEnclose(	returns a string enclosed within a RTF header control grouping, eg "{\rtf AText}"
AText: string): string;	the text to be enclosed within the RTF header grouping
function RTFLine(	returns line feed codes, eg "\line "
ACount: Integer = 1): string;	the number of line feed codes to return (default = 1)
function RTFLiteral(	returns a string with syntax characters qualified as literals
AText: string): string;	the text in which to add a preceding "\" for literal syntax characters
function RTFParagraph(	returns paragraph end codes, eg "\par "
ACount: Integer = 1): string;	the number of paragraph end codes to return (default = 1)
function RTFStyle(	codes a string with a font style and size
AText: string;	the text to encode
ATextStyle: TTextStyle;	the style to apply (tsNormal for plain text) else tsB, tsU, tsI, tsS, tsBU, tsBI, tsBS, tsUI, tsUS, tsIS, tsBUI, tsBUS, tsBIS, tsUIS, tsBUIS
ASize: Integer = NA): string;	font point size to apply, (default = NA = no size change)
function RTFStyle(	codes a string with a font size
AText: string;	the text to encode
ASize: Integer): string;	font point size to apply
function RTFTab(	returns tab codes, eg "\tab "
ACount: Integer = 1): string;	the number of tab codes to return (default = 1)
function RTFTabBullet(	returns leading tabs, a bullet, and trailing spaces, eg "\tab \bullet "
ATabCount: Integer = 0;	the number of tab codes to add before the bullet (default = 0)
SpacesAfter: Integer = 2): string;	the number of spaces to add after the bullet (default = 2)
procedure RTFClearTab(	clears a tab position
X: Double);	the tab position to clear
procedure RTFClearTabs;	clears all tab positions
procedure RTFSetTab(	sets a tab position
X: Double);	the position at which to set the tab

VPE+ ReportWriter (TReportWriter) General

The Report Writer (TReportWriter)

A descendant of TVPEngine providing access to the underlying Virtual Print Engine and additional functionality. Typically, a TReportWriter component is placed locally for each report, although multiple reports can be generated through the same TReportWriter just as easily.

Custom report parameters may be presented in a group box assigned to ReportOptionGroupBox.

Custom report header and footer titles may be presented in a group box assigned to ReportTitleGroupBox.

Report title and subtitle strings describing the report as a whole, page header title and subtitle strings, and a page footer title string can be defined for each report and optionally exposed in the default setup form. Additionally, a footer "stamp" string (eg for a company name) with optional date string appended is available. These report elements can be output in ReportInterface events OnSystemPageHeader and OnSystemPageFooter, for example, but their use is entirely optional. Properties controlling these features are listed below.

The following properties and procedures extend the functionality of TVPEngine:

property CustomFormatCount: Integer;	the number of custom formats defined (default = 0, no custom formats)
property CustomFormatDefaultIndex: Integer;	indicates which custom format option is selected by default (1..CustomFormatCount, if any).
function CustomFormatExt(	returns the file extension to be used for a custom format
AFormatIndex: Integer): string;	index of custom format
property DefaultFonts	Defines a default font in 5 sizes for general use (optionally applied as saved fonts, 1..5). if not defined (ie no font name or 0 size), defers to ReportInterface.DefaultFonts instead.
FontName: string;	name of font to use (default = none)
Size1 to Size5: Integer;	font sizes to use: defaults = Size1 (0), Size2 (0), Size3 (0), Size4 (0), Size5 (0) call ***procedure SetDefaultFonts;*** to save as fonts 1..5 (eg in ReportWriter.OnConfigure)
property EnablePageHeaderTitle: Boolean;	if False, disables the displayed page header title in setup (default True)
property EnablePageHeaderSubTitle: Boolean;	if False, disables the displayed page header subtitle in setup (default True)
property EnablePageFooterTitle: Boolean;	if False, disables the displayed page footer title in setup (default True)
property FixedBandEnabled: TFixedBandEnabled;	Fixed band enable states to be applied on report execution. If bsDefault, defers to ReportInterface.DefaultFixedBandEnabled state. Otherwise, overrides ReportInterface.DefaultFixedBandEnabled state.
LetterfootEnabled: TFixedBandEnabled;	default for UseLetterfoot state (default bsDefault)
LetterheadEnabled: TFixedBandEnabled;	default for UseLetterhead state (default bsDefault)
PageFooterEnabled: TFixedBandEnabled;	default for UsePageFooter state (default bsDefault)
PageHeaderEnabled: TFixedBandEnabled	default for UsePageHeader state (default bsDefault) NOTE: Remittance band is disabled by default - enable it using EnableRemittance. NOTE: FixedBand "Use" states can be overridden in ReportWriter.OnConfigure.
TBandState	band states
bsDefault	band state defers to the ReportInterface default
bsDisabled	band is disabled
bsEnabled	band is enabled
property OutputDefaultFileName: string;	specifies a default output filename (default = "Report")
property OutputDefaultFormat: TFileOutputFormat;	specifies which output format (see list below) is selected by default (default ofPDFFile)
property OutputFormats: TFileOutputFormatSet;	the set of output formats to be made available (default [ofPDFFile, ofCustomFile])
TFileOutputFormat	output actions that can be selected from setup
ofPDFFile	PDF file format requested
ofODTFile	ODT file format requested
ofHTMLFile	HTML file format requested
ofXMLFile	XML file format requested
ofVPEFile	VPE native file format requested
ofCustomFile	custom file format requested (identified by CustomFormatIndex)

VPE+ Application Reporting Interface

VPE+ ReportWriter (TReportWriter) General

property Options: TReportOptionSet;	options controlling the behaviour of the report process (applied when report initialised)
TReportOption	report options; option set of type TReportOptionSet
roCanDuplex: Boolean;	allow duplex options if supported (default = True)
roCanEmail: Boolean;	allows emailing options for filed reports (default = True)
roCanFile: Boolean;	allow filing of VPE reports to any available format (default = True)
roCanPreview: Boolean;	allow preview of VPE reports (default = True)
roCanPreviewLoad: Boolean;	allow user to open further report files in preview (default = True)
roCanPrint: Boolean;	allow printing of VPE reports (default = True)
roConfirmOverwrite: Boolean;	show confirm dialogue for report file overwrites (default = True)
roConfirmFiled: Boolean;	show confirm dialogue when report filed (default = True)
roConfirmPrinted: Boolean;	show confirm dialogue when report printed (default = True)
roConfirmBatchOutput: Boolean;	show confirm dialogue when batch has been printed or filed
roPrintAbortDialogue: Boolean;	show the VPE print abort/progress dialogue while generating reporting (default = True)
roOutputFileNamePrompt: Boolean;	prompt for report filename (else accept pre-defined) (default = True)

Once initialised with the defaults, you can set and reference report options with the following (eg in ReportWriter.OnConfigure):

procedure OptionAllow(	to allow a single report option
AReportOption: TReportOption);	report option to allow
procedure OptionDisallow(	to disallow a single report option
AReportOption: TReportOption);	report option to disallow
procedure OptionsAllow(	allows a set of report options
AReportOptionSet: TReportOptionSet);	set of report options to allow
procedure OptionsDisallow(	disallows a set of report options
AReportOptionSet: TReportOptionSet);	set of report options to disallow
property OptionAllowed[	set or get whether a report option is allowed
Index: TReportOption]: Boolean;	index of report option
property OptionDisallowed[	set or get whether a report option is disallowed
Index: TReportOption]: Boolean;	index of report option
property OutputFileName: string;	Name of report file to output. OutputFileName is cleared when a report is initialised. Set a value in ReportWriter.OnConfigure, or adjust in ReportWriter.OnReportFileName. (specify BEFORE ExecuteReport called to override the ReportInterface default for setup.) if no OutputFileName is specified (default), the VPE base file is assumed to be temporary.
property PageFooterStamp: string;	optional "stamp" for use in the page footer - eg a company name (default none) NB if none, defaults to TReportInterface.DefaultPageFooterStamp
property PageFooterStampDated: Boolean;	if False, the page footer stamp is not dated (default True).
property PageFooterStamped: Boolean;	if False, the page footer stamp is not included in the page footer (default True).
property PageFooterTitle: string;	a page footer title string
property PageHeaderSubTitle: string;	a page header subtitle string
property PageHeaderTitle: string;	a page header title string
property PaperLayout: TPaperLayout;	sets the page orientation (see section on "Paper Layout/Orientation")
TPaperLayout	paper orientation options
plPortrait	portrait orientation (default, sets VPE.PageOrientation = VORIENT_PORTRAIT).
plLandscape	landscape orientation (sets VPE.PageOrientation = VORIENT_LANDSCAPE).
property PreviewWindow: TPreviewWindow;	defines various defaults for the report preview window
Height: Integer	height to set preview form (default = 600; 0 if PreviewWindow.PercentScreenHeight specified)
PercentScreenHeight: Integer	height to set preview form as percentage of screen height (0 if PreviewWindow.Height specified
PercentScreenWidth: Integer	width to set preview form as percentage of screen width (0 if PreviewWindow.Width specified)
ScaleMode: TPreviewScaleMode	scale mode to apply when preview opens; psmFullPage or psmPageWidth (default)
Width: Integer	width to set preview form (default = 700; 0 if PreviewWindow.PercentScreenWidth specified)
WindowState: TWindowState	window state to apply when preview opens; wsMaximized (default), wsMinimized, or wsNormal
procedure QueryCustomFormat(	allows custom format details to be queried per OnCustomFormatQuery event.
AFormatIndex: Integer;	index of custom format to be queried
out AFormatName,	returns the custom format name
AFormatExt: string);	returns the custom format file extension
property ReportDescription: Boolean;	report description used in setup and as the print spool job description. If blank, ReportInterface.DefaultReportDescription is used. Value is passed to ReportWriter.OnDescribeReport event for dynamic description changes.
property ReportFileNameConfirmed: Boolean;	where a report filename has already been confirmed in a save dialogue, say, set True to prevent any subsequent prompts to confirm the filename.
property ReportOptionGroupBox: TGroupBox;	assign a GroupBox containing custom report parameter edit controls etc to be displayed full-width at the bottom of the setup form > height will be accommodated as required. > width will be increased to 421 minimum, but greater is accommodated.

VPE+ Application Reporting Interface

VPE+ ReportWriter (TReportWriter) General

property ReportSubTitle: string;	subtitle to describe a report (eg used in the default report setup form)
property ReportTag: Integer;	an optional identification tag assigned to a report can be set in ReportWriter.OnDescribeReport
property ReportTitle: string;	title to describe a report (eg used in the default report setup form) (could also be used to set ReportWriter.ReportDescription for the print spool)
eg ReportTitle =	"Customer Credit Summary"
ReportSubTitle =	"for Overdue Accounts"
property ReportTitleGroupBox: TGroupBox;	assign a GroupBox containing custom header/footer edit controls etc to be displayed in the setup form in place of the default Title box > height will be increased to 110 minimum, but greater is accommodated. > width will be increased to 335 minimum, but greater is accommodated.
procedure RetainVPESourceFile(VPESourceFileName: string);	call (eg in ReportWriter.OnConfigure) to force the base VPE source file to be retained. a filename for the VPE source file (rather than a temporary file) (this file is saved in addition to any output pdf or html file etc)
property SelectedCustomFormatIndex: Integer;	marks the selected custom format index, 1..CustomFormatCount default = 0, none selected. See following "Custom Formats" section.)
procedure SendFileByEmail(AFileName: string);	send a file(s) by email per OnSendEmail event handler name of file to be sent
procedure SendSelectedFilesByEmail(ADefaultFolder ADefaultExt AFilter: string);	send a selection of files by email per OnSendEmail event handler applied as InitialDir in the TOpenDialog applied as DefaultExt in the TOpenDialog applied as Filter in the TOpenDialog
property TitleSetup: string;	setup form caption (if not specified, returns ReportInterface.DefaultTitleSetup)
property TitleStatus: string;	status form caption (if not specified, returns ReportInterface.DefaultTitleStatus)
property UsePageFooterTitle: Boolean;	if False, the page footer title is hidden and not output (default True)
property UsePageHeaderSubTitle: Boolean;	if False, the page header subtitle is hidden and not output (default True)
property UsePageHeaderTitle: Boolean;	if False, the page header title is hidden and not output (default True)
property UsePageTitles: Boolean;	if False, hides the header/footer title GroupBox in setup (default True) (includes the System title group box or an assigned ReportTitleGroupBox)
function ValidateSetup(ASetupAction: TBatchOutputAction): Boole	allows ReportWriter.OnSetupValidate event to be called manually intended setup action
TBatchOutputAction	output actions that can be selected from setup
baPreview	intended output action is previewing report
baPrint	intended output action is printing report
baFile	intended output action is filing report
baEmail	intended output action is emailing filed report

VPE+ ReportWriter (TReportWriter) Event Handlers

Report Generation Events NB Reports in a multi-report batch can be identified per ReportInterface.BatchIndex.

property OnConfigure;	Allows the ReportInterface or ReportWriter components to be configured as required. NB The VPE doc is NOT open at this point.
property OnGenerate;	Main point of execution for VPE report code.
property OnGenerateStart;	Fires just before the report generation process starts.
property OnGenerateEnd;	Fires just after the report generation process ends.
property OnPageStart;	Fires just after a new page is started (for first page, and when NewPage is called).
property OnPageEnd;	Fires just before a page is finished (when NewPage is called and for last page).
property OnReportFileName(	allows a report filename to be assigned, default = OutputFileName
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
ReportWriter: TReportWriter;	ReportWriter generating the report
OutputFormat: TFileOutputFormat;	format of report file
AFormatIndex,	if OutputFormat = ofCustomFile, the index of the custom format
AReportTag: Integer;	the report tag assigned to the report (default = 0)
var APath,	the report file path name
AName,	the report file name
AExt: string);	the report file name extension
property OnFileOutput(	Allows an output file to be saved to a database, for example.
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
ReportWriter: TReportWriter;	ReportWriter generating the report
AFileName: string;	name of output file.
var IsTemporary: Boolean);	Output file will be deleted if True or set True.
property OnStreamBefore(	Allows streaming of an output file via OnStreamOutput to be prevented.
OutputFormat: TOutputFormat;	format of output file to be streamed, which occurs only if Assigned(OnStreamOutput).
var AllowStream: Boolean);	if False, prevents OnStreamOutput streaming the output file (default = True).
property OnStreamOutput(	Creates and exposes a report memory stream (can be prevented in OnStreamBefore).
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
ReportWriter: TReportWriter;	ReportWriter generating the report
OutputFormat: TOutputFormat;	format of output file being streamed.
ReportStream: TVPEStream;	the report stream (created by VPE CreateMemoryStream).
ReportTag: Integer);	the user defined option tag assigned to the report.
var IsTemporary: Boolean);	Output file will be deleted if True or set True.
property OnGenerateException(	Allows processing of an exception raised while generating a report.
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
ReportWriter: TReportWriter;	ReportWriter generating the report
ReRaise: Boolean)	set True to re-raise the exception with a call to CheckAborted (default = False). if not Assigned(OnGenerateException), CheckAborted is called by default. (see error management details below)

VPE+ Application Reporting Interface

VPE+ ReportWriter (TReportWriter) Event Handlers

ReportWriter Setup Events — Use the default setup form, or provide an alternative per OverrideSetup event.

property OnPreviewConfigure(— Allows the preview panel to be configured before showing a report
- ReportInterface: TReportInterface; — ReportInterface component managing the reporting process
- ReportWriter: TReportWriter; — ReportWriter generating the report
- PreviewConfigureState: TPreviewConfigureState); — indicates the action performed in opening the document

OnPreviewConfigure always fires AFTER a document is OPENED, but BEFORE previewing it.
Set preview window layout properties in ReportWriter.PreviewWindow

TPreviewConfigureState — output actions that can be selected from setup
- pcsFirstReport — about to preview the first report (preview form is showing for the first time)
- pcsSwitchReport — about to preview another (batched) report (preview may already be configured OK)
- pcsUserAdjustment — report is being re-opened to accommodate a user adjustment (eg toggling grid lines or rulers)

property OnSetupBefore(— Fires just before the setup form is shown.
- ReportInterface: TReportInterface; — ReportInterface component managing the reporting process
- ReportWriter: TReportWriter);

property OnSetupValidate(— Allows validation of setup details and return to setup if rejected.
- ReportInterface: TReportInterface; — ReportInterface component managing the reporting process
- ReportWriter: TReportWriter; — ReportWriter generating the report
- SetupAction: TBatchOutputAction; — The intended action to take following setup: baPreview, baPrint, baFile, baEmail
- var Accept: Boolean); — if False, rejects details and re-shows the setup form (default = True).

property OnSetupAfter(— Allows setup to be rejected (after it has been accepted by the user).
- ReportInterface: TReportInterface; — ReportInterface component managing the reporting process
- ReportWriter: TReportWriter; — ReportWriter generating the report
- var Accept: Boolean; — if False, cancels report generation altogether (default = True).
- var ReasonSetupRejected: string); — optionally provide a reason for setup rejection (to be included in AbortReasonMessage)

NB Read the setup state from properties SelectedSetupAction, SelectedSetupMode, SelectedFormat and SelectedFormatIndex.

property OverrideSetup(— Allows an alternative setup form to be used during the report process.
- ReportInterface: TReportInterface; — ReportInterface component managing the reporting process
- ReportWriter: TReportWriter; — ReportWriter generating the report
- OverrideState: TOverrideFormState; — indicates form state to implement: ofsFree, ofsCreate, ofsShow, ofsHide
- var OverrideForm: TForm; — the override form instance as created and returned when OverrideState = ofsCreate.
- OptionTag: Integer); — an optional tag to be used for setup options

Email Event — Allows report(s) to be sent by email.

procedure OnSendEmail(— Fires in response to request to email a report(s).
- ReportInterface: TReportInterface; — ReportInterface component managing the reporting process
- ReportWriter: TReportWriter; — ReportWriter generating the report
- AttachSummary: String; — summary string describing reports attached
- AttachList: TStrings); — list of reports to be included as attachments (giving full file name)

VPE+ ReportWriter (TReportWriter) Custom Formats

In addition to the "system" output file formats handled by VPE, "custom" file formats may be added as extra output options available in the file format selection list. Custom format name and file extension details must be provided through the event handler OnCustomFormatQuery, and a suitable file generator through OnCustomFormatGenerate.

To output directly to a custom format (ie without selecting a format via the setup form GUI), set SelectedCustomFormatIndex to the relevant value (1..CustomFormatCount).

property SelectedCustomFormatIndex: Integer;	indicates which custom format option has been selected. either set from users selection in a setup form, or set in code prior to custom file output (SetupMode = smCustomFile, smEmailCustomFile) else custom format indexes 1..CustomFormatCount.
property CustomFormatCount: Integer;	specifies the number of custom file formats defined (indexed 1..CustomFormatCount).
property CustomFormatDefaultIndex: Integer;	indicates which custom format option is selected by default (1..CustomFormatCount, if any).
function CustomFormatExt(AFormatIndex: Integer): string;	returns the file extension to be used for a custom format index of custom format
procedure QueryCustomFormat(AFormatIndex: Integer; out AFormatName, AFormatExt: string);	allows custom format details to be queried per OnCustomFormatQuery event. index of custom format to be queried returns the custom format name returns the custom format file extension
procedure OnCustomFormatQuery(FormatIndex: Integer; out FormatName: string; out FormatExt: string);	Event: for each custom format, a format name and file extension will be queried index of the custom format (1..CustomFormatCount) to be described. descriptive name for the custom format (eg 'CSV Text File'). filename extension to use for this format (eg 'csv'). NB called as required to populate format or filter lists etc.
procedure OnCustomFormatGenerate(ReportInterface: TReportInterface; FormatIndex: Integer; var OptionTag: Integer; var FileName, Description: string; var GeneratedOK: Boolean);	Event: for each custom format, a file generator must be defined. ReportInterface component managing the reporting process index of the custom format (1..CustomFormatCount) to be generated. optionally tag to assign to report (default = ReportWriter.TagReport). filename and report description provided from setup - may be changed as required. set True once the file has been generated OK (default = False). if left False, or if an exception is raised, the error will be reported automatically. additional error detail may be added by setting AbortReason.

Report Frames (TxxxFrame) Frames & Bands

Report frames introduce a banded structure about which to (optionally) design a report.

The various bands are described in the "Page Frame Metrics" diagramme shown later. Essentially, there are fixed-height bands at the top of the page (Letterhead and PageHeader) and fixed-height bands at the bottom of the page (PageFooter, Letterfoot, Remittance), with dynamic-height bands filling the space between.

Report code is executed in event handlers associated with each band. The parent frame controls the sequence in which these band events are fired, and how they are cycled and terminated. Each band may have a default font and line tab set assigned via their FontIndex and TabIndex properties. These indexes reference the fonts and line tab sets saved for the report in the ReportInterface.

In the absence of any frame components, ReportInterface effectively maintains a single band representing the entire printable area on a page. Frames are useful, but entirely optional. You can still create a report without frames.

TPageFrame is the main frame component. It encapsulates a full page structure, and would be the base frame typically used for most reports. Its DetailFrame property allows an additional TMasterFrame or TDetailFrame frame to be asigned and automatically executed as a "nested frame" for master/detail reports.
Any TMasterFrame frame can, in turn, have a further TMasterFrame or TDetailFrame nested via its own DetailFrame property. Master or detail frames can also be executed directly at any time, meaning there is no limit to the number of sub-frames that can be used in a report.

Note that each frame has a single inherent "loop" in its structure which continues cycling until its respective "Valid" property is set to False in code. These loops can, for example, be used to cycle through a dataset, but this is optional since you can also cycle the dataset yourself using a while/next loop if that is more convenient. If your data is subdivided with header/footer structure, or you wish to automatically trip a new page based on remaining line count or remaining height, then a frame component will probably be the option of choice. If there is no further structure of this sort, then a simple local loop with your own check for remaining space will suffice.

TLabelFrame is a special frame designed to cycle and generate labels in multiple columns and rows.

The structure and loop control of each of these frames is diagrammed on following pages. Properties and functions related to page metrics are summarised below.

Page Boundaries & Margins

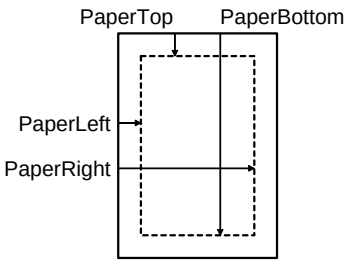
Page margins (distance from the respective page edge) demarcate the printable area.

property PaperMarginLeft: Double;	left paper margin (relative to left paper edge)
property PaperMarginTop: Double;	top paper margin (relative to top paper edge)
property PaperMarginRight: Double;	right paper margin (relative to right paper edge)
property PaperMarginBottom: Double;	bottom paper margin (relative to bottom paper edge)

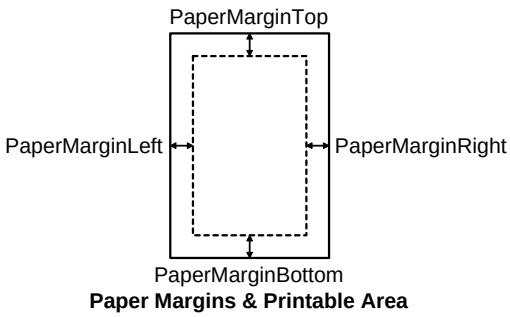
Paper margins can be set collectively using SetPaperMargins or BumpPaperMargins. If a band has already been setup, call its' ResetBandBoundaries method to re-adjust the band accordingly, or call its' ResetBand method to also reset band fonts and tabs. These methods are described below in the section "Adjusting Bands On the Fly".

Page boundaries (in X, Y axis distances) are determined by the respective paper margins and paper size.

function PaperLeft: Double;	left side of printable paper area (relative to left paper edge)
function PaperTop: Double;	top of printable paper area (relative to top paper edge)
function PaperRight: Double;	right side of printable paper area (relative to left paper edge)
function PaperBottom: Double;	bottom of printable paper area (relative to top paper edge)



Paper Boundaries & Printable Area



Paper Margins & Printable Area

Report Frames (TxxxFrame) Frames & Bands

The space on a "page" or piece of paper is indicated by:

function PaperHeight: Double;	height of the paper used (full printable and non-printable extent)
function PaperWidth: Double;	width of the paper used (full printable and non-printable extent)
function PaperPrintHeight: Double;	available printable paper height within the paper margins
function PaperPrintWidth: Double;	available printable paper width within the paper margins
function PaperPrintHeightLeft: Double;	remaining printable paper height between current YPos and PaperBottom
function PaperPrintWidthLeft: Double;	remaining printable paper width between current XPos and PaperRight
function EnoughPaperPrintHeight(HeightNeeded: Double): Boolean;	True if remaining printable paper height accommodates the height needed print height needed
function EnoughPaperPrintWidth(WidthNeeded: Double): Boolean;	True if remaining printable paper width accommodates the width needed print width needed

NOTE:
The various "PaperPrintXXX" and "EnoughPaperPrintXXX" functions refer to the printable area of the *full page.*
Use the corresponding "BandXXX" and "EnoughBandXXX" functions to reference the current band instead.

Band Boundaries & Margins

Band boundaries (in X, Y axis distances) are set dynamically within the page constraints. Reset them if required.

property BandLeft: Double;	left side of band (relative to left paper edge)
property BandTop: Double;	top of band (relative to top paper edge)
property BandRight: Double;	right side of band (relative to left paper edge)
property BandTop: Double;	bottom of band (relative to top paper edge)

Assessing Band Space:

function BandHeight: Double;	returns height of current band (ie height available to band)
function BandHeight(BandType: TBandType): Double;	returns height of a band type of band to return height for
function BandHeightLeft: Double;	returns remaining band height between current YPos and BandBottom
function BandLinesLeft: Integer;	returns number of remaining lines using current font
function BandWidth: Double;	returns width of current band (ie width available to band)
function BandWidth(BandType: TBandType): Double;	returns width of a band type of band to return width for
function BandWidthLeft: Double;	returns remaining band width between current XPos and BandRight
function EnoughBandHeight(HeightNeeded: Double): Boolean;	returns True if enough space to fit a given output height height to fit
function EnoughBandLines(LinesNeeded: Integer): Boolean;	returns True if enough space to fit a given number of output lines number of lines to fit using current font
function EnoughBandWidth(WidthNeeded: Double): Boolean;	returns True if enough space to fit a given output width width to fit

Miscellaneous

function PaperCentreXPos: Double;	centre XPos of papers printable area width
function PageCentreYPos: Double;	centre YPos of papers printable area height
function BandCentreXPos: Double;	centre XPos of bands width
function BandCentreYPos: Double;	

Report Frames (TxxxFrame) Frames & Bands

Using a Remittance Band (for PageFrame only)

A remittance band is an optional fixed band which must be manually engaged by calling EnableRemittance, typically in OnBodyFooter. It is placed at the bottom of a page below the PageFooter/Letterfoot bands, and immediately above the paper bottom margin. It may be used for the tear-off remittance advice section of an invoice, for example, or for any similar purpose.

EnableRemittance does NOT confirm there is sufficient space to accommodate the remittance band, but simply allocates the (fixed) band in the correct place. Check EnoughSpaceForRemittance and start a new page if it returns False. Remember to call ResetBand to reset band metrics on the new page if you call NewPage.

function EnableRemittance(	returns True if a remittance band is successfully enabled (band height must be over 0)
ARemittanceHeight: Double = NA): Boolea	height to reserve - if NA (default), PageFrame.BandRemittance.Height is used.
function EnoughSpaceForRemittance(	returns True if the remittance band can be accommodated on the current page
ARemittanceHeight: Double = NA): Boolea	height required - if NA (default), PageFrame.BandRemittance.Height is used.
procedure DisableRemittance;	disables remittance band on current page
property RemittanceHeight: Double;	sets or gets the remittance band height
property UseRemittance;	directly enables or disables the remittance band on the current page
property UseRemittanceHeight: Double;	directly sets the height of the remittance band

Adjusting Bands "On the Fly"

Typically, report frames will process the various bands according to frame structure and assigned properties, setting bands up automatically and triggering them as appropriate. However, it is possible to manipulate and trigger bands more "manually" when the need arises.

This is done, for example, in printing the "Page Metrics" diagramme shown following this section. In this case, the TPageFrame band structure is used to generate the diagramme, but is placed entirely within the "report body area" of the greater report (which has its own header/footer details). This involves four key manipulations:

1. Changing the paper margins so the printable area fits entirely within the "report body area" (using "BumpPaperMargins").

2. Setting up band boundaries on demand (using "SetBandBounds").

3. Differentially adjusting the "Use" height of a fixed band for a given page (see "Use" properties below). Specifically, the PageFooterHeight " is increased to 1 cm to accommodate the extra diagramme details included.

4. Restoring paper and band settings so that the page footer again matches all other pages (ie reversing the changes made above with ResetPageFooterHeight and ClearPaperMarginBumps etc).

To adjust the boundaries for the page or for a band, use:

procedure BumpPaperMargins(	applies temporary "bumps" to the paper margins (NA = no bump)
ALeft, ATop, ARight, ABottom: Double);	paper margin bumps to apply (NA = no bump)
procedure ClearPaperMarginBumps;	removes any paper margin bumps applied using BumpPaperMargins
procedure SetPaperMargins(	sets the paper margins
ALeft, ATop, ARight, ABottom: Double);	paper margins to apply (NA = no change)
procedure SetBandBounds(	sets the band bounds to utilise all available band space
BandType: TBandType;	type of band to set bounds for btFullPage, btRemainingPage, btFullBody btLetterhead, btPageHeader, btBodyTitle, btBody btPageFooter, btLetterfoot, btRemittance
RetainTopBoundary: Boolean = False);	True to retain original top position of a dynamic band rather than use the current YPos.

The height used for fixed page bands is calculated when a PageFrame first executes (and so can be subsequently overridden). It is first defaulted to the property defined under ReportInterface.DefaultFixedBandHeights:

eg ReportInterface.DefaultFixedBandHeights.PageFooterHeight

This value is overridden by the Height property (if set) of the respective band as defined in the PageFrame:

eg PageFrame.BandPageFooter.Height

You may override this value in code by setting the "Use" properties listed below. Either use these properties BEFORE the respective band is setup, or use ResetBand or ResetBandBoundaries to force a setup. Changes to these properties remain in effect for all subsequent pages in the document. To restore a fixed bands default height, call the respective height "Reset" procedure.

Report Frames (TxxxFrame) Frames & Bands

property UseLetterheadHeight: Double;
property UseLetterfootHeight: Double;
property UsePageHeaderHeight: Double;
property UsePageFooterHeight: Double;
property UseRemittanceHeight: Double;
procedure ResetLetterheadHeight;
procedure ResetLetterfootHeight;
procedure ResetPageHeaderHeight;
procedure ResetPageFooterHeight;
procedure ResetRemittanceHeight;

Manually adjusting a bands boundaries, or changing fixed bands which impact on the placement of the current band, or calling NewPage within a band, will potentially invalidate current band settings. To reinstate band settings or apply new settings, call ResetBand or ResetBandBoundaries. Neither of these procedures moves the cursor, so call CursorHome, for example, to default the cursor back to the bands top left position if required.

procedure ResetBand(	resets boundary, font and tab settings
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
RetainTopBoundary: Boolean = False);	True to retain original top position of a dynamic band rather than use the current YPos.
procedure ResetBandBoundaries(	resets boundaries (but not font and tab settings)
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
RetainTopBoundary: Boolean = False);	True to retain original top position of a dynamic band rather than use the current YPos.

Band Properties and Behaviour

Each time a band is executed within a frame, the bands SetupBand method is called to set boundaries and apply font and tab settings as appropriate. For dynamic bands, band height and line constraints (if defined) are checked, and a new page may be automatically triggered. The OnTripPageBefore and OnTripPageAfter events allow you to intercept these automatic page breaks and take action if necessary.

The following properties are common to all bands:

property FontIndex: Integer;	restores a saved font (1..10) when SetupBand called (default 0, font left unchanged)
property TabIndex: Integer;	restores a saved line tab set (1..10) when SetupBand called (default 0, tabs left unchanged)

For fixed bands (Letterhead, Letterfoot, PageHeader, PageFooter, Remittance):

property Height: Double;	fixed height allocated to band if 0, ReportInterface default heights (from DefaultFixedBandHeights) are applied

For dynamic bands (BodyHeader, BodyFooter, GroupHeader, GroupFooter, Row):

property MinHeight: Double;	minimum band height required before a NewPage is called in SetupBand default 0, no mimimum height constraint applies
property MinLines: Integer;	minimum number of lines (using bands font) required before a NewPage is called in SetupBand default 0, no mimimum lines constraint applies

PageFrames and MasterFrames both support an automatically nested detail frame, being a MasterFrame or a DetailFrame:
(but any number of frames may be nested within a report structure)

property DetailFrame: TDetailFrame;	sets the nested frame that will be called to implement the Detail band of the parent frame.

PageFrames, MasterFrames and DetailFrames provide the following page events:

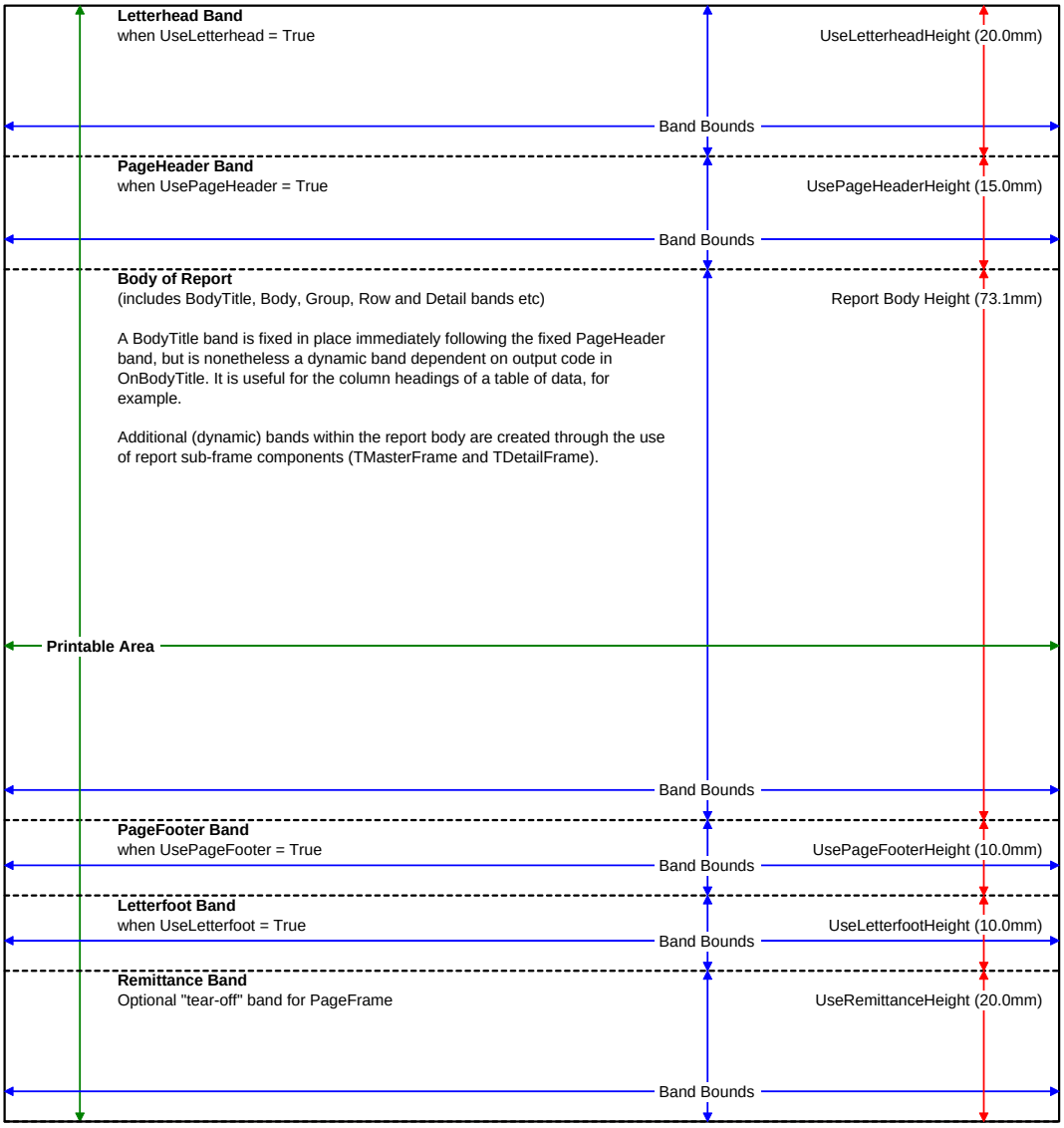
property OnTripPageBefore(	fires before a new page is triggered due to space constraints
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
ReportWriter: TReportWriter;	ReportWriter generating the report
ReportFrame: TReportFrame;	ReportFrame in which the new page has been triggered
ReportBand: TReportBand);	band in which the new page has been triggered
property OnTripPageAfter(	fires after a new page has been triggered due to space constraints
ReportInterface: TReportInterface;	ReportInterface component managing the reporting process
ReportWriter: TReportWriter;	ReportWriter generating the report
ReportFrame: TReportFrame;	ReportFrame in which the new page has been triggered
ReportBand: TReportBand);	band in which the new page has been triggered

VPE+ Application Reporting Interface

Report Frames (TxxxFrame) Frames & Bands

Page Frame Metrics

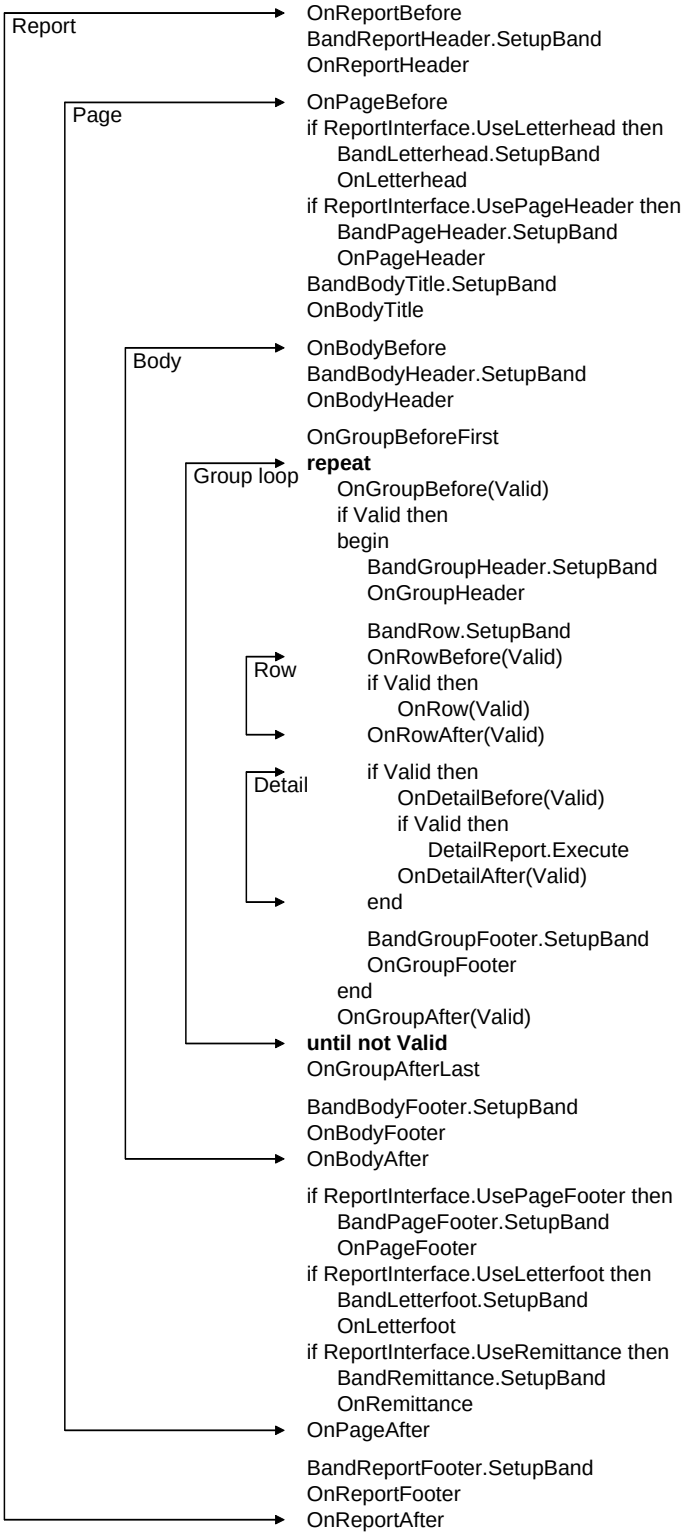
The following diagramme illustrates band layout on the printable area of the paper. Fixed band heights shown reflect the actual defaults for these bands, with whatever page height remains in the diagramme being allocated to the dynamic report body area.



VPE+ Application Reporting Interface

Report Frames (TPageFrame) Page Frame

A framework providing page structure for a report.
Can be executed per Execute(ReportWriter) method.
Allows an additional sub-frame to be called (per PageFrame.DetailFrame) - either a TMasterFrame or TDetailFrame.



ReportHeader is outside the (fixed band) page structure.
A new page is automatically started after a ReportHeader

INTERFACE FIXED BAND DEFAULTS: Letterhead, PageHeader, PageFooter and Letterfoot fixed bands are enabled and sized by default using ReportInterface properties DefaultFixedBandEnabled and DefaultFixedBandHeights. The latter also sets a default height for the Remittance band, but this must be enabled using PageFrame.EnableRemittance.

FIXED BAND CONTROL: In ReportWriter.OnExecuteReport, override the default fixed band properties by:
> Enabling/Disabling bands using properties UseLetterhead, UsePageHeader, UsePageFooter, UseLetterfoot (T/F)
> Setting band heights using properties UseLetterheadHeight, UsePageHeaderHeight, UsePageFooterHeight, UseLetterfootHeight.
BodyTitle follows the fixed PageHeader band, but its height is dynamic, allowing column headers to be output, for example, immediately prior to the body of the report.

GROUP LOOP: The single ("repeat") loop within the body of the PageFrame is a master/detail loop. ie One loop is made for each primary ("master") row, with the next master row typically being called in OnGroupAfter.
For a simple loop without repeatedly cycling this structure, data records can be cycled within the OnRow handler instead.

BAND SETUP: SetupBand applies the bands font and line tab set (if defined).
Band boundaries reflect the page extent available to the current band and can be referenced in ReportInterface (BandLeft, BandTop, BandRight, BandBottom).
A manual call to SetupBand will recalculate boundaries, but otherwise they remain static on the same page.

BAND REPRINTING: Dynamic bands can be re-printed (eg after a new page is manually initiated) by calling these procedures:
PrintBodyHeader, PrintBodyFooter
PrintGroupHeader, PrintGroupFooter

REMITTANCE: Call PageFrame.EnableRemittance (typically in OnBodyFooter) to enable this band with the height specified by BandRemittance.Height or ReportInterface.DefaultFixedBandHeights.RemittanceHeight.
Check PageFrame.EnoughSpaceForRemittance first, and call NewPage if necessary.
The band is automatically disabled when processed, but otherwise call PageFrame.DisableRemittance.

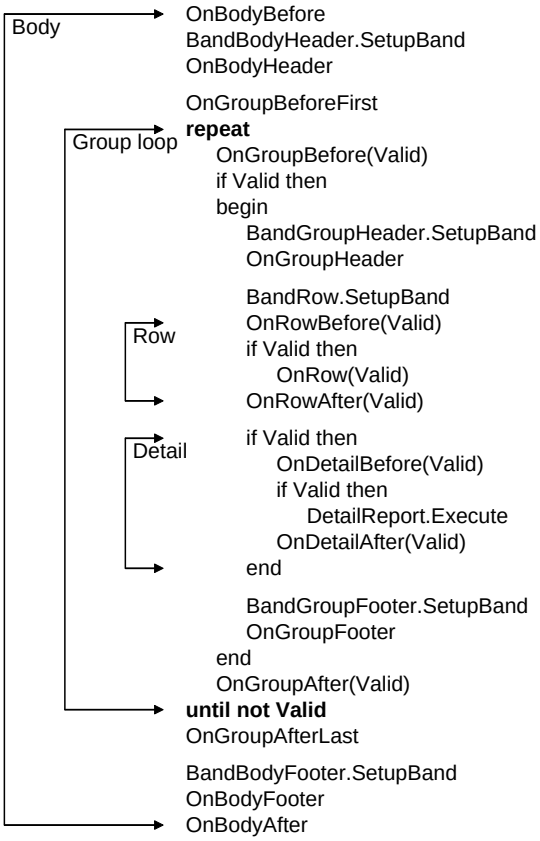
ReportFooter is outside the (fixed band) page structure.
A new page is automatically started before a ReportFooter.

function IsTopOfPageBody: Boolean; returns True if YPos is at the top of the body of the page

VPE+ Application Reporting Interface

Report Frames (TMasterFrame) Master Frame

A report sub-frame providing a master/detail structure without the page elements of TPageFrame.
Can be used as a detail frame for a TPageFrame, or executed independently per Execute(ReportWriter) method.
Allows an additional sub-frame to be called (per MasterFrame.DetailFrame) - either another TMasterFrame or TDetailFrame.



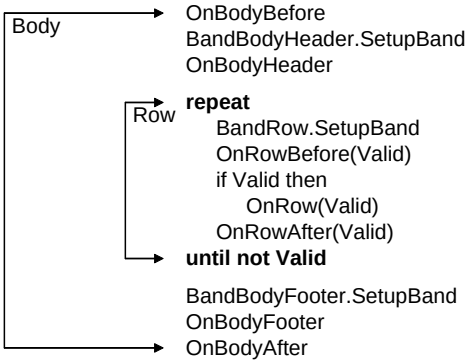
GROUP LOOP: As with TPageFrame, the single ("repeat") loop within the body of the MasterFrame is a master/detail loop. For a simple loop without repeatedly cycling this structure, data records can be cycled within the OnRow handler instead.

BAND REPRINTING: Dynamic bands can be re-printed (eg after a new page is manually initiated) by calling these procedures:
PrintBodyHeader, PrintBodyFooter
PrintGroupHeader, PrintGroupFooter

Report Frames (TDetailFrame) Detail Frame

A report sub-frame providing a simple detail structure without the page & group elements of TPageFrame and TMasterFrame.

Can be used as a detail frame for a TPageFrame or TMasterFrame, or executed independently per Execute(ReportWriter) method.



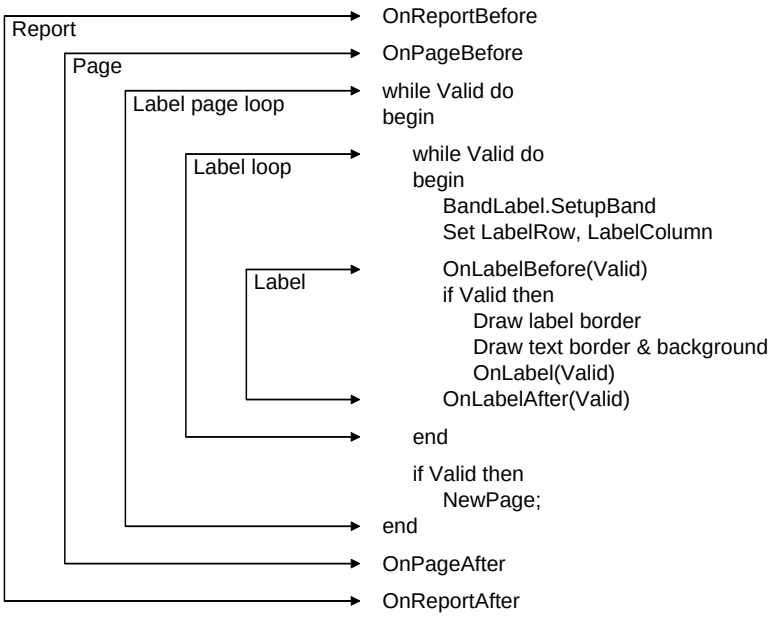
ROW LOOP: The single ("repeat") loop within the body of the DetailFrame sets up the row band on each cycle.
For a simple loop without repeatedly cycling this structure, data records can be cycled within the OnRow handler instead.

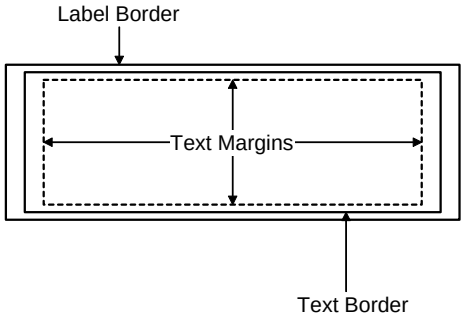
BAND REPRINTING: Dynamic bands can be re-printed (eg after a new page is manually initiated) by calling these procedures:
PrintBodyHeader, PrintBodyFooter

Call a bands SetupBand procedure to reset its font and line tabs and top boundary (BandTop). The top boundary can also be optionally reset with a call to ResetBandBoundaries(ReportInterface, False), but otherwise it remains fixed on any given page.

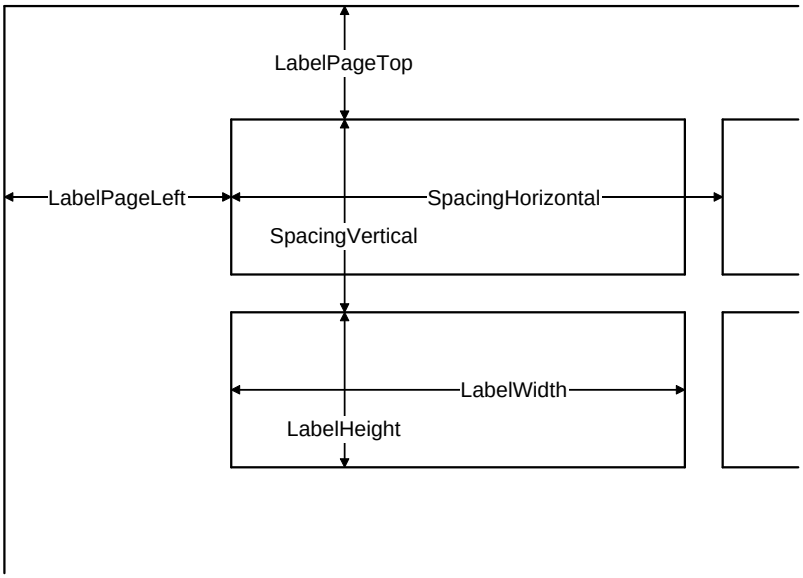
Report Frames (TLabelFrame) Label Frame

A report frame for generating labels. Code label output in the OnLabel event.
Execute the label report per Execute(ReportWriter) method.





The label border (outer label extent) is defined by the label dimensions described below.



property LabelPageLeft: Double; (0 mm)
Left page edge to left edge of first label column.

property LabelPageTop: Double; (11 mm)
Top page edge to top edge of first label row.

property SpacingHorizontal: Double; (70 mm)
Space between left edges of adjacent label columns.

property SpacingVertical: Double; (25 mm)
Space between top edges of adjacent label rows.

property LabelWidth: Double; (70 mm)
Width of label.

property LabelHeight: Double; (25 mm)
Height of label.

---

property LabelColumns: Integer; (1)
Number of label columns per label sheet.

property LabelRows: Integer; (1)
Number of label rows per label sheet.

property LabelSkipCount: Integer; (0)
Labels to skip on first sheet.

property LabelStartColumn: Integer; (1)
Starting column of first label on first sheet.

property LabelStartRow: Integer; (1)
Starting row of first label on first sheet.

---

property LabelColumn: Integer;
Current label column being printed.

property LabelRow: Integer;
Current label row being printed.

property LabelIndex: Integer;
Index of current label being printed (1..LabelCount).

property LabelPageIndex: Integer;
Index of current label on the page (1..LabelsPerPage).

property LabelBorder: TBorderType;	sets the printed label border type: btNone (default), btRect, or btEllipse
property LabelBorderCornerRadius: Double;	rounds the label border corners to given radius when LabelBorder = btRect, 0 mm (default) for s
property LabelBorderPenWidth: Double;	sets the label border pen: 0.3 mm (default)
property LabelOrder: TLabelOrder;	set the direction in which labels are printed: loByRow or loByColumn)
TLabelOrder	direction in which labels are sequentially printed
loByRow	print labels across then down (default)
loByColumn	print labels down then across
property TextBorder: TBorderType;	sets the text border type: btNone (default), btRect, or btEllipse
property TextBorderMargins: TMargins;	sets margins for the text border relative to the label border Left (2.5 mm), Top (1 mm), Right (2.5 mm), Bottom (1 mm)
property TextBorderCornerRadius: Double;	rounds the text border corners to given radius when TextBorder = btRect, 0 mm (default) for sq
property TextBorderPenWidth: Double;	sets the text border pen width: 0.3 mm (default)
property BackgroundColour: TColor;	sets background colour within the text border (default clNone)
property TextMargins: TMargins;	sets margins for the inner text print area relative to the label border Left (5 mm), Top (2 mm), Right (5 mm), Bottom (2 mm)
property BandLabel: TLabelBand;	allows a FontIndex and TabIndex to be set for OnLabel output

Report Frames (TColumnFrame) Column Frame

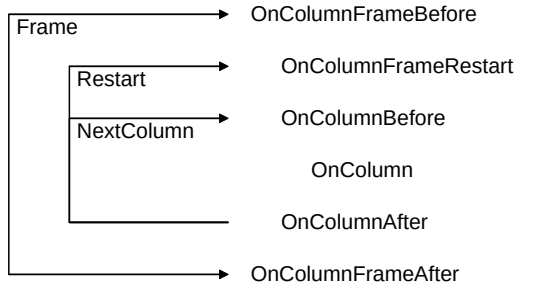
Use TColumnFrame to manage output in columns. Start by defining columns using the methods AddColumn or AddColumnWidth (to add sequential columns one at a time) or AddColumns (to add a number of equal width columns). Column boundaries are relative to the left side of the column frame, which can in turn be placed anywhere on the page. Once the ColumnFrame has been executed, you cannot add further columns, or alter existing column definitions (but you can move the column frame).

By default, the frame is placed to fill the remaining space in the active band (from BandLeft and the current YPos) when it is executed, so ensure the correct band boundaries are applied at the time. Call a bands ResetBand method if necessary. However, the frame can be repositioned (ie change its top left position) using the method PlaceColumnFrame, allowing column output to be directed to different areas of the same page, or to another page as required. The height of the column block can be adjusted using methods SetColumnFrameBottom or SetColumnFrameHeight - the block does not have to fill the page.

Call the frames Execute method to start column output. OnColumnFrameBefore allows the frames initial position to be adjusted, or resources to be accessed. Note that if you include other report output in this handler, it may encroach on the already allocated column output space, so remember to move the frame accordingly using eg PlaceColumnFrame(NA, YPos). Clean-up can be handled in OnColumnFrameAfter. Once executed, you can only change the relative column frame position, or the height available for column output.

Columns have methods and properties similar to those of bands to indicate boundaries and space remaining (see below). Use these methods rather than the band methods (although the latter are still valid). Call NextColumn (rather than NewPage) to start a new column. When called in the last column, NextColumn will return to the first column and notify you in OnColumnFrameRestart. You must call NewPage in this event handler if appropriate, or otherwise reposition the column frame to continue output.

Column headers or line tabs for column output can be set in OnColumnBefore. TextBlocks can be output to columns using PrintColumnText. Otherwise, output to columns is no different from output to bands.



The "loop" in a ColumnFrame is controlled in your OnColumn code by calls to NextColumn. After the last column, output returns to the first column and OnColumnFrameRestart is called to allow the frame to be repositioned. This event does NOT fire on the first cycle.

Column Frame functions:

function ColumnCount: Integer;	returns the number of columns defined
function ColumnCentreXPos: Double;	returns the centre XPos of the current column
function ColumnCentreXPos(AColumnIndex: Integer): Double;	returns the centre XPos of a particular column index of the column (1-based)
function ColumnHeightLeft: Double;	returns the height remaining in the current column
function ColumnLeft: Double;	returns the left boundary of the current column
function ColumnLeft(AColumnIndex: Integer): Double;	returns the left boundary of a particular column index of the column (1-based)
function ColumnLinesLeft: Integer;	returns the number of lines remaining in the current column
function ColumnRight: Double;	returns the right boundary of the current column
function ColumnRight(AColumnIndex: Integer): Double;	returns the right boundary of a particular column index of the column (1-based)
function ColumnWidth: Double;	returns the width of the current column
function ColumnWidth(AColumnIndex: Integer): Double;	returns the width of a particular column index of the column (1-based)
function EnoughColumnHeight(HeightNeeded: Double): Boolean;	returns True if the current column will accommodate a given height the height needed
function EnoughColumnLines(LinesNeeded: Integer): Boolean;	returns True if the current column will accommodate a given number of lines the number of lines needed

Report Frames (TColumnFrame) Column Frame

Column Frame procedures:

procedure AddColumn(	adds the next sequential column definition
ALeft,	the left column boundary relative to the column frames left
ARight: Double);	the right column boundary relative to the column frames left
procedure AddColumns(	adds a number of equal columns to fill a given width
AColumnCount: Integer;	the number of columns to add
AColumnFrameWidth,	the width in which the columns must fit
AColumnGap: Double);	the gap to use between columns
procedure AddColumnWidth(	adds the next sequential column definition with a given width
AColumnWidth,	the width of the column
AColumnGap: Double);	the gap to use after the previous column
procedure ClearColumnList;	clears existing column definitions
procedure Execute;	executes the column frame
procedure NextColumn;	moves to the next column, or back to the first column
procedure PlaceColumnFrame(	repositions the column frame
ALeft,	the left position of the column frame
ATop: Double);	the top position of the column frame
procedure SetColumnFrameBottom(	sets the bottom position of the column frame
ABottom: Double);	the bottom position
procedure SetColumnFrameHeight(	sets the height of the column frame
AHeight: Double);	the height

Column Frame properties:

property ColumnFrameBottom: Double;	returns the bottom extent of the column frame (or columns)
property ColumnFrameTop: Double;	returns the top extent of the column frame (or columns)
property ColumnIndex: Integer;	returns the curren tcolumn index (1-based)
property ReportWriter: TReportWriter;	the TReportWriter generating the report (must be set before executing the column frame)

Printing Text Blocks to Columns

You can use a TTextBlock to automatically output blocks of text to columns. Configure and open a TTextBlock with the desired text, then pass it to the method PrintColumnText.

procedure PrintColumnText(	outputs blocks of text to columns
ATextBlock: TTextBlock;	an opened TTextBlock object
AColumnTextMode: TColumnTextMode);	controls how the text is output
TColumnTextMode	column text output modes
ctmFillColumns	columns are sequentially filled, top to bottom then across as defined
ctmEvenColumns	the column frame height is re-calculated to fit the text evenly across all columns

When printing text in ctmFillColumns mode, text will follow on from whatever has already been output to the columns, proceeding down then across. You can then follow with more output as required. However, using ctmEvenColumns mode necessarily dedicates the column frame to the text block exclusively because the columns are adjusted to achieve an even fill with the text being output.

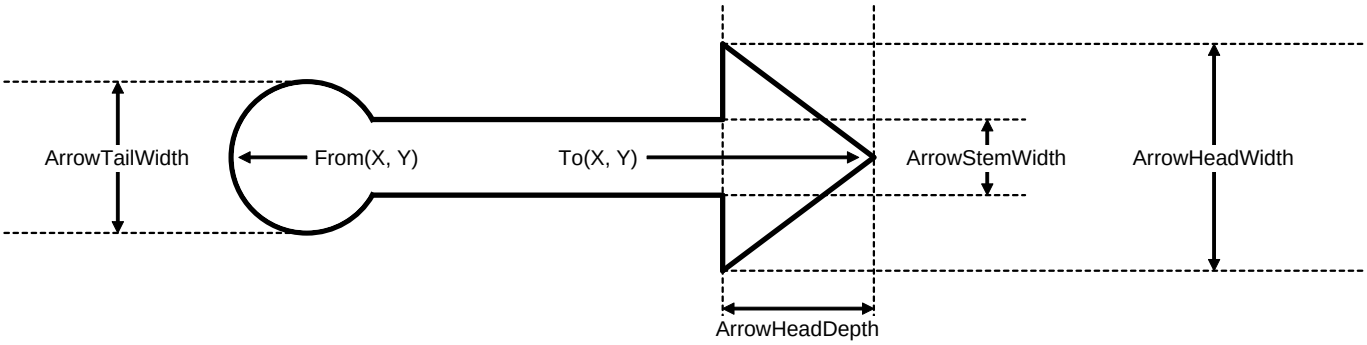
Note also that while you can easily output text in the OnColumnFrameBefore handler, the default column frame output zone will probably overwrite it - so remember to relocate the frame accordingly (using method PlaceColumnFrame).

VPE+ Report Interface (TReportInterface) Arrows

Arrows can be drawn using the method DrawArrow. The size and shape of the arrow is set or adjusted using the various properties and methods described below. Call SetDefaultArrow to start with a default arrow (see the default specifications below).

Arrow dimensions can be set collectively using the method SetArrowDimensions. Pass brush colour clNone or NA to DrawArrow to set a transparent brush, giving the arrow an outline according to the pen settings.

procedure DrawArrow(	draws an arrow from point (FromX, FromY) to point (ToX, ToY)
FromX,	XPos of arrow starting point ("tail" of arrow)
FromY,	YPos of arrow starting point ("tail" of arrow)
ToX,	XPos of arrow end point ("head" of arrow)
ToY: Double;	YPos of arrow end point ("head" of arrow)
APenWidth: Double = NA;	pen width for arrow lines (default = NA = pwNormal = 0.3 mm)
APenColour: TColor = NA;	pen colour for arrow lines (default = NA = clBlack)
APenStyle: TVPEPenStyle = psSolid;	pen style to use: psSolid (default), psDash, psDot, psDashDot, psDashDotDot
ABrushColour: TColor = clBlack);	background brush colour (default = clBlack; NA = clNone = transparent)



NB VPE lines are drawn CENTRED on the respective coordinate, so a pen will extend a half pen width beyond the expected point.
ie The resultant width of the arrow stem is (ArrowStemWidth + ArrowPenWidth), and a plain line drawn to match the stem must use this width.

Drawing properties:

property ArrowHeadShape: TArrowShape;	shape of arrow head: asNone, asTriangle (default), asDot
property ArrowTailShape: TArrowShape;	shape of arrow tail: asNone (default), asTriangle, asDot

Dimension properties: (refer to the arrow diagramme above)

property ArrowHeadDepth: Double;	depth of arrow head (for asTriangle shape only), default 1.5 mm
property ArrowHeadWidth: Double;	width of arrow head (of asTriangle shape, or diameter of asDot shape), default 1.5 mm
property ArrowTailDepth: Double;	depth of arrow tail (for asTriangle shape only), default 1.5 mm
property ArrowTailWidth: Double;	width of arrow tail (of asTriangle shape, or diameter of asDot shape), default 1.5 mm
property ArrowStemWidth: Double;	width of arrow stem, default 1/4 ArrowHeadWidth NB ArrowStemWidth = 0 effectively draws an arrow stem at pen width.

Other arrow methods:

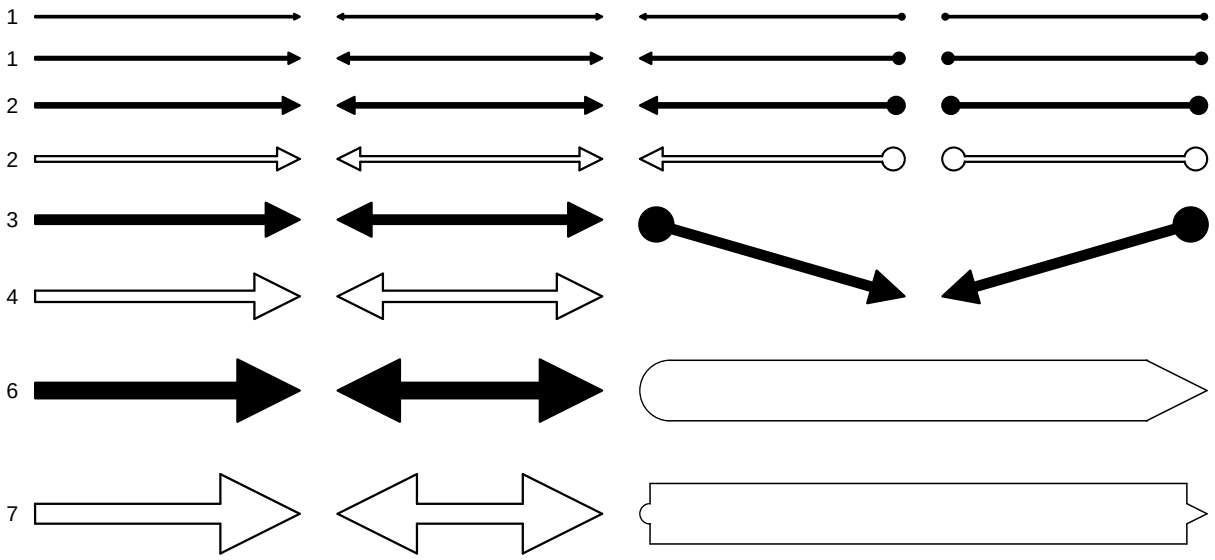
procedure ScaleArrow(	scales the current arrow dimensions
ArrowScaleFactor: Single);	factor by which to scale the arrow (the arrow pen is not scaled)
procedure SetArrowDimensions(	set the arrow dimensions collectively
StemWidth,	width of arrow stem (the "line" component)
HeadDepth,	depth of the head triangle shape, not used for the head dot shape
HeadWidth,	width of the head triangle shape, or diameter of the head dot shape
TailDepth,	depth of the tail triangle shape, not used for the tail dot shape
TailWidth: Double);	width of the tail triangle shape, or diameter of the tail dot shape
procedure SetDefaultArrow;	resets arrow dimension, pen and shading properties to defaults
Defaults are:	
ArrowHeadShape	= asTriangle;
ArrowTailShape	= asNone;
ArrowHeadDepth, ArrowTailDepth	= 1.5 mm
ArrowHeadWidth, ArrowTailWidth	= 1.5 mm
ArrowStemWidth	= (ArrowHeadWidth / 4) = 0.375 mm

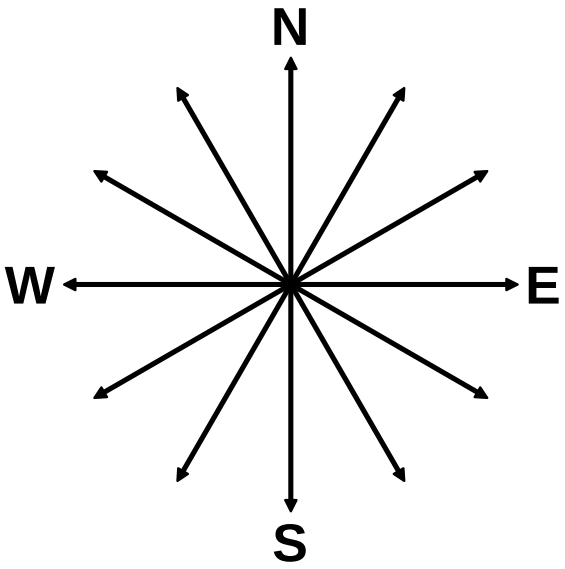
VPE+ Report Interface (TReportInterface) Arrows

Some arrow examples:

ArrowScaleFactor (applied to default arrow size):

Other arrow variations:





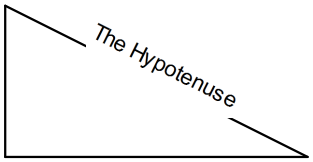
VPE+ Report Interface (TReportInterface) Text Rotation

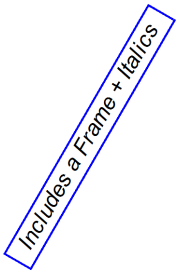
VPE allows text to be rotated at 90 degree intervals. To rotate and justify a single line of text to any angle, VPE+ provides the method PrintRotatedText.

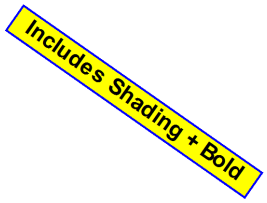
In principle, VPE+ writes the text into an enhanced metafile (EMF), rotates this EMF and then draws the result into the VPE document according to the justification parameters supplied. The entire process is managed with (small) memory streams - no on-disk files are created.

You can specify a font style or offset the (x, y) placement of the text (in addition to justification). Use the procedure BumpRotationBox to specify a frame, shading, or extra left and right text margins within the text box.

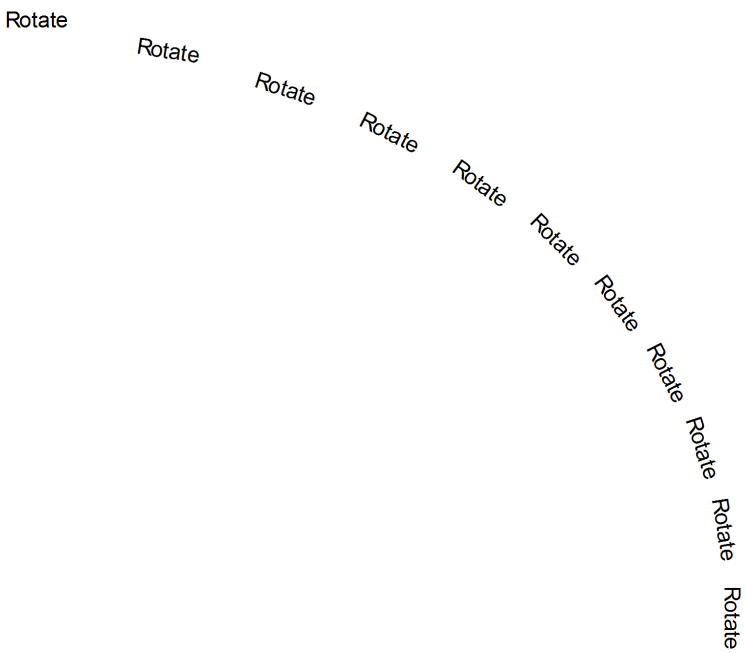
Some output examples are included here:







procedure PrintRotatedText(	prints a line of rotated text
Text: string;	text to rotate and print
AngleDegrees: Double;	angle of rotation in degrees (clockwise from horizontal = 0 degrees)
AJustify: TPrintJustify;	alignment of text line relative to AlignTo point along the line of text ("horizontal") jLeft, jCentre, jRight
ALineMetric: TLineMetric;	alignment of text line relative to AlignTo point perpendicular to the line of text ("vertical") lmFontTop, lmFontMiddle, lmFontBaseline, lmFontBottom, lmLineTop, lmLineMiddle, lmLineBottom
AlignToX, AlignToY: Double;	point (AlignToX, AlignToY) about which the text line should be aligned
ATextStyle: TTextStyle = tsNormal;	font style to apply (default = tsNormal)
OffsetX: Double = 0;	horizontal offset for text line in addition to AJustify (negative left, positive right)
OffsetY: Double = 0;	vertical offset for text line in addition to ALineMetric (negative up, positive down)
procedure BumpRotationBox(	specifies text box settings for an item of rotated text
APenColour: TColor = clNone;	colour of text box frame (NA = default = clNone = no box frame)
ABrushColour: TColor = clNone;	colour of text box shading (NA = default = clNone = no box shading)
ALeftTextMargin: Double = 0;	left text margin (default = 0) between left frame line (if any) and text
ARightTextMargin: Double = 0;	right text margin (default = 0) between right frame line (if any) and text
procedure ClearRotationBoxBumps;	clears any bumps applied using BumpRotationBox

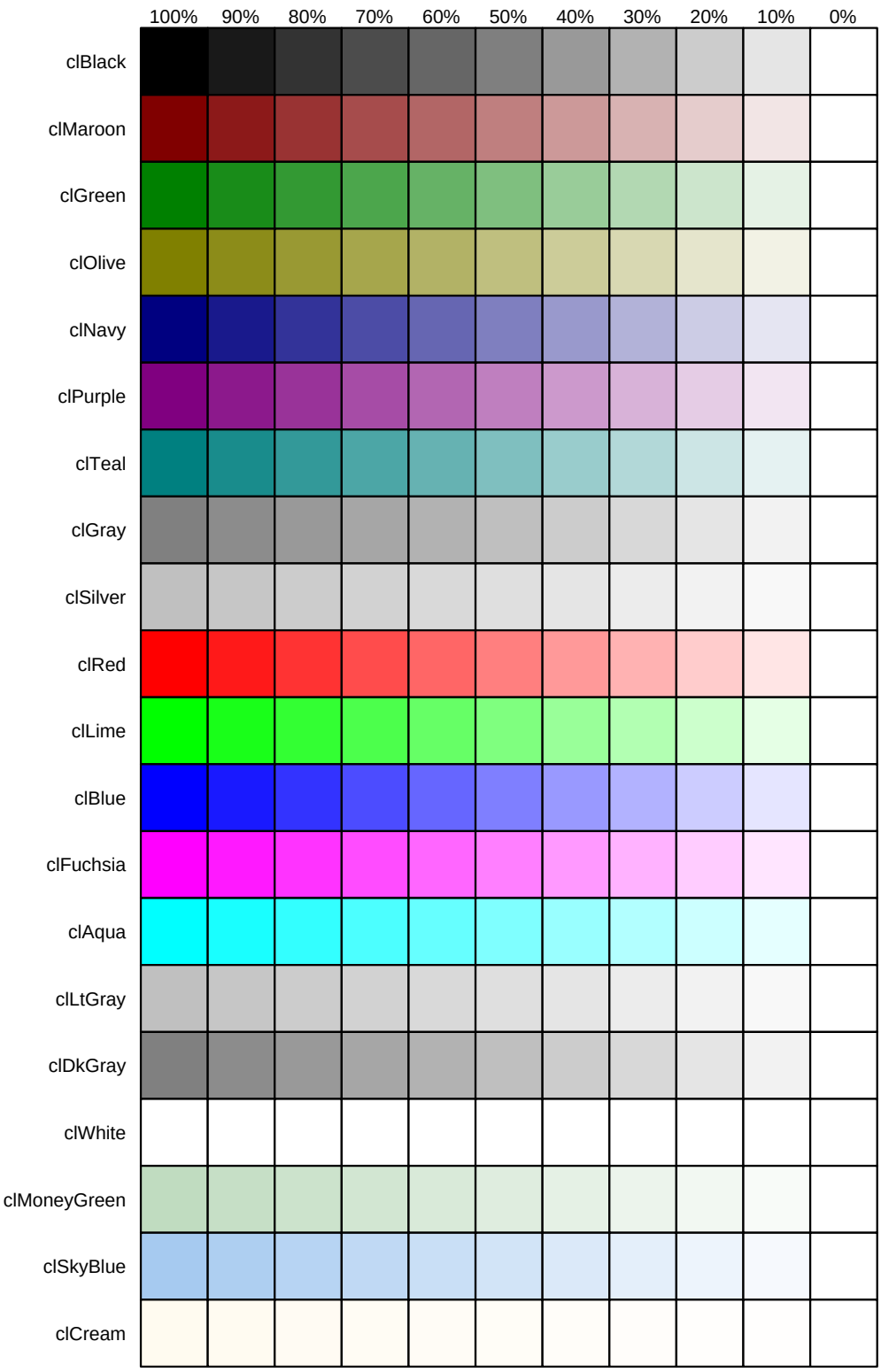


VPE+ Application Reporting Interface

VPE+ Report Interface (TReportInterface) Colour Shades

procedure ColourPercent(
BaseColour: TColor;
PercentColour: SmallInt): TColor;

returns a colour adjusted to a lighter shade
base colour to lighten (clNone is returned if clNone is passed)
percentage value to lighten colour by



VPE+ Application Reporting Interface

Categorical Index

This index lists only VPE+ methods, events and properties. There are a great deal more associated with VPE itself. Consult the comprehensive VPE documentation for more information.
Note also, that some items appear in more than one category below, as appropriate. The listing is intended as a brief indication and reminder of available features, and as a cue to finding related items.

Arrows

Bands

Column Frame

Custom Formats

Detail Frame

VPE+ Application Reporting Interface

Categorical Index

Devices

Drawing

Fonts & Font Metrics

Images

General Interface

VPE+ Application Reporting Interface

Categorical Index

Label Frame

Master Frame

Page Frame

Page Numbering

Position & Cursor

VPE+ Application Reporting Interface

Categorical Index

Printing

Report Writer (TReportWriter)

RTF On the Fly

Tabs

VPE+ Application Reporting Interface

Categorical Index

Text Blocks