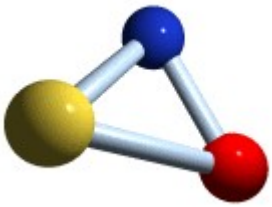
Virtual Print Engine v1.3

Demo Version



Copyright © 1996 by IDEAL Software

T. Radde

DISCLAIMER:

IDEAL Software and T. Radde hereby disclaim any warranty, either expressed or implied, as to the suitability of this product for any purpose whatsoever. IDEAL Software and T. Radde will not be responsible for any damage, loss of data, incorrect data, or other problems arising from the use or misuse of this software.
By using this product, the user agrees to accept responsibility for any problems, and to hold IDEAL Software and T. Radde free of any liability (NOTE: You should include a disclaimer of your own with any files you distribute).

THE INFORMATION AND CODE PROVIDED HEREUNDER (COLLECTIVELY REFERRED TO AS "SOFTWARE") IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL IDEAL SOFTWARE AND/OR T. RADDE BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF BUSINESS PROFITS, OR SPECIAL DAMAGES AS THE RESULT OF USING THIS PROGRAM, EVEN IF IDEAL SOFTWARE AND/OR T. RADDE HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

USING (RUNNING) THIS SOFTWARE ACKNOWLEDGES YOUR ACCEPTANCE OF THIS AGREEMENT.

License Agreement:

By using the flag VPE_NO_INFOBTN, and/or using VPE without a preview, you agree to include a copyright notice such as:"'Virtual Print Engine' Copyright © by IDEAL Software, T. Radde" in your "About" dialog or the help-file of your software (if they exist) or in your documentation.

You agree not to release documentation concerning how to control VPE from any programming language (i.e. description of the function call interface or any flags) to third parties.

When using the full licensed version:

The only redistributable files are: VPENGINE.DLL, DAVINCI.DLL and EASYBAR.DLL (VPE32. DLL, DAV32.DLL, EZBAR32.DLL)

VPE may only be given away to third parties as an integrated part of your software. Your software must have significant main functions other than VPE. This means that it may not be considered a product in any way; or that is in any way equal to VPE; and that it may only be used for printing/previewing. Examples of applications that may not be developed with VPE: Report Generator, Barcode Printing Application, Graphics Presentation Software, etc.

This license gives you the possibility to integrate VPE in your products and give VPE to third parties. But these third parties have no right to give VPE away to other third parties. If you have such needs, contact T. RADDE for special conditions.

Your application may generate reports, print barcodes or make (graphical) presentations.

Note: EASYBAR.DLL (EZBAR32.DLL) may only be used/distributed in conjunction with VPE. You may not use it outside of VPE. If you have such needs, contact BOKAI Corp.:
Bokai Corporation
1221 Dundix Rd., #106
Mississauga, ON L4Y 3Y9
Canada
Fax: (+1) 905 276-7692

Note, that DAVINCI.DLL (DAV32.DLL) may only be used/distributed in conjunction with VPE. You may not use it outside from VPE. If you have such needs, contact:
Ing. Büro Bernd Herd
Heidelberger Landstr. 316
64297 Darmstadt
Germany
Tel and Fax: 06151 / 59 12 16

USING (RUNNING) THIS SOFTWARE ACKNOWLEDGES YOUR ACCEPTANCE OF THIS AGREEMENT.

On Ordering VPE:

Many people outside Germany may fear language difficulties or of missing support . Read a customer's comment:

"We have searched the market, and have found no other print engine with your capabilities.

"I was concerned about buying from a European company, because of possible language problems and/or lack of support. Your quick email responses have laid that concern to rest. Feel free to use Monarch Bay as a US reference."
Ron Phillips, Monarch Bay Software, Inc. ("HelpTrac™" application), Houston, Texas

This is the demo-version of VPE. It is fully functional, but prints only half pages and a demo-banner. Also, the graphics-import DLL gives a shareware message each time an image is imported.

This demonstration program is freeware. Give it to your friends, colleagues and anyone else who may be interested.

You must distribute this program unchanged and intact; all files must be included with it, including "vpe.wri".

Freeware/Shareware vendors may distribute this version of the program freely.

The difference of the full version compared to this demo version is that the pages are printed completely and without any notes. Also, the shareware message for the graphics-import DLL will disappear.
There is no functional difference between the unlicensed and licensed versions.

A 32-bit version is also available for the same pricing as the 16-bit version. The 32-bit version is fully compatible to the 16-bit version, **except** that the maximum page size is 999cm x 999cm regardless of the printer, **and** that the Adobe / Microsoft Filters are not supported for graphics import. The combination price for 16-bit and 32-bit versions together is the single price, multiplied by a factor of 1.5.

The full version of VPE is available as a trial and development version for $149. Please note: the $149 version is NOT license-free.
The trial version grants you the right to use VPE ONLY in-house in one location. If you want to distribute VPE within your products, or if you want to use it in different office branches, you will need to license it. The price for the licensed version is $399. This allows you to make as many copies as you like (embedded within your products - see "License Agreement"). If you purchase the $149 version first, you may upgrade to the full license later for $250 (current).

Order by check or money order. Checks from outside Germany: Please send Eurocheck in DM currency. Otherwise be so kind as to add $15 to the sum; that is, the bank's foreign money transactions charge. If you can't receive VPE via CIS e-mail (currently I have no Internet binary support), or if you wish to receive VPE for any other circumstances via mail, please add $10 for packaging and shipping (see pricing on next page).

Pricing:

All prices as of 2/96	**Unlicensed** (16 OR 32 bit)	**Unlicensed** (16 + 32 bit)	**Licensed** (16 OR 32 bit)	**Licensed** (16 + 32 bit)
Base Price	$149	$223	$399	$598
Outside Germany, no Eurocheck in DM	+ $15	+ $15	+ $15	+ $15
Via CIS e-mail	free	free	free	free
Via mail	+ $10	+ $10	+ $10	+ $10

The **16-bit demo** can be found on CompuServe / Winsdk / Printing.
The **32-bit demo** can be found on CompuServe / Winsdk / Tools - Third Party.
Search in both cases for the keyword „preview“, to get the latest version.

I'm looking for additional interfaces and demos for all kinds of programming languages, and also for distributors in all countries worldwide.

My address:
T. Radde
Grefrather Weg 96
41464 Neuss
Germany

CompuServe: 100 430, 34 27

Internet: 100430.3427@compuserve.com

Table of Contents

1 Files in this Archive

c\	- directory with header-files and demo source for use with C/C++ compilers, contains also import-library for other compilers
delphi\	- directory with interface and demo source for use with Delphi
pascal\	- directory with demo source for use with Borland Pascal (interface is vpengine.pas in \delphi)
sqlwin\	- directory with APL and APT demo source for use with SQLWindows
vb\	- directory with interface and demo source for use with Visual Basic (interface is usable for Access, demo is readable for Access users)
davinci.dll (dav32.dll)	- graphics dll from Ing. Büro Bernd Herd (Darmstadt / Germany) (fully licensed ONLY for use with VPE)
easybar.dll (ezbar32.dll)	- barcode dll from BOKAI Corporation (Mississauga / Canada) (fully licensed ONLY for use with VPE)
fruits.bmp	- a sample bitmap for vpedemo.exe
gew.tif	- a sample bitmap for vpedemo.exe
logo.bmp	- a sample bitmap for vpedemo.exe (and the IDEAL Software logo)
minidemo.cpp	- mini sample C-program on how to use VPE (displays measuring in inches)
minidemo.exe (mini32.exe)	- the compiled mini sample program
vpe.wri	- this file
vpedemo.exe (vpedmo32.exe)	- a demonstration of what VPE can do (explanation see below)
vpengine.dll (vpe32.dll)	- the engine itself

All interfaces and demos (except for C/C++) are unsupported.
They are an add-on and not an essential part of this product.
This documentation overrides all definitions and implementations of interfaces and demo's.
(Most of them are written by other people.)

There is only a 16-bit import-library. 32-bit library-formats differ from compiler to compiler. Each compiler has its tools, to generate import-libraries. Please consult your manual.

There might also be a directory or archive called "**expengin**". This is a demo of another product from IDEAL Software. It's a DLL for evaluating numeric expressions like "(x -5) * sin(y)" during runtime of your program.

Don't forget the "-d" parameter for pkunzip to extract the subdirectories. ("pkunzip -d vpengine.zip")

2 Shortcut: What's New in Version 1.3

Now supports 21 barcode types

Now supports the following graphics file formats:
- Windows and OS/2 Bitmaps (2 / 16 / 256 / True Color)
- Windows WMF (Metafile)
- AutoCAD DXF
- GIF (2 / 16 / 256 Colors)
- PCX (2 / 16 / 256 Colors)
- JPG (256 / True Color)
- TIFF 5.0 (2 / 16 / 256 / True Color, LZW / PackBits / Fax G3 / Fax G4 / Tiled Images)
- Installed Adobe / Microsoft filters (some restrictions and only 16-bit version; for example, those that come with Word for Windows)
- Due to the new supported graphics file formats, the function VpeBitmap has been renamed to VpePicture
- New function VpeGetPictureTypes()
- Flag VPE_PIC_KEEP removed
- New flags: VPE_PIC_KEEPIMAGE, VPE_PIC_DISCARD_DIB_DRAW, VPE_PIC_KEEP_DIB_PAGE

Page size is freely defineable:
- 16-Bit version: up to 138cm x 138cm (for 600 DPI Printer); 69cm x 69cm (for 1200 DPI printer); etc.
- 32-Bit version: 999cm x 999cm
- New parameters page_width and page_height for function: VpeOpenDoc (both in 1/10 mm)
- Default parameters for DIN A 4 and US-Standard-Letter
- New function: VpeSetRulersMeasure() - rulers can show cm or inch measurement
- Output rectangle (similar to page-dimensions) can be defined and retrieved
- Each page in a document may have individual output rectangle

Improved dynamic positioning:
- Each page keeps track of the positioning rectangle of the last inserted object
- All objects can be positioned and sized relative to this rectangle
- All objects can be sized relative to their upper left corner
- Removed functions: VpeGetX and VpeGetY
- For new functions and flags see "Dynamic Positioning"

Improved printer handling:
- VPE_LANDSCAPE flag reflected in printer setup-dialog
- VPE can do the printer setup separate from printing, printing can be done without setup
- VPE can now remember the last printer, driver and port used, and keep the last settings for that printer
- VPE is able to save/read settings to/from file, so you can save settings not only for your application, but for every individual document type
- VpePrintDoc new parameter "BOOL with_setup"
- New function: VpeSetupPrinter

Improved control:
- New message: VPE_HELP
- New toolbar flags: VPE_NO_HELPBTN, VPE_ROUTE_HELP, VPE_NO_INFOBTN

New file managing and memory swapping features:

- Documents can be stored and retrieved from file (new function: VpeOpenDocFile())
- File formats are fully compatible between 16- and 32-bit versions
- Fast memory swapping algorithm, so that VPE needs only minimum RAM regardless of document size

Bugfixes / Workarounds:

- Workaround for bug in Win95 Laserjet drivers (unidrv.dll) - GPF when printing is now gone
- VpePrint(Box) now works correctly
- Fixed positioning of letters (too large gap)
- In special cases, justified layouting was inaccurate
- Fixed print garbage which occurred on some printers

Interfaces and Demos for:

- C/C++
- SQLWindows
- Visual Basic 4.0 (16 and 32 bit) - also usable for MS Access
- Delphi
- Borland Pascal

3 Introduction

VPE is a tool for designing and generating printouts and presentations under Windows. It comes as a DLL which can be used from any programming language for Windows that supports standard DLL calls (such as C/C++, Pascal, Basic, SQLWindows, etc.). All outputs can be generated by using simple commands in a way like those old “print” commands in DOS.

With VPE you can fill out pre-printed forms exactly (or print complete forms, as shown in the demo) regardless of the connected printer, because all coordinates and sizes are given in 1/10 mm. This is also ideal for printing labels.

VPE ”speaks” English or German, depending on the language chosen in the control-panel.

Overview:

VPEngine is database independent since you feed it the needed data.
Open as many virtual documents as you like.
Use colors, lines, frames, boxes, barcodes, bitmaps, and - for sure - text.
Give all drawing coordinates in 1/10 mm.
Use all text-formatting features (left, right, centered, justified, bold, italic, underlined).
Print 100 virtual pages and move virtually to the first page to enter new data.
Don't worry about the printer, it's resolution, or printing-offset (this is the part of the page the printer cannot print on). Your document will look the same on every printer as much as technically possible.
Don't worry about previews and printing-dialogs. VPEngine does it for you.
Show the user a preview, let him make choices in your program, then rework the report. This gives you the possibility of INTERACTIVE printing.
Let the user zoom through the preview since it's true WYSIWYG-Vector-Graphics!
In fact, VPEngine renders all objects in a virtual high resolution and then transforms it to the specified device, be it the screen, a printer, a fax or whatsoever. This gives the best possible WYSIWYG results.
Using special, optimized algorithms (since 1993 under development), VPEngine is really FAST!

What future releases may bring:

- Templates for page-layouts (instead of a header and a footer, you can define complete pages to be pre-layouted after initiating a PageBreak()-command)
- Templates for tables and lists
- A designer program for simply laying-out page- and table-templates with the mouse
- Scale to gray for bitmaps
- Scaled printing
- Big pages printed in segments over several small pages
- Charts

4 Explanation of the demo ”vpedemo.exe”

When you start the demo, you see a dialog and a blank window, both created and owned by vpedemo.exe.

Capabilites + Precision
This demo shows text formatting features, drawing features, bitmap handling, form filling, and printing.
Important: the VPE-DLL ”docks” its view **inside** of the window owned by vpedemo.exe! This is very easily done by a few lines of C code!
With the mover-buttons in the dialog of vpedemo.exe, you can scroll through the document.
The button ”Background” shows how to print **without** showing a preview and no setup-dialog (default printer is used).
The window sends the VPE_HELP message to the calling application **instead** of showing the standard help dialog, so you see the message dialog "User requested help" on the screen.
Note, that the form on the last page is a gray-scale bitmap with 96 by 96 DPI resolution. So the printout of it isn’t very nice.

Speed + Tables
Here you can see how fast VPE builds a report with a size of about 110 pages:
First of all, you have to generate this pseudo-report. This is done by clicking on the button ”Generate Report”.
Then, a textfile with random data is generated (journal.rpt). Clicking on the ”Speed + Tables” button makes vpedemo.exe read the textfile line by line, interpreting it and instructing VPE how to build the report.
Since it is random data, the number of pages differs from 110-116 pages. **Note** that vpedemo prints how many pages were generated on the FIRST page of the report in the upper left corner. This is done by the virtual processing of the document, where you can move to every **(ANY???)** page at any time to draw on it. In this case the demo generates all pages and then jumps to the first page to draw the message.

Colors
There you can see a fixed scaled window. Also, the toolbar has only the print buttons and the status bar is deactivated. The user cannot close the document; it can only be closed through vpedemo.exe by pushing the ”Close” button.
If you print the page on a color printer, you will get a true-color result.

Report
This is just another pseudo-report, showing various colors. The mover-buttons in the toolbar are hidden.

5 Important Notes

-The **macro WINDOWS_DLL** has to be defined when using a C/C++ compiler before including the header files. Otherwise, your linker might produce errors.

-The text-output functions have several mechanisms to calculate widths and / or heights. All calculations need time. **The more VPE has to calculate, the slower it works**. Keep this in mind.

-VPE doesn't need EASYBAR.DLL (ezbar32.dll) until a barcode-function is called.

-VPE doesn't need DAVINCI.DLL (dav32.dll) until a picture-function is called.
-Also VPE can handle BMP-files (NOT: RLE) without DAVINCI.DLL (dav32.dll).

6 How To Use VPE

1) Open a virtual document with the function ”VpeOpenDoc()”
2) Use all possible output calls
3) Use ”VpePageBreak()” to generate new pages
4) Use ”VpePreviewDoc()” to show the preview to the user (there the user can zoom and scroll through the document freely)
5) Use ”VpePrintDoc()” to print the document
6) Close the document with ”VpeCloseDoc()”

Example for SQLWindows:
A good place for calling VpeCloseDoc () is when you are processing the SAM_Destroy message of the window that called VpeOpenDoc ().

Example:

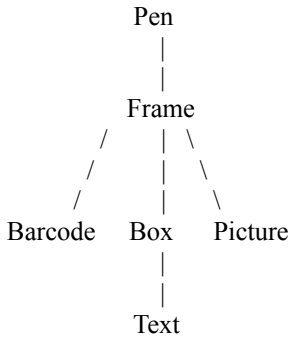
```
Number: hDoc

On SAM_CreateComplete
  Set hDoc = VpeOpenDoc( hWndForm, "Test", -1, -1, 0 )
  Call VpeLine( hDoc, 100, 100, 500, 500 )
  Call VpeOpenPreview( hDoc, 0, 0, 400, 400, VPE_SHOW_NORMAL )

On VPE_DestroyWindow
  Set hDoc = 0

on SAM_Destroy
  if hDoc != 0
    Call VpeCloseDoc( hDoc )
```

6.1 The Object-Oriented Style

VPE knows these objects:
1) Pen
2) Frame/Box
3) Barcode
4) Picture
3) Text

Attributes for one object inherit to the others.

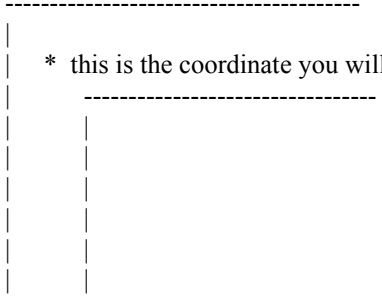
The inheritance order is:



The border of a frame is drawn with the size, style and color you use for the pen.
The borders of barcodes, boxes (filled frames) and bitmaps are drawn with the size, style and color you use for the pen.
A text(box) is drawn with the settings for the box (background fill color) and the pen. So if you set the pensize to 0, no frame is drawn.

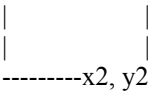
6.2 Important note about frames

In the current version of VPE, frames are drawn **around** the coordinates you give (this is needed in most cases).
For example: Imagine you have a 1cm bold frame. The frame will be drawn 0.5 cm to the left and 0.5cm to the right (also 0.5cm to the top and 0.5cm to the bottom) of the given coordinates.



6.3 Dynamic Positioning

Until version 1.2, objects could only be absolute positioned and sized. There were always two coordinate pairs x,y and x2,y2 to set the position of the top left corner and the bottom right corner:



Now the values for x2 and y2 may be negative and are then interpreted as width and height relative to the upper left corner. But x2 and y2 may NOT be in the range of -1 to -9; this is reserved for the V-flags (see below). But it's less than 1mm, so there shouldn't be a problem.

The central text output functions VpeWrite(Box), VpePrint(Box), and the VpePicture function are able to compute their object's height/height and width. But how large is the height/width? How can you position the next object relative to the last inserted?

The following constants will help:
VFREE: a flag for indicating, that VPE shall compute a width/height. It can only be used as y2 (height) for the VpeWrite(Box) function. This means that the coordinate shall be computed due to the text-length and font size.
VFREE can also be used for x2 (width) AND / OR y2 (height) for pictures. These coords will then be computed with the picture dimensions found in the file:

VLEFT: the x-coordinate of the last inserted object
VRIGHT: the x2-coordinate of the last inserted object
VTOP: the y-coordinate of the last inserted object
VBOTTOM: the y2-coordinate of the last inserted object

Example:
VpeWrite(hDoc, 100, 100, 600, VFREE, "long text.....");
This inserts a text object without frame at position 1cm, 1cm with a width of 5 cm. Its height (y2) is calculated and depends on the length of the text and the font size used.

VpeStorePos(hDoc);
This will store the coordinates x,y,x2,y2 of the last inserted object on a dynamic stack. This is limited only by memory in size.

VpeWrite(hDoc, VRIGHT, VTOP, -400, VFREE, "another text");
This inserts the next text object at position 6cm (the x2-coord. of the last inserted object), 1cm (the y2-coord. of the last inserted object), with a width of 4cm (height is calculated).

VpeRestorePos(hDoc);
This will now restore the last stored coords from the stack.

VpeWrite(hDoc, VLEFT, VBOTTOM, VRIGHT, VFREE, "another text2");
This inserts the next text object at position 1cm (the x-coord. of the restored coords), ?cm (the y2-coord. of the restored coords), with a width of 5cm (the y2-coord. of the restored coords), the height is calculated.

Also, VPE now knows an output rectangle on each page. This can be compared to the printable area or page-dimensions (if you wish).
-The output rectangle can be defined and retrieved.
-Each page in a document can have its individual output rectangle.
-After VpeOpenDoc() the default-rectangle is top = 200, left = 200, right = page_width - 200, bottom = page_height - 200
VPE_LANDSCAPE is automatically taken into consideration.

ATTENTION:
The output-rectangle is only for your own orientation purposes when positioning objects.
Also, VpePrint(Box) uses the output-rectangle to consider the maximum right border. You're still able to place objects outside the output-rectangle.

The following flags help consider the current output rectangle. They can be used in similar way to the other V-Flags (described above):
VLEFTMARGIN, VRIGHTMARGIN, VTOPMARGIN, VBOTTOMMARGIN

All V-Flags except VFREE can be used for ALL objects.
This also makes sense when developing a report. Imagine changing the width of one field in a table. Do you want to edit all coordinates of all other fields beside it???

What if you want to place an object relative to another, but with a gap between them?
You can retrieve and set all coords with the functions VpeGet(long hDoc, int what) and VpeSet(long hDoc, int what, int value).
The parameter is one of the V-Flags **except** VFREE.

Example:
VpeWrite(hDoc, VLEFT, VpeGet(hDoc, VBOTTOM) + 100, VRIGHT, VFREE, "another text3");

Inserts the next text object 1cm below the bottom border of the last inserted object.

6.4 Some Notes About Pictures in VPE

When displaying multiple bitmaps at the same time in a preview, VPE tries to balance the color-palette between these bitmaps. Sometimes this can lead to wrong color representation, but printing will always be ok.

Frames around bitmaps may make problems, because there is always the possibility of a one-pixel-error (see "WYSIWYG").
A gap may occur between the picture and the frame. If you use frames that are too fat, they will overlap the picture (see "Important Note About Frames"). This doesn't matter if you have pictures with a white background such as the logo.pcx, but the best solution is to have the frame in the image-files itself.

All metafiles (like WMF and DXF) are always converted to bitmaps in this version of VPE.
All metafiles (also those that are imported through Microsoft filters) always need the x2 and y2 parameters - if you leave them out, the image will not be processed. This is because vector-graphics have no fixed defined size.

The support of MS filters is an add-on feature in this version and not a true reliable embedded function.

Standard memory usage mechanism:

Definition of terms:
An **image** is the (uncompressed and sometimes huge) data in the file that is read into memory. A **DIB** is an image rendered to the device and another (sometimes huge) datablock.

On VpePicture() the whole image is read into memory and the the image-dimensions are calculated. Afterwards the image is immediately removed from memory.

If the picture has to be drawn (e.g. when the picture is displayed in the preview, or the image is printed) [LOOP], the image is read again into memory and then the DIB is rendered.

Afterwards the image-data is immediately removed from memory, since only the DIB is needed to be displayed.

The DIB is held in memory until the user moves to another page. Or printing continues on the next page.

When the actual output-device changes (printing with the open preview), the DIB is removed from memory and the things then continue at "LOOP".

This is the standard behaviour of VPE and a balance between execution speed and memory usage.

This leads to following implications:
A black-and-white bitmap of 1MB size will need 4 MB for a 16-color video display (4 bits per pixel). The same bitmap will need 8MB for a 256-color video display (8 bits per pixel). This changes incrementally as the bit-per-pixel variable increases. Keep this in mind.

See also the explanation of the VPE_PIC_xyz flags to change this standard behaviour (to have a faster processing and more memory usage or less memory usage and slower processing).

6.5 Messages Generated By VPE

6.5.1 VPE_DESTROYWINDOW

This is sent when the preview window was closed by the user (the document is also closed!):
LPARAM contains the document-handle, so you can determine which preview/document was closed (if you have more than one open).

6.5.2 VPE_PRINT

This is sent when the user starts printing, or when printing has ended successfully:
WPARAM:
1 = start of printing
0 = end of printing or user-abort

LPARAM contains the document-handle, so you can determine which preview prints (if you have more than one open).

6.5.3 VPE_PRINTCANCEL

This is sent when the user aborts the printer-setup dialog:
WPARAM:
0 = user-abort
LPARAM contains the document-handle, so you can determine which preview-print was cancelled (if you have more than one open).

VPE_PRINT and VPE_PRINTCANCEL can be processed as one and the same message. Just evaluate WPARAM to determine whether or not the print was aborted.

NEVER CALL VpeCloseDoc() IN RESPONSE TO VPE_PRINT OR VPE_PRINTCANCEL. YOU WILL TERMINATE A MODULE THAT IS STILL WORKING (OTHERWISE IT COULDN'T HAVE SENT ONE OF THE MESSAGES).

6.5.4 VPE_HELP

This is sent if the flag VPE_ROUTE_HELP was provided when calling VpeOpenDoc(), **and** when the user pressed the Help-Button in the toolbar or pushed the F1-key.
WPARAM: unused
LPARAM: unsused

6.6 WYSIWYG

WYSIWYG means "What you see is what you get;" in other words: "The output displayed on the screen will be the same as on the printer".
VPE is such a system. But due to some technical circumstances, there are limitations of these facilities.
The main point is that the resolution of the screen is rather poor in comparision to a printer. This leads to the possibility of a "usually-one-pixel-error". This means that a pixel difference is always
correct due to rounding problems in calculations. Sometimes it might be more than only one pixel, but be assured: VPE is a system that does its best to be WYSIWYG.
This chapter is only for people who want it very, very exact...

(Also some problems arise out of printer- or video-driver bugs. See "Trouble With Printer- and Video-Drivers").

6.7 Positioning On the Printer

The correctly positioned output depends on the mechanical capabilities of the printer. I don't know of any printer which can draw in a page with the exactness of 1/10mm. Depending on the quality of the printer, there are tolerances between 1 or 2mm (when the paper is misplaced in the y-direction.)

6.8 Embedded Flag-Setting

VPE knows many flags and settings for text-output. To reduce development time and the code-size of your EXE, most of the settings cannot be done only by calling functions, but by embedding them within the text you want to output.

For example:
You want to print the text ”Hello, World!” in bold and italic.
The string you would give to one of the text-output functions would be:
”[B I]Hello, World!”

The ”[” will only be interpreted at the very first position within the string. All following characters will be interpreted as flag-settings, until a ”]” is encountered.
A ”[[” sequence will be interpreted as one ”[” that will be printed!

The flags:
Each flag can be given in a long or a short form. Uppercase or lowercase-forms are not significant, i.e., ”NoHold” can also be written as ”nOhoLd”
Some flags must be followed by one or more numeric parameters.

The following fixed colors are defined and can be used as parameters for the color-flag settings:
"Black", "DkGray", "Gray", "LtGray", "White"
"DkRed", "Red", "LtRed"
"DkGreen", "Green", "LtGreen", "BlueGreen", "Olive"
"DkBlue", "Blue", "Cyan"
"DkPurple", "Purple", "Magenta"
"LtYellow"

”NoHold”, ”N”
If used, this flag must be used as the first in the sequence. This means that the setting is not permanently stored within the engine.
If you don’t use this flag, all further output-calls will use the settings you have specified.

"PenSize", ”PS” <pensize>
Sets the pensize. <pensize> is the size of the pen in 1/10mm

"PenColor", ”PC” <color-name>
Sets the pencolor. <color-name> is one of the color-strings above.

"PenColorRGB", ”PCRGB” <red> < green> <blue>
Sets the pencolor with RGB-values <red> < green> <blue>.
Example: ”PCRGB 200 210 30”

”” [it’s a single quote!]
Use the font specified within single quotes.
Example: ”['Arial']”

"FontSize", ”S” <fontsize>
Sets the fontsize. <fontsize> is the size of the font in points (NOT 1/10mm!).

"Color", ”C” <color-name>
Sets the text color. <color-name> is one of the color-strings above.

"ColorRGB", ”RGB” <red> < green> <blue>
Sets the text color with RGB-values of <red> < green> <blue>.

"BkgColor", ”BC” <color-name>
Sets the background color. <color-name> is one of the color-strings above.

"BkgColorRGB", ”BCRGB” <red> < green> <blue>
Sets the background color with RGB-values of <red> < green> <blue>.

"Transparent", ”T”
Sets the transparent-mode to ON.

"TransparentOff", ”TO”
Sets the transparent-mode to OFF.

"Justified", ”J”
Sets the text alignment to justified.

"Right", ”R”
Sets the text alignment to right.

"Left", ”L”
Sets the text alignment to left.

"Center", ”CE”
Sets the text alignment to centered.

"Bold", ”B”
Sets the font setting to bold.

"BoldOff", ”BO”
Sets the font setting to bold off.

"Underline", ”U”
Sets the font setting to underlined.

"UnderlineOff", ”UO”
Sets the font setting to underlined off.

"Italic", ”I”
Sets the font setting to italic.

"ItalicOff", ”IO”
Sets the font setting to italic off.

6.9 Predefined Colors In the Header File ”colors.h”

```
#define COLOR_BLACK                 RGB(0, 0, 0)
#define COLOR_DKGRAY        RGB(128, 128, 128)
#define COLOR_GRAY                  RGB(192, 192, 192)
#define COLOR_LTGRAY        RGB(230, 230, 230)
#define COLOR_WHITE                 RGB(255, 255, 255)
#define COLOR_DKRED                 RGB(128, 0, 0)
#define COLOR_RED                   RGB(192, 0, 0)
#define COLOR_LTRED                 RGB(255, 0, 0)
#define COLOR_DKGREEN       RGB(0, 128, 0)
#define COLOR_GREEN                 RGB(0, 192, 0)
#define COLOR_LTGREEN       RGB(0, 255, 0)
#define COLOR_BLUEGREEN     RGB(0, 128, 128)
#define COLOR_OLIVE                 RGB(128, 128, 0)
#define COLOR_DKBLUE        RGB(0, 0, 128)
#define COLOR_BLUE                  RGB(0, 0, 255)
#define COLOR_CYAN                  RGB(0, 255, 255)
#define COLOR_DKPURPLE      RGB(128, 0, 128)
#define COLOR_PURPLE                RGB(192, 0, 192)
#define COLOR_MAGENTA       RGB(255, 0, 255)
#define COLOR_LTYELLOW      RGB(255, 255, 0)
```

For the screen, VPE always takes the nearest solid color. On all other output devices, the color will be what you have chosen.

6.10 Trouble With Printer- and Video-Drivers

Due to some printer- and video-drivers, the WYSIWYG capabilities of VPE are in some cases distorted.

6.10.1 Printers

Current Win95 HP Laserjet Drivers are slow and work with problems. Please report bugs regarding Win95 HP Laserjet Drivers to Microsoft or HP.

Problems with clipping:
HP drivers cause problems with text clipping. Some HP drivers are not able to clip letters, so instead they print the whole letter.
The y2 border of VpeWrite(Box) might be crossed by letters, which should be printed only in part.
You might experience this problem when using VpeWrite(Box) with a fixed y2 coordinate. If y2 has a value (the last line of the text is printed in part on the screen), it might be printed completely on the printer.
The solution is to set y2 smaller, so that the whole line is clipped.

Some printer drivers may not print colored text, printing nothing instead.

6.10.2 Some video card drivers do not work correctly. Symptoms are:

- layout incorrect (wrong positioning of text and lines)
- the moving markers in the rulers might be drawn incorrectly
- colors of bitmaps are shown incorrectly
- bitmaps are not shown when color-resolution of video-adapter is higher than the bitmap-resolution (i.e., video-adapter = true-color, bitmap=256 colors)
- in multiple-colored bitmaps, text colors are damaged and texts not drawn/half drawn/drawn incorrectly
- driver crashes when using bitmaps
- driver may crash under other circumstances

Discuss bugs with the video-card manufacturer, unless you also experience these problems with the standard VGA driver (in which case VPE is the problem).

7 Functions (In Functional Order)

7.1 Management Functions

7.1.1 int VpeGetVersion()

Action: return the current version number coded as follows

Parameters: none

Returns: the current version number coded as follows
Hi-Byte = major no. (0-99) / Lo-Byte = minor no. (0-99)

for example: 0x0115 = 1.21

7.1.2 long VpeOpenDoc(HWND hwndParent, char *title, int page_width, int page_height, long flags)

Action: Prepare the VPE for a new document

Parameters:
HWND hwndParent a window of the calling application VPE exchanges messages with
and the window where VPE connects its preview to, if embedded
char *title title of the windows
int page_width the width of the document in 1/10mm
int page_height the height of the document in 1/10mm
long flags style of preview and behaviour of VPE

The page size is free defineable:
-16-Bit version: up to 138cm x 138cm (for 600 DPI Printer); 69cm x 69cm for 1200 DPI; etc.
-32-Bit version: 999cm x 999cm
If page_width = -1, standard DIN A 4 size (21cm x 29.7 cm) is taken.
If page_width = -2, standard US Letter size (8.5 x 11 inch) is taken.
Don't worry if VPE_LANDSCAPE is not set!
Parameters for a page of 30cm width and 50cm height are:
page_width=3000 and page_height=5000 (VPE internally rotates the paper).

Setting the page size will not affect the printer's page size (this may be addressed in a later release).
Note: 1 inch is 2.54cm, so 5 inch = 5 * 2.54 cm = 12.7cm

You don't have to worry about the printer. Resolution is only important for the page dimension, since with 16-bit number representation there are limits for the maximum positions. The positioning itself is done with 1/10mm coordinates; they are internally computed into point coordinates. Those computations may not exceed the number-range of an 16-bit integer, so a page may not be bigger than 138cmx138cm for 600 DPI and 69x69cm for 1200 DPI, etc. Using a size of US-standard letter or DIN A 4 will never give you trouble with the printer resolution, unless the printer has **MORE** than 2400 DPI.

Each printer has a printing-offset. This is the gap at the top and at the left on the page, where the printer cannot print. VPE takes care of this, so that the positioned text will always be at the same position. (If you're printing too close to the top border, the text might not be printed, but the layout on every printer is exactly the same (this is ideal for printing labels or forms)). If the printer-driver is bad, this offset might be wrong (discuss this with the vendor). Currently there is no driver known to have such problems.

The flags:

There are many flags with VPE, but we want to give you most control possible.
In standard cases you will just give zero as parameter.

```
VPE_GRID_INBACKGROUND       1     // grid in background
VPE_GRID_INFOREGROUND       2     // grid in foreground
VPE_GRID_TOOLBARBUTTON      4     // grid toolbar-button visible
VPE_GRID_VISIBLE            8     // grid on open visible
VPE_NO_RULER                16    // ruler NOT visible
VPE_NO_TOOLBAR              32    // toolbar NOT visible
VPE_NO_USER_CLOSE           64    // user cannot close
                                  the preview (only the application)

                                  // stop-button invisble and sys-menu

                                  // disabled (if not embedded doc)

                                  // VpeCloseDoc() works
VPE_NO_USER_MOVE            128   // user cannot leaf through
                                  the doc
VPE_NO_MOUSE_SCALE          256   // user cannot scale
                                  with the mouse-buttons  (but
                                  VpeSetScale() works)
VPE_NO_USER_SCALE           768   // user can not scale (but
                                  VpeSetScale() works)
VPE_NO_STATBAR              1024  // statusbar invisible
VPE_NO_PRINTBUTTON          2048  // print-button
                                  invisible (but VpePrintDoc()
                                  works
VPE_EMBEDDED                4096  // document
                                  window is embedded within a
                                  window of the calling application
VPE_LANDSCAPE               8192  // document will
                                  be printed in landscape-format
VPE_NO_HELPBTN              32768 // Help-Button invisible
VPE_ROUTE_HELP              65536 // if Help-Button
                                  visible, pressing this or pushing F1
                                  (will cause the message
                                  VPE_HELP to be send to the
                                  owner-window)
VPE_NO_INFOBTN              131078          // Info-
                                  Button invisible
```

Most of the flags can be combined using the OR operator (or simply the "+" operation).
Here some predefined combinations:

```
VPE_GRID_POSSIBLE   (VPE_GRID_INFOREGROUND |
                    VPE_GRID_TOOLBARBUTTON)
VPE_GRID_ON         (VPE_GRID_INFOREGROUND |
                    VPE_GRID_TOOLBARBUTTON |
                        VPE_GRID_VISIBLE)
VPE_GRID_BKGON      (VPE_GRID_INBACKGROUND |
                    VPE_GRID_TOOLBARBUTTON | VPE_GRID_VISIBLE)
VPE_GRID_OFF        0
```

A preview window can be **embedded** into the window of the calling application. This means that VPE does not open its own window, but draws its preview into the caller's window.

To do this, you just need to:
1) Use the flag VPE_EMBEDDED
2) In the window-procedure of the window where the preview shall be embedded, give the following command to the WM_SIZE message:
MoveWindow(VpeWindowHandle(hDoc), 0, 0, LOWORD(lParam), HIWORD(lParam), FALSE)
- this will size the VPE preview window correctly.
3) In the window-procedure of the window where the preview shall be embedded, give the followint command to the WM_KEYDOWN message:
SendMessage(VpeWindowHandle(Precision), WM_KEYDOWN, wParam, lParam) - this will have the keyboard messages routed to the VPE preview window.

CAUTION:
Using VPE from interpreters can cause some trouble when stopping program execution without a prior call to VpeCloseDoc().
Some interpreters will not unload VPE, so that the document stays open. In this case, the memory used by VPE isn't released to the system, and in some cases GPFs might occur.

Returns:
The handle (=identifier) to the virtual document.
This handle has to be provided to all other VPE calls.

7.1.3 long VpeOpenDocFile(HWND hWndParent, LPSTR file_name, char *title, int page_width, int page_height, long flags)

Action: Same as VpeOpenDoc(), but instead of storing all document pages in RAM, only the actual page is held in RAM. All other pages are swapped to file. This implies minimum RAM usage at a very high performance. Also you can store and retrieve a document to/from the file and later add new pages to the document.

Understanding this mechanism:
In contrast to a full memory document, you can't modify a page after it has been swapped to file. But you can still add pages at the end of the document. A page is swapped after:
1) calling VpePageBreak()
2) after calling VpeGotoPage()

It is only swapped if it has not yet been swapped. So you can't modify it after it has been swapped once. New inserted objects will appear on the screen, but they are not stored in the file.

You can also retrieve a stored file, because VPE looks first to see if the file already exists. If so, its first page is loaded into RAM.
So if you want to add pages at the end of the document, you have to call VpePageBreak() first, so that a new empty page is added at the end of the document . If the document doesn't contain any pages, then the first page is the empty page.

The 16- and 32-bit file formats are 100% compatible.

Parameters: Same as VpeOpenDoc(), file_name is the name of the swapper-file

Returns:
The handle (=identifier) to the virtual document.
This handle has to be provided to all other VPE calls.

7.1.4 int VpeCloseDoc(long hDoc)

Action: Closes the specified document (and preview, if open)

Parameters: Document handle

Returns:
TRUE: ok
FALSE: couldn't close, because the document is currently printed

7.1.5 void VpePreviewDoc(long hDoc, RECT *rc, int show_hide)

Action: Opens the preview window.

Parameters: Document handle, position of the preview window in rc, show_hide:one of the flags below
If rc is NULL, it is ignored. For use with interpreter languages, you can set rc.right = -1; then it is also ignored.

show_hide Flag Parameter:

```
enum
{
  VPE_SHOW_NORMAL = 1,
  VPE_SHOW_MAXIMIZED,
  VPE_SHOW_HIDE,
};
```

Returns: -

7.1.6 HWND VpeWindowHandle(long hDoc)

Action: Returns the window handle of the preview.

Parameters: Document handle

Returns: The window handle of the preview

7.1.7 void VpePageBreak(long hDoc)

Action: Adds a new blank page to the end of the document; all further output-calls will draw onto this new page.

Parameters: Document handle

Returns: -

7.1.8 int VpeGetPageCount(long hDoc)

Action: ...

Parameters: Document handle

Returns: The page count

7.1.9 int VpeGetCurrentPage(long hDoc)

Action: ...

Parameters: Document handle

Returns: The page-number of the current active page

7.1.10 void VpeGotoPage(long hDoc, int page)

Action: Goes to the specified page; all further output-calls will draw onto this page.
If the preview is open, this will also affect the preview (see VpeSetUpdate()).

Parameters: Document handle, page number

Returns: -

7.1.11 void VpeStoreSet(long hDoc, int id)

Action: All current flag-settings (pen-size, alignment, colors, font, etc.) are stored.
This is useful if you need them again later.

Parameters: Document handle, id = a number you specify

Returns: -

7.1.12 void VpeUseSet(long hDoc, int id)

Action: This sets all flags to the stored values.

Parameters: Document handle, id: the id under which you stored the flags

Returns: -

7.1.13 void VpeRemoveSet(long hDoc, int id)

Action: Removes the in id specified setting from memory.

Parameters: Document handle, id: the id under which you stored the flags

Returns: -

7.1.14 void VpeSetRulersMeasure(long hDoc, int rulers_measure)

Action: Sets the measurement for the rulers. This does not affect the internal coordinate system! All coordinates must always be in 1/10mm.

Parameters: Document handle, rulers_measure: 0 = cm, 1 = inch

Returns: -

7.1.15 void VpeSetScale(long hDoc, double scale)

Action: Sets the scale for the preview (**not for printing -** printing cannot be scaled in the current version).

For example:
1.0 is 1:1,
0.25 is 1:4
4.0 is 4:1

Parameters: Document handle

Returns: -

7.1.16 void VpeSetUpdate(long hDoc, int yes_no)

Action: If the preview is open, this flag determines whether all drawing actions of your application are directly reflected and made visible in the preview. USE THIS FLAG WITH CAUTION!!!
If you have too many output calls, the system will slow down. The default is NO (no immediate update).

Parameters: Document handle, -

Returns: -

7.1.17 void VpeRefreshDoc(long hDoc)

Action: When the preview is open, this call will refresh the view and make all changes to the document in the current page visible. If the user scrolls a page, all changes to the document will be visible automatically.

Parameters: Document handle

Returns: -

7.2 Printing Functions

7.2.1 int VpeSetupPrinter(long hDoc, LPSTR file_name, int dialog_control)

Action: You can make a setup for the printer separate from printing. The settings can be stored/retrieved in/from the file (optional). Otherwise, the setting is only remembered for the lifetime of the document.
This function gives you the possibility to save settings not only for your application, but for every individual document type.
This will give you the most control possible: You can implement a "Printer Setup" in your applications menu. Use dialog_control=2 in this case. Before printing, call SetupPrinter() with dialog_control=1 and specify a file-name. If your user hasn't done the setup (the specified file is not found), it will be done then and only once! VPE will save ALL settings: Printer, Driver, Port, and ALL other driver-specific stuff - **always** to the specified file after a successful setup.

Parameters:

hDoc	document handle
file_name	(path and) file name of the file where to restore or store the setup or NULL
dialog_control	0 = never show setup-dialog (if file_name is NULL, the last setting or the setting of the default-printer will be taken) 1 = show setup-dialog only, if file-read fails 2 = show setup-dialog always

Returns:

0 = Ok
1 = User pushed cancel button in dialog
2 = I/O problems during read (only if dialog_control = 0)
3 = I/O problems during write

7.2.2 void VpePrintDoc(long hDoc, BOOL with_setup)

Action: Prints the document. You must not close your application or the document **until** VPE finishes all print-jobs. (See messages VPE_PRINT / VPE_PRINTCANCEL in "Messages Generated By VPE," and also Function VpeIsPrinting()).
Your application code will hold at the VpePrintDoc() call as long as all pages are printed. But your application will still be able to receive Windows Messages. So it is your responsibility to disable your application, or to take care that your print-functions are stopped from being reentered (see vpedemo.cpp).

Parameters: Document handle / show setup-dialog before

Returns: -

7.2.3 int VpeIsPrinting(long hDoc)

Action: If your program, or the user, started printing (by clicking the toolbar button), this function will return TRUE. While printing, the document may not be modified, nor closed. VPE will ignore all function calls with this document handle.

Parameters: Document handle

Returns: TRUE = is printing, else FALSE

7.3 Layout Functions

7.3.1 int VpeGet(long hDoc, int what)

Action: Returns the coordinate specified by one of the V-flags provided in parameter "what" (see "Dynamic Positioning").

Parameters: Document handle, V-flag

Returns: The coordinate specified by parameter "what"

7.3.2 void VpeSet(long hDoc, int what, int value)

Action: Sets the coordinate specified by one of the V-flags provided in parameter "what" (see "Dynamic Positioning").

Parameters: Document handle, V-flag, new value of the coordinate

Returns: -

7.3.3 void VpeSetDefaultOutputRect(long hDoc, RECT *r)

Action:
Sets the default output rectangle (will be used after each page break).

Parameters: Document handle, output rectangle

Returns: -

7.3.4 void VpeSetOutputRect(long hDoc, RECT *r)

Action: Sets the output rectangle for the current active page.

Parameters: Document handle, output rectangle

Returns: -

7.3.5 void VpeGetOutputRect(long hDoc, RECT *r)

Action:
Gets the output rectangle for the current active page

Parameters: document handle, output rectangle

Returns: -

7.3.6 void VpeSetPosRect(long hDoc, RECT *r)

Action: Sets a dummy position as the last inserted object. All calls with V-flag parameters will take these values.

Parameters: Document handle, dummy position rectangle

Returns: -

7.3.7 void VpeGetPosRect(long hDoc, RECT *r)

Action: Retrieves the rectangle of the last inserted object.

Parameters: Document handle, rectangle

Returns: -

7.3.8 void VpeStorePos(long hDoc)

Action: Stores the coordinates x,y,x2,y2 of the last inserted object on a dynamic stack (limited only by memory in size).

Parameters: Document handle

Returns: -

7.3.9 void VpeRestorePos(long hDoc)

Action: Restores the last stored coordinates x,y,x2,y2 from the stack.

Parameters: Document handle

Returns: -

7.4 Drawing Functions

7.4.1 void VpeSetPen(long hDoc, int pen_size, int pen_style, COLORREF color)

Action: Sets the style of the pen. The default is 0.3 mm, PS_SOLID, black.

You can use the PS_xyz pen styles from Windows GDI, but pen styles other than PS_SOLID are limited from the GDI to pens with a size of 1 pixel! So you should always use PS_SOLID, until the GDI changes.

Parameters: Document handle, pen_size in 1/10mm; pen_style = one of the windows pen styles (PS_SOLID, etc.),
pen_color = RGB(x, y, z)

Returns: -

7.4.2 void VpeNoPen(long hDoc)

Action: Hides the pen (pensize is internally set to zero). All drawing objects (functions) that are inherited from the pen-object will use no pen.

Parameters: Document handle

Returns: -

7.4.3 void VpeLine(long hDoc, int x, int y, int x2, int y2)

Action: Draws a line with the current pen from x, y to x2, y2.

Parameters: Document handle

Returns: -

7.4.4 long VpePolyLine(long hDoc, POINT *p, unsigned int size)

Action: Draws a poly-line with the current pen.

Parameters: Document handle, Array of POINT-structures that contain the coordinate pair where the line is to be drawn, the size of the array (count of elements, NOT bytes).

You can set parameter <p> to NULL (zero). This instructs VPE to create an empty array of <size> elements, ready prepared for use with the function VpeAddPolyPoint(). Some interfaces only support setting <p> to null (there, p is declared as long, as in Visual Basic).

Structure of p:
-The first element contains the starting coordinate
-Each other element contains the next coordinate where to draw to
-If an element is -1, -1, the next coordinate is interpreted as a NEW starting coordinate

Returns: A handle to the object (this can be used in further calls to VpeAddPolyPoint())

7.4.5 void VpeAddPolyPoint(long hDoc, long p, int x, int y)

Action: Adds a new point to the polyline-object.

Parameters: Document handle, polyline-object-handle, x, y

Returns: -

7.4.6 void VpeSetBkgColor(long hDoc, COLORREF color)

Action: Sets the background color. The background color is the color inside of a box.
The default is WHITE (but transparent mode is activated!).

Parameters: Document handle, color = RGB(x ,y ,z)

Returns: -

7.4.7 void VpeSetTransparentMode(long hDoc, int on_off)

Action: Sets background color to transparent/not transparent. The background color is the color inside of a box.
The default is ON.

Parameters: Document handle, -

Returns: -

7.4.8 void VpeBox(long hDoc, int x, int y, int x2, int y2)

Action: Draws a box object at position x, y with the right border at x2 and the bottom border at y2. Penstyle and background color are used.

Parameters: Document handle, coordinates of the box

Returns: -

7.5 Text Functions

7.5.1 void VpeSelectFont(long hDoc, char *name, int size)

Action: Selects a font and its size. The default is ”Arial” and 10 pt. You can only select True-Type fonts installed on the machine on which VPE is running.

Parameters: Document handle, font-name, size in points **(NOT 1/10mm!)**

Returns: -

7.5.2 void VpeSetFontAttr(long hDoc, int alignment, int bold, int underlined, int italic)

Action: Sets the font-attributes.
The default is: ALIGN_RIGHT, FALSE, FALSE, FALSE,

Parameters: document handle,

alignment:
enum
{
	ALIGN_LEFT,
	ALIGN_RIGHT,
	ALIGN_CENTER,
	ALIGN_JUSTIFIED,
	ALIGN_PRINT, // like ALIGN_LEFT, X2 and Y2 are computed -
	ONLY for internal usage: **DO NOT USE**
};

Returns: -

7.5.3 void VpeSetAlign(long hDoc, int alignment)

Action: ...

Parameters: Document handle, ...

Returns: -

7.5.4 void VpeSetBold(long hDoc, int bold)

Action: ...

Parameters: Document handle, ...

Returns: -

7.5.5 void VpeSetUnderlined(long hDoc, int underlined)

Action: ...

Parameters: Document handle, ...

Returns: -

7.5.6 void VpeSetItalic(long hDoc, int italic)

Action: ...

Parameters: Document handle, ...

Returns: -

7.5.7 void VpeSetTextColor(long hDoc, COLORREF color)

Action: ...

Parameters: Document handle, ...

Returns: -

7.5.8 int VpeWrite(long hDoc, int x, int y, int x2, int y2, char *s)

Action: Outputs text formatted with the current alignment settings within a rectangle at position x, y, with the right border at x2 and the bottom border at y2. The pen is invisible; the background color is always transparent.
If you specify VFREE (-1) for y2, it will be calculated.

Parameters: Document handle, position and dimensions, the string to output

Returns: The bottom y-coordinate generated by the output

7.5.9 int VpeWriteBox(long hDoc, int x, int y, int x2, int y2, char *s)

Action: Same as VpeWrite, but pen- and box-settings are used. If you specify VFREE (-1) for y2, it will be calculated.

Parameters: Document handle, position and dimensions, the string to output

Returns: The bottom y-coordinate generated by the output

7.5.10 int VpePrint(long hDoc, int x, int y, char *s)

Action: Like VpeWrite, but you need no box-coordinates. The pen is always off and the background mode is transparent. The alignment is ALIGN_PRINT (i.e. ALIGN_LEFT), if the right border of the page (VRIGHTMARGIN, the x2 coordinate of the Ouput Rectangle) is reached, the text is automatically broken to the next line; the new starting coordinate is then again x. This function does a lot of calculations and is time-consuming in relation to VpeWrite or VpeWriteBox.

Parameters: Document handle, position, text

Returns: The bottom y-coordinate generated by the output

7.5.11 int VpePrintBox(long hDoc, int x, int y, char *s)

Action: Like VpePrint, but pen- and box-settings are used.

Parameters: Document handle, position, text

Returns: The bottom y-coordinate generated by the output

7.5.12 void VpeDefineHeader(long hDoc, int x, int y, int x2, int y2, char *s)

Action: Exactly the same as VpeWriteBox, but the string is outputted automatically on each new page.
The String may contain the sequence ”@PAGE” which will insert the current page-numer.

Parameters: Document handle, position and dimensions, the string to output

Returns: -

7.5.13 void VpeDefineFooter(long hDoc, int x, int y, int x2, int y2, char *s)

Action: Exactly the same as VpeWriteBox, but the string is outputted automatically on each new page.
The String may contain the sequence ”@PAGE” which will insert the current page-number.

Parameters: Document handle, position and dimensions, the string to output

Returns: -

7.6 Picture Functions

7.6.1 void VpeKeepBitmapAspect(long hDoc, int on_off)

Action: Setting this mode on/off determines whether a scaled bitmap shall not be distorted when the x OR y dimension is calculated automatically. This makes sense if only ONE parameter is -1: x2 OR y2.
The default is: ON

Parameters: Document handle, -

Returns: -

7.6.2 void VpeDefaultBitmapDPI(long hDoc, int dpix, int dpiy)

Action: Some bitmap-files do not contain what DPI resolution in which they were originally created. For example, GIF images have no such information (use 96 by 96).
Since VPE is a WYSIWYG-system, it needs this information for correct representation of the bitmap.
The default is 96 by 96 DPI which is the resolution of the screen.

Parameters: Document handle, -

Returns: -

7.6.3 LPCSTR VpeGetPictureTypes()

Action: Returns a string with the extensions of all supported file-formats.

Parameters: -

Returns: A pointer to a string. The string should be copied. For example:
"*.WMF;*.BMP;*.TIF;*.JPG;*.PCX;*.TIF"

7.6.4 void VpePicture(long hDoc, int x, int y, int x2, int y2, LPSTR file_name, int flags)

Action: Creates a picture-object. Currently VPE supports the following formats:
-Windows and OS/2 Bitmaps (2/16/256/True Color)
-Windows WMF (Metafile)
-AutoCAD DXF (basic format)
-GIF (2/16/256 Color)
-PCX (2/16/256 Color)
-JPG (256/True Color)
-TIFF 5.0 (2/16/256/True Color, LZW/PackBits/Fax G3/Fax G4/Tiled Images)
-ALL installed Adobe / Microsoft filters (for example, those who come with Word for Windows - only 16-Bit Windows)

Parameters: Document handle, position and dimensions, the file-name, some flags

Dimensions:
if x2 is VFREE, it is calculated automatically; if y2 is VFREE, it is calculated automatically

File-name: VPE determines the file-type by the suffix (i.e. TIF = TIFF, etc.) - including a full path is optional

Flags:
An **image** is the (uncompressed and sometimes huge) data in the file that is read into memory. A **DIB** is an image rendered to the device and another (sometimes huge) datablock.

#define VPE_PIC_MERGE 1
Merge background with bitmap (SRCAND instead of SRCCOPY).
This flag is only useful, if you have things already drawn and want to merge this with the image. Normally you put the image first onto the page, and text and so on afterwards. So you don't need this flag. (slows down processing time)

The following KEEP-Flags are only relevant when working with a preview.

#define VPE_PIC_KEEPIMAGE 2
With this flag the turn-around between displaying and printing will speed up, because the image-data is always held in memory.
It is also useful, when having only one page (or multiple pages with small bitmaps, or enough RAM) in an open preview, else VPE reads the image-data twice (for getting the dimensions and then for displaying).

#define VPE_PIC_DISCARD_DIB_DRAW 4
VPE holds in a preview all DIBs in memory until the user moves to another page. Using this flag will discard the DIB immediately after drawing from memory. Useful, if you have large page dimensions and many pictures on it. Or if you need as much free ram as possible.

#define VPE_PIC_KEEP_DIB_PAGE 8
VPE discards in a preview all DIBs from memory when the user moves to another page. Using this flag will stop this mechanism. The DIB is always held in memory. It also overrides the VPE_PIC_DISCARD_DIB_DRAW flag.

Returns: -

7.7 Barcode Functions

7.7.1 void VpeSetBarcodeParms(long hDoc, int orientation, BYTE top_bottom, BYTE add_top_bottom)

Action: Specifies the orientation of the barcode, the position of the text and the position of the add-on text (if add-on barcode).

Parameters:
-document handle
-orientation in 90 degree steps (0 / 90 / 180 / 270) - other values have no effect
-top_bottom: text on top (1) or on bottom of barcode (0)
-add_top_bottom: add-on text on top (1) or on bottom of barcode (0)

Returns: -

7.7.2 void VpeBarcode(long hDoc, int x, int y, int x2, int y2, int code_type, LPSTR code, LPSTR add_code)

Action: Generates and draws a barcode within a rectangle at position x, y, with the right border at x2 and the bottom border at y2.
VPE doesn't enforce the absolute size of the barcode (left to the responsibility of the caller), but it does enforce the exactness of the relative widths of the bars at the output device's pixel level (other than screen). Consider a barcode consisting of 1 bar, 1 space and 1 bar, with width ratios 3:1:2. The minimum width of the barcode is 6 pixels, other possible sizes are 12, 18 and so on. If you give a rectangle where only 5 pixels might fit, the barcode will still occupy 6 pixels. Don't make the rectangle too small. Normally the barcodes will be drawn inside the rectangle and as big as possible due to the exactness of the relative widths of the bars.

Parameters:
-document handle
-position and dimensions
-code_type, one of the following constants / values (defined in vpecomon.h):

#define BCT_EAN13	1
#define BCT_EAN8	2
#define BCT_UPCA	3
#define BCT_CODABAR	5
#define BCT_CODE39	6
#define BCT_2OF5	7
#define BCT_INTERLEAVED2OF5	8
#define BCT_UPCE	9
#define BCT_EAN13_2	10
#define BCT_EAN13_5	11
#define BCT_EAN8_2	12
#define BCT_EAN8_5	13
#define BCT_UPCA_2	14
#define BCT_UPCA_5	15
#define BCT_UPCE_2	16
#define BCT_UPCE_5	17
#define BCT_EAN128A	18
#define BCT_EAN128B	19
#define BCT_EAN128C	20
#define BCT_CODE93	21
#define BCT_POSTNET	22

-string with the code (i.e. "123456")
-string with the add-on code, if add-on barcode selected, else NULL (0)

Returns: -

8 Release Notes

8.1 Release 1.1

-Now supports 21 barcode types (VPENGINE.DLL doesn't need EASYBAR.DLL until a barcode-function is called).

Bugfixes / Workarounds:
-Implemented a workaround for bug in Win95 Laserjet drivers (unidrv.dll) - GPF when printing is now gone.

Trademarks

True Type is a registered trademark of Apple Computer, Inc.
SQLWindows, Centura and Gupta are registered trademarks of Gupta Corporation.
Hewlett-Packard and Laserjet are registered trademarks of Hewlett-Packard Company.
Delphi, Borland Pascal and Borland are registered trademarks of Borland International.
Microsoft, Visual Basic, Visual FoxPro, Microsoft Access, and Windows are registered trademarks of Microsoft Corporation.
All other trademarks and service marks are the property of their respective owners.