

Programmer's Manual

Virtual Print Engine v2.0

Copyright © 1995 - 1997 by IDEAL Software.

All rights reserved.

IDEAL Software

Grefrather Weg 96

41464 Neuss

Germany

Phone: (+49) (0)2131 / 9800-23

Fax: (+49) (0)2131 / 9800-24

E-Mail: Support@IdealSoftware.com

Web: www.IdealSoftware.com

Your registration code:



Contents

1 General

1.1 Limited Warranty and License Agreement

The following paragraph is not related to the common distributable files. It is related to the use of VPE by software developers:

This software is protected by copyright law and international copyright treaty. Therefore, you must treat this software just like a book, except that you may copy it onto a computer to be used and you may make archive copies of the software for the sole purpose of backing up our software and protecting your investment from loss.

By saying "just like a book", IDEAL Software means, for example, that this software may be used by any number of developers, and may be freely moved from one computer or location to another, so long as there is no possibility of it being used by more than one developer at a time. Just as a book can't be read by two different people in two different places at the same time, neither can the software be available for development purposes by two different people in two different places at the same time without IDEAL Software's permission (unless, of course, IDEAL Software's copyright has been violated). So you need to order VPE per each developer using it.

Adding Users (Software-Developers)

You may add users by paying for a separate software package for each user you wish to add. You may also add users by purchasing LAN packs/LAN licenses (for use through a local area network only), multi-user versions, or additional licenses for the software, so long as the number of persons who are able to use the software at one time isn't more than the number of authorized users specified in our package or license (otherwise, you are not using the software "just like a book"). The number of authorized users is not increased if IDEAL Software simply puts more than one type of media (such as 3.5- and 5.25-inch diskettes) in the software package.

Transferring the Software

You may transfer all of your rights to use the software to another person, provided that you transfer to that person all of the software, diskettes, and documentation provided in this package (including this statement), and transfer or destroy all copies in any form. Remember, once you transfer the software you no longer have any right to use it, and the person to whom it is transferred may use it only in accordance with the copyright law, international treaty, and this statement.

If you have purchased an upgrade version of the software, it constitutes a single product with the "IDEAL Software" software that you upgraded. For example, the upgrade and the software that you upgraded cannot both be available for use by two different people at the same time, and cannot be transferred separately, without written permission from IDEAL Software.

Except as provided in this statement, you may not transfer, rent, lease, lend, copy, modify, translate, sublicense, time-share, or electronically transmit or receive the software, media, or documentation.

License Agreement - Distributable Files

If you bought the unlicensed version, you may not distribute any files outside of your company. Even for the use in different office branches within one company you need to order the full licensed version.

Depending on the license and platform support you have bought, the only distributable files are:

16 Bit:	32-Bit:
VPENGINE.DLL	VPE32.DLL
VPECTL16.OCX	VPECTRL.OCX
EASYBAR.DLL	EZBAR32.DLL
DAVINCI.DLL	DAV32.DLL
LEON.DLL	LEON32.DLL

Note on DAVINCI.DLL (DAV32.DLL) and LEON.DLL (LEON32.DLL):

Copyright © by Ingenieurbuero Herd & Nuding, they may only be used / distributed in conjunction with VPE. You may not use them outside of VPE. If you have such needs, contact:

Ingenieurbuero Herd & Nuding; Niederstr. 36; 64285 Darmstadt; Germany
Tel. 06151-664717 * Fax. 06151-664740 * BBS 06161-664741

Note on EASYBAR.DLL (EZBAR32.DLL):

Copyright © by Bokai Corporation, they may only be used / distributed in conjunction with VPE. You may not use them outside of VPE. If you have such needs, contact:

Bokai Corporation; 1221 Dundix Rd., #106; Mississauga, ON L4Y 3Y9; Canada; Fax: (+1) 905 276-7692

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, Database-Engines, any possible combinations of these product categories, 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 IDEAL Software for special conditions. Your application may generate reports, print barcodes or make (graphical) presentations.

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" in your "About" dialog or the help-file of your software or - if both don't exist - in your documentation. In any case your software needs to have a copyright notice.

You agree not to release documentation concerning how to control VPE from any programming language (function calls, parameters, flags) to third parties.

You may not modify any of the distributable files.

You will remain solely responsible to anyone receiving your programs for support, service, upgrades / updates, or technical or other assistance, and such recipients will have no right to contact IDEAL Software for such services or assistance.

You will indemnify and hold IDEAL Software, its related companies, and its suppliers harmless from and against any claims or liabilities arising out of the use, reproduction, or distribution of your programs.

Limited Warranty

IDEAL Software warrants the physical media and physical documentation provided by IDEAL Software to be free of defects in materials and workmanship and that the SOFTWARE will perform substantially in accordance with the accompanying user documentation for a period of ninety (90) days from the original purchase date. If IDEAL Software receives notification within the warranty period of defects in materials or workmanship, and determines that such notification is correct, IDEAL Software will replace the defective media or documentation. DO NOT RETURN ANY PRODUCT UNTIL YOU HAVE CONTACTED IDEAL SOFTWARE.

IDEAL SOFTWARE SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, REPRESENTATIONS, OR CONDITIONS, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OR CONDITION OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. ALL OTHER IMPLIED TERMS ARE EXCLUDED.

IDEAL Software's entire liability and your exclusive remedy shall be, at IDEAL Software's option, either (a) return of the price paid or (b) repair or replacement of the SOFTWARE or hardware that does not meet IDEAL Software's Limited Warranty and which is returned to IDEAL Software with a copy of your receipt. This Limited Warranty is void if failure of the SOFTWARE or hardware has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or 30 days, whichever is longer.

NO OTHER WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IDEAL SOFTWARE DISCLAIMS ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THE SOFTWARE, THE ACCOMPANYING PRODUCT MANUAL(S) AND WRITTEN MATERIALS, AND ANY ACCOMPANYING HARDWARE. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS.

The entire and exclusive liability and remedy for breach of this limited warranty shall be limited to replacement of defective media or documentation and shall not include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profit, data, or use of the software or special, incidental or consequential damages, or other similar claims, even if IDEAL Software has been specifically advised of the possibility of such damages. In no event will IDEAL Software's liability for any damages to you or any other person ever exceed the lower of the list price or the actual price paid for the package or the license to use the software, regardless of the form of the claim.

Specifically, IDEAL Software makes no representation or warranty that the software or documentation are "error free", or meet any user's particular standards, requirements, or needs. In all events, any implied warranty, representation, condition, or other term is limited to the physical media and documentation and is limited to the 90-day duration of the limited warranty.

IDEAL Software is not responsible for, and does not make any representation, warranty, or condition concerning product, media, software, or documentation not manufactured or supplied by IDEAL Software, such as third parties programs which are designed using "IDEAL Software" software or which include IDEAL Software programs or files.

This statement may only be modified in writing signed by you and an authorized officer of IDEAL Software. If any provision of this statement is found void or unenforceable, the remainder will remain valid and enforceable according to its terms. If any remedy provided is determined to have failed of its essential purpose, all limitations of liability and exclusions of damages set forth in the limited warranty shall remain in effect. Use, duplication, or disclosure of the software and documentation in this package is forbidden.

This statement shall be construed, interpreted and governed by the laws of the Bundesrepublik Deutschland (FRG). This statement gives you specific legal rights; you may have others which vary from state to state and from country to country. (UK STATUTORY RIGHTS ARE NOT AFFECTED OR PREJUDICED AS TO SOFTWARE ACQUIRED OTHER THAN IN THE COURSE OF A BUSINESS.) IDEAL Software reserves all rights not specifically granted in this statement.

Each party hereby agrees to submit to the exclusive in personam jurisdiction of the courts of the Bundesrepublik Deutschland for the resolution of all disputes between them, with venue to be in the city Neuss.

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

1.2 Lizenzbestimmungen und Gewährleistungsbegrenzung

Erläuterungen des Nutzungsrechts (Lizenzbestimmungen) und der Gewährleistungsbegrenzung, sowie Allgemeine Geschäftsbedingungen von IDEAL Software

Diese Software genießt den Schutz des Urheberrechtsgesetzes und des Internationalen Urheberrechtsabkommens. Sie sind verpflichtet, diese Software wie ein Buch zu behandeln. Jedoch sind Sie berechtigt, die Software zur Nutzung auf einen Computer zu übertragen und berechtigt, Archivkopien herzustellen, soweit dies zu "Backup" Zwecken sowie zum Schutz ihrer Investition erforderlich ist.

IDEAL Softwares Vergleich der Software mit einem "Buch" bedeutet beispielsweise, daß, soweit hierdurch nicht die Möglichkeit einer gleichzeitigen Nutzung durch mehrere Personen eröffnet wird, die Software von einer beliebigen Anzahl von Software-Entwicklern genutzt werden darf, daß sie ohne Einschränkung von einem Computer auf einen anderen kopiert werden darf oder von einem Betriebsort an einen anderen umgezogen werden darf. Wie ein Buch nicht von zwei verschiedenen Menschen an zwei verschiedenen Orten zur gleichen Zeit gelesen werden kann, darf ohne IDEAL Softwares Zustimmung (es sei denn, unter Verletzung der Urheberrechte von IDEAL Software) die Software nicht von zwei verschiedenen Software-Entwicklern an zwei verschiedenen Orten zur gleichen Zeit genutzt werden. Sie müssen also je Software-Entwickler, der diese Software benutzt, eine eigene Lizenz erwerben.

Weitere Nutzer (Software-Entwickler)

Die Anzahl der Nutzer können Sie durch den Erwerb weiterer einzelner "Software-Pakete" nach Belieben erhöhen. Ebenso möglich ist der Erwerb eines LAN Pakets/einer LANLizenz (lediglich für die Nutzung innerhalb eines Local Area Networks), der Erwerb einer Vielfachnutzerversion oder der Erwerb weiterer Nutzungsrechte an der Software. Die Anzahl der tatsächlich gleichzeitigen Nutzer der Software darf dabei die Anzahl der berechtigten Nutzer nicht übersteigen (andernfalls nutzen Sie ja die Software gerade nicht "wie ein Buch"). Übergibt IDEAL Software in dem Software-Paket verschiedene Speichermedien (beispielsweise 3 1/2 und 5 1/4 Zoll Disketten), so liegt hierin keine Erlaubnis für eine Mehrfachnutzung der Software.

Übertragung der Software

Ihre Nutzungsrechte an der Software können Sie insgesamt an eine dritte Person übertragen. Dazu ist erforderlich, daß Sie die vollständige Software nebst Disketten und Dokumentationen, also alle Bestandteile dieses Software Pakets (dies umfaßt auch diese Erläuterungen) und alle Kopien davon, gleich welcher Art, übertragen oder, daß Sie die nicht übertragenen Kopien zerstören. Denken Sie daran, sobald Sie die Software übertragen haben, sind Sie selber zur Nutzung nicht mehr berechtigt. Auch die Person, an die Sie die Software übertragen haben, ist lediglich berechtigt, diese in Übereinstimmung mit dem Urheberrechtsgesetz, dem Internationalen Urheberrechtsabkommen und diesen Erläuterungen zu benutzen.

Soweit Sie ein "Upgrade" einer Software erworben haben, stellt dieses eine untrennbare Einheit mit der ursprünglichen "IDEAL Software" Software dar. Beispielsweise darf das Upgrade und die Ursprungssoftware ohne schriftliche Erlaubnis von IDEAL Software nicht von zwei unterschiedlichen Personen zur gleichen Zeit genutzt werden und ebensowenig können die beiden Versionen getrennt voneinander übertragen werden.

Vorbehaltlich des Inhalts dieser Erläuterung sind Sie nicht berechtigt, die Software zu ändern, zu übersetzen, unterzulizensieren, im Timeshare-Verfahren zu nutzen oder elektronisch zu übertragen oder zu empfangen; dies gilt entsprechend für die Speichermedien und für die Dokumentation.

Lizenzbestimmungen - Verteilbare Dateien

Ausgenommen von den obigen Bestimmungen sind folgende Dateien, für die Sie mit Kauf dieses Software-Pakets das Recht erworben haben, diese eingebettet in Ihre Produkte an Dritte weiterzugeben. Diese Weitergabe beinhaltet die ausschließliche Nutzung der genannten Dateien in Zusammenhang mit Ihrem Produkt (Ihren Produkten).

Abhängig von der erteilten Lizenz und Plattform-Unterstützung sind folgende Dateien verteilbar:

16 Bit:	32-Bit:
VPENGINE.DLL	VPE32.DLL
VPECTL16.OCX	VPECTRL.OCX
EASYBAR.DLL	EZBAR32.DLL
DAVINCI.DLL	DAV32.DLL
LEON.DLL	LEON32.DLL

Hinweis für DAVINCI.DLL (DAV32.DLL) und LEON.DLL (LEON32.DLL):

Copyright © by Ingenieurbüro Herd & Nuding, diese DLL's dürfen nur in direkter Verbindung mit VPE vertrieben und genutzt werden. Sie dürfen diese nicht selbst benutzen / aufrufen. Wenn Sie solche Bedürfnisse haben, wenden Sie sich bitte direkt an:

Ingenieurbüro Herd & Nuding; Niederstr. 36; 64285 Darmstadt; Germany
Tel. 06206-707775 * Fax. 06206-707776 * BBS 06161-664741

Hinweis für EASYBAR.DLL (EZBAR32.DLL):

Copyright © by Bokai Corporation, diese DLL's dürfen nur in direkter Verbindung mit VPE vertrieben und genutzt werden. Sie dürfen diese nicht selbst benutzen / aufrufen. Wenn Sie solche Bedürfnisse haben, wenden Sie sich bitte direkt an:

Bokai Corporation; 1221 Dundix Rd., #106; Mississauga, ON L4Y 3Y9; Canada; Fax: (+1) 905 276-7692

VPE muß ein integrierter Bestandteil Ihrer Software sein. Ihre Software muß einen signifikant abweichenden Einsatzzweck von VPE haben. D.h. Ihre Software darf kein Produkt darstellen, das gleich oder ähnlich VPE ist. VPE darf nur zum Drucken und zur Darstellung von Previews benutzt werden. Beispiele für Anwendungen, in die VPE nicht integriert werden darf: Report-Generatoren, Barcode Druck Anwendungen, Grafik-Präsentations-Software, Datenbank-Engines, jegliche Kombinationen dieser Produktkategorien, etc., o.ä. Ansonsten dürfen Ihre Anwendungen mit VPE Reports generieren, Barcodes drucken und (grafische) Präsentationen erstellen.

Diese Lizenz gibt Ihnen das Recht, VPE (in Form der vorgenannten Dateien - abhängig von der erteilten Lizenz; 16- und/oder 32 Bit) in Ihre Anwendungen zu integrieren und an Dritte in dieser Form weiterzugeben. Aber diese Dritten haben kein Recht, VPE an wieder andere Dritte weiterzugeben, gleich in welcher Form. Wenn Sie solche Bedürfnisse haben, so wenden Sie sich bitte direkt an IDEAL Software.

Wenn Sie die Flag VPE_NO_INFOBTN oder VPE_NO_TOOLBAR benutzen, oder ohne Preview arbeiten, erklären Sie sich damit einverstanden, einen Copyright-Vermerk wie: "Virtual Print Engine Copyright © by IDEAL Software" in Ihren "About" Dialog oder der Hilfedatei (Helpfile) oder - wenn beides nicht vorhanden in Ihrer Dokumentation einzufügen. In jedem Fall muß Ihre Software einen Copyright-Vermerk tragen.

Sie erklären sich weiterhin damit einverstanden, keine Informationen darüber an Dritte weiterzugeben, wie man VPE von einer Programmiersprache aus benutzen kann (Funktionsaufrufe, Parameter, Flags).

Sie dürfen die verteilbaren Dateien in keiner Weise modifizieren.

IDEAL Software schließt jede Gewährleistung gegenüber Dritten aus, ausgenommen hiervon ist die Gewährleistung die gegenüber dem Ersterwerber übernommen wird.

Sie alleine sind gegenüber jedem Empfänger Ihrer Programme verantwortlich für Unterstützung, Service, Upgrades / Updates oder technischen oder sonstigen Beistand.

Sie stellen IDEAL Software und mit ihr verbundene Unternehmen und Zulieferer von jeglichen Ansprüchen und jeglicher Haftung in Verbindung mit der Nutzung, der Vervielfältigung oder dem Vertrieb der Programme frei und ersetzen jeden daraus resultierenden Schaden.

Gewährleistungsbeschränkungen

Für einen Zeitraum von 90 Tagen nach dem Ersterwerb gewährleistet IDEAL Software, daß die Speichermedien und die Dokumentation frei von Material- und Verarbeitungsfehlern sind, und daß die Software im wesentlichen gemäß dem begleitenden Produkthandbuch arbeitet. IDEAL Software wird fehlerhafte Speichermedien und Dokumentationen austauschen, soweit die Material- und Verarbeitungsfehler IDEAL Software innerhalb der Gewährleistungsfrist angezeigt werden und IDEAL Software der Auffassung ist, daß die Anzeige zutreffend ist. Senden Sie das Produkt auf keinen Fall ein, bevor Sie sich mit IDEAL Software in Verbindung gesetzt haben.

Diese Garantie wird von IDEAL Software als Hersteller des Produktes übernommen; etwaige gesetzliche Gewährleistungs- oder Haftungsansprüche gegen den Händler, von dem Sie Ihr Exemplar der Software bezogen haben, werden hierdurch weder ersetzt noch beschränkt.

Die gesamte Haftung von IDEAL Software und Ihr alleiniger Anspruch besteht nach Wahl von IDEAL Software entweder (a) in der Rückerstattung des bezahlten Preises oder (b) in der Reparatur oder dem Ersatz der SOFTWARE oder der Hardware, die der beschränkten Garantie von IDEAL Software nicht genügt und zusammen mit einer Kopie Ihrer Quittung an IDEAL Software zurückgegeben wird. Diese beschränkte Garantie gilt nicht, wenn der Ausfall der SOFTWARE oder der Hardware auf einen Unfall, auf Mißbrauch oder auf fehlerhafte Anwendung zurückzuführen ist. Für eine Ersatz-SOFTWARE übernimmt IDEAL Software nur für den Rest der ursprünglichen Garantiefrist oder für 30 Tage eine Garantie, wobei der längere Zeitraum maßgebend ist.

IDEAL SOFTWARE SCHLIESST FÜR SICH JEDE WEITERE GEWÄHRLEISTUNG BEZÜGLICH DER SOFTWARE, DER ZUGEHÖRIGEN HANDBÜCHER UND SCHRIFTLICHEN MATERIALIEN UND DER BEGLEITENDEN HARDWARE AUS.

WEDER IDEAL SOFTWARE NOCH DIE LIEFERANTEN VON IDEAL SOFTWARE SIND FÜR IRGENDWELCHE SCHÄDEN (UNEINGESCHRÄNKT EINGESCHLOSSEN SIND SCHÄDEN AUS ENTGANGENEM GEWINN, BETRIEBSUNTERBRECHUNG, VERLUST VON GESCHÄFTLICHEN INFORMATIONEN ODER VON DATEN ODER AUS ANDEREM FINANZIELLEN VERLUST) ERSATZPFLICHTIG, DIE AUFGRUND DER BENUTZUNG DIESES IDEAL SOFTWARE-PRODUKTES ODER DER UNFÄHIGKEIT, DIESES IDEAL SOFTWARE-PRODUKT ZU VERWENDEN, ENTSTEHEN.

Die vorgenannte Gewährleistung ist dem Grunde nach abschließend. Dem Umfang nach ist sie auf den Ersatz des fehlerhaften Speichermediums oder der fehlerhaften Dokumentation begrenzt. Der Ersatz eines weitergehenden Schadens ist ausgeschlossen. Dies gilt insbesondere für entgangenen Gewinn, für Datenverlust oder für fehlende Benutzbarkeit der Software sowie für mittelbare oder Mangelfolgeschäden. Die Beschränkung behält ihre Gültigkeit auch für den Fall, daß IDEAL Software besondere Kenntnis von der Möglichkeit eines Schadenseintritts hatte. Auf jeden Fall ist die Haftung von IDEAL Software, gleich aus welchem Rechtsgrund, für einen Ihnen oder einer Dritten Person entstandenen Schaden der Höhe nach auf maximal den geringeren Betrag des Listenpreises oder des tatsächlich für das Software-Paket oder das Nutzungsrecht an der Software gezahlten Betrages begrenzt.

IDEAL Software schließt hiermit ausdrücklich alle ausdrücklich oder stillschweigend gegebenen Gewährleistungen, Zusicherungen oder Eigenschaften insbesondere alle Zusicherungen im Hinblick auf die Verwendungsfähigkeit für einen vertraglich vorgesehenen Gebrauch der Software aus. Entgegenstehende Bedingungen sind ungültig.

IDEAL Software übernimmt insbesondere keine Gewähr für die "Fehlerfreiheit" der Software oder der Dokumentation und steht auch nicht für das Erreichen von Kundenstandards oder die Befriedigung von Bedürfnissen des Kunden ein. In jedem Fall sind stillschweigend erteilte Gewährleistungen und Zusicherungen auf die Eigenschaften der Speichermedien und der Dokumentation begrenzt und gelten maximal für 90 Tage.

Jede Zusicherung oder Gewährleistung von Eigenschaften im Hinblick auf Produkte, Speichermedien, Software oder Dokumentation, die IDEAL Software weder hergestellt noch geliefert hat, ist ausgeschlossen. Dies gilt insbesondere für Programme dritter Parteien, die zur Nutzung in Verbindung mit "IDEAL Software" Software vorgesehen sind oder die IDEAL Software Programme oder Dateien enthalten.

Änderungen dieser Erläuterungen bedürfen der Schriftform und müssen von Ihnen und einem Vertreter von IDEAL Software unterzeichnet sein.

Soweit Teile dieser Erläuterungen unwirksam oder undurchsetzbar sind, wird die Wirksamkeit und Durchsetzbarkeit der Erläuterungen im übrigen hiervon nicht berührt. Schlägt eine Nachbesserung fehl, so bleiben die Haftungs-begrenzungen und die Gewährleistungsbeschränkungen wirksam. Jede Nutzung, Vervielfältigung oder Offenbarung der Software oder der Dokumentation, die Bestandteil dieses Pakets ist, ist untersagt.

Diese Erläuterungen unterliegen dem Recht der Bundesrepublik Deutschland. Neben den hierin aufgeführten Rechten können Ihnen von Land zu Land unterschiedlich weitere Rechte zustehen. Mangels abweichender Regelung in diesen Erläuterungen behält sich IDEAL Software alle weiteren Rechte ausdrücklich vor.

Für die Geschäftsbedingungen und die gesamten Rechtsbeziehungen zwischen IDEAL Software und dem Käufer / Anwender gilt das Recht der Bundesrepublik Deutschland. Die Stadt Neuss ist ausschließlicher Gerichtsstand für alle sich aus dem Vertragsverhältnis unmittelbar oder mittelbar ergebenden Streitigkeiten.

Sollten einzelne oder mehrere Bestimmungen dieser "Erläuterungen des Nutzungsrechts (Lizenzbestimmungen) und der Gewährleistungsbegrenzung, sowie Allgemeine Geschäftsbedingungen von IDEAL Software" unwirksam sein oder werden oder eine Regelungslücke enthalten, so verpflichten sich die Vertragsparteien, in Verhandlungen mit dem Ziel einzutreten, die unwirksame oder unvollständige Bestimmung durch eine angemessene Individualabrede zu ersetzen oder zu ergänzen, die dem wirtschaftlichen Zweck der gewollten Regelung weitestgehend entspricht. Die Gültigkeit der übrigen Bestimmungen bleibt davon unberührt.

DIE BENUTZUNG (AUSFÜHRUNG) DIESER SOFTWARE BEDEUTET DIE ANERKENNUNG DIESER BESTIMMUNGEN.

1.3 Trademarks

Adobe is a registered trademark of Adobe Systems Incorporated.

True Type is a registered trademark of Apple Computer, Inc.

SQL Windows, Centura and Gupta are registered trademarks of Centura Corporation.

Hewlett-Packard and Laserjet are registered trademarks of Hewlett-Packard Company.

Delphi, Turbo Pascal and Borland are registered trademarks of Borland International.

Watcom is a registered trademark of Sybase, Inc. and its subsidiaries.

Micro Focus and Micro Focus COBOL are registered trademarks of Micro Focus Ltd.

Microsoft, Visual Basic, Visual C, Visual FoxPro, FoxPro, Microsoft Access, Word for Windows and Windows,

Windows 95 and Windows NT are registered trademarks of Microsoft Corporation.

Windev is a trademark of PCSOFT France.

All other trademarks and service marks are the property of their respective owners.

1.4 Installation

Run **SETUP.EXE** and follow the instructions on the screen. SETUP will install the following directories / files in the directory you specify:

File / Directory	Meaning
c\ delphi\ foxpro\ imp_libs\ mf_cobol\ pascal\ sqlwin\ vb\ vfoxpro\ davinci.dll (dav32.dll) easybar.dll (ezbar32.dll) leon.dll (leon32.dll) minidemo.cpp minidemo.exe (mini32.exe) vpe.cnt vpe.hlp vpe_ov.cnt vpe_ov.hlp vpectl16.ocx (vpectrl.ocx) vpedemo.exe (vpedmo32.exe) vpengine.dll (vpe32.dll) several demo images	<p>directory with header-files and demo source for use with C/C++ compilers, contains also import-library for other compilers</p> <p>directory with interface and demo source for use with Delphi 16-bit and 32-bit including subdirectories with 16- and 32-bit VCLs (installation see below)</p> <p>directory with interface and demo source for use with FoxPro and Visual FoxPro</p> <p>directory with import library for linking the DLL with compilers (also .DEF file for 32-bit version and instructions)</p> <p>directory with demo source for use with Micro Focus COBOL for Windows</p> <p>directory with demo source for use with Borland Pascal (interface is vpengine.pas in directory \delphi)</p> <p>directory with engine-APL and APT demo source for use with SQLWindows / Centura</p> <p>directory with demo source for use with Visual Basic 4.0 16-bit and 32-bit (can also be used as example for Access)</p> <p>directory with demo source for use with Visual FoxPro</p> <p>image import filter DLL</p> <p>barcode DLL</p> <p>image manipulation DLL</p> <p>mini sample C-program on how to use VPE (displays measuring in inches)</p> <p>the compiled mini sample program</p> <p>vpe.hlp's contents file for Win95 / NT help</p> <p>helpfile, Programmer's Manual</p> <p>vpe_ov.hlp's contents file for Win95 / NT help</p> <p>helpfile, OCX / VCL Reference</p> <p>VPE-OCX, ready registered on system level</p> <p>a demonstration of what VPE can do (for explanation see "" on page)</p> <p>the engine itself</p> <p>in BMP and PNG formats</p>

All directories may contain important README.TXT files!

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 the runtime of your application.

1.4.1 How is the Windows System Directory Affected by SETUP?

Without version checking, SETUP installs VPEngine and all related files in the target directory you specify.

With version checking, **all** DLL's that come with VPEngine are additionally copied to the Windows System (or System32) directory. Also the OCX supplement DLL's are installed with version checking in the Windows System (or System32) directory.

Version checking means: SETUP does not overwrite files, which have higher version numbers in their resource version information.

The OCX supplement DLL's for the 16 bit version are:

- OC25.DLL
- MFC250.DLL

The OCX supplement DLL's for the 32 bit version are:

- MFC40.DLL
- MSVCRT40.DLL
- OLEPRO32.DLL

1.4.2 Installing the VPE-OCX

After SETUP has been executed successfully, the OCX's (16 and / or 32 bit) are ready installed on system level. This means they are available for all container applications. Nevertheless each container (like Visual Basic 4.0 or Visual FoxPro) requires, that you register it within it, before you can use it. How to register OCX's in a specific container can be read in the manual of the container application.

For Example: In Visual Basic 4.0 you select the menu entry "Tools" and then "Custom Controls". A dialog box appears, with a list of all available OCX's on your system. Scroll the list down until "VPEngine Control" is listed. Click at the line so that it is checked, then click on "Ok". The VPE-OCX is ready for use with Visual Basic 4.0.

If you open AUTOLD16.VBP (AUTOLD32.VBP), execute the above procedure and save the project then, the OCX will always be ready installed in the Toolbox when you start Visual Basic.

1.4.3 Installing the Delphi VPE-VCL and the .KWF File

After SETUP has been executed successfully, you will find the following two directories within the VPE \ Delphi directory:

- VPE_VCL.16 - the 16 bit VCL
- VPE_VCL.32 - the 32 bit VCL

Delphi 1.0: Select the menu entry "Options" and then "Install Component".

Delphi 2.0: Select the menu entry "Component" and then "Install".

Both: A dialog box appears, with a list of all installed VCL's. Click on "Add", a browse dialog appears. Browse to the directory of the VCL you want to install (for example "C:\VPE\DELPHI\VPE_VCL.32"). Select as filetype "*.PAS" and you will see "VPE_VCL.PAS". Select this one and click "Ok". You return to the dialog with the list of installed VCL's. Now click on "Ok" there, and Delphi will install the VPE-VCL. This will take some time, and your computer will make intensive use of your harddisk. This is, because Delphi links the new VCL into its component DLL. After having finished this procedure, VPE-VCL is ready for use. You will find its icon in the **system-tab** of the toolbar.

The **.KWF file** in the directory VPE \ Delphi allows you, to link VPE_OV.HLP - the help file for the VPE-OCX and VPE-VCL - into the IDE of Delphi.

To install the .KWF file, execute HELPINST.EXE. It is located in

Delphi 1.0: <your Delphi directory> \ HELP \ HELPINST.EXE

Delphi 2.0: <your Delphi directory> \ HELP \ TOOLS \ HELPINST.EXE

Run HELPINST.EXE, select the menu entry "File" and then "Open", browse to Delphi's "BIN" directory. Select "DELPHI.HDX". Click on the "Add" button in the toolbar (the one with the "+" sign) and browse to the directory, where VPE_OV.KWF is located (for example "C:\VPE\DELPHI"). Select "VPE_OV.KWF" and click "Ok". Afterwards select the menu entry "File" and then "Save".

For 16-bit Delphi, you need to have VPE_OV.HLP in the search path, or you copy it for example into the DELPHI/BIN directory. For 32-bit Delphi this is not needed, you may located VPE_OV.HLP the first time the system is searching for it.

If you now highlight a keyword of VPE in the editor of Delphi's IDE and press F1, you will get the right help.

For both, installation of the VCL as well as installing the .KWF file please refer also to your Delphi manuals.

1.5 What's New in Version 2.0 (10/96)

This chapter is intended for user's of previous versions of VPEngine, to give them a short overview of what has changed. Skip this section, if you're a new user. If you left updates out: read also the chapter "Revision History" at the end of this manual.

- **The major enhancement: a brandnew VPE-OCX and VPE-VCL !!!**

- New SETUP.EXE

- The user interface of VPE now uses correctly the color scheme of the system settings.

- New functions:

VpePenSize(long hDoc, int PenSize)

VpePenStyle(long hDoc, int PenStyle)

VpePenColor(long hDoc, COLORREF PenColor)

- OpenDoc-Flag VPE_GRID_INBACKGROUND removed (who did ever use it?). The grid is per default in background unless you specify VPE_GRID_INFOREGROUND.

- New OpenDoc-Flag VPE_NO_SCALEBTN and redefined VPE_NO_USER_SCALE:

```
#define VPE_NO_MOUSE_SCALE      256    // User can't scale with the MOUSE
#define VPE_NO_SCALEBTN        512    // No Scale Buttons in Toolbar
#define VPE_NO_USER_SCALE      768    // User can't scale
```

- Message VPE_PRINTCANCEL removed. WPARAM of message VPE_PRINT can have the following values:

```
enum    // WPARAM of message VPE_PRINT
{
    PRINT_MSG_ABORT=0,        // User aborted
    PRINT_MSG_START=1,       // Print started
    PRINT_MSG_END =2,        // Print ended
};
```

- Minor changes and bug-fixes in the user interface.

- Minor bug-fixes in the code.

2 Introduction

2.1 Overview

VPEngine 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.). Additionally it comes as **OCX** and as Delphi **VCL**. All outputs can be generated by using simple commands as easy as if you were using a flat DOS screen.

With VPEngine you generate dynamic reports and documents, draw graphs and charts or fill in labels and 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 0.1 mm.

VPEngine offers FAST! scaleable vector graphics, intelligent printer setup and total control about what's happening.

VPEngine is a high-quality product created in Germany. Its absolutely professional features, ease of use, performance and robustness make it a **MUST** for every serious software house and programmer.

It helps increasing productivity by many times, having it's position in the list of your tools at the point, where you can't go further on with standard report generators.

VPEngine's user interface "speaks" English, French or German, depending on the language chosen in the control-panel of the system VPE is *actually* running on.

VPEngine In Short:

- Unlimited number of pages per document (only limited by memory / harddisk space)
- Unlimited number of simultaneously open documents (only limited by memory / harddisk space)
- VPEngine is database independent since you feed it the needed data.
- 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 for example 100 virtual pages and move virtually to the first page to draw 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 (including footers and headers)
- Scale to gray for bitmaps (and doubling images: one for the screen display / one for printing)
- Scaled printing
- Big pages printed in segments over several small pages
- Charts
- VPE Designer for simply laying-out page- and table-templates with the mouse
- VPE Server for use with EVERY programming language / tool on EVERY computer system (for example AS/400, UNIX, OS/2, mainframes, ...) which is able to generate textfiles and to access a PC's harddisk via network. You simply generate a scriptfile (plain text) which is interpreted and printed by VPE Server on the PC.
- VPE Form which will give your end-users the ability to fill in forms on the screen interactively with the keyboard and mouse.

2.2 The Demo VPEDEMO.EXE

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

Welcome

An introduction.

Capabilities + Precision

This demo shows text formatting features, drawing features, bitmap handling, form filling and of course 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 box "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. Feel invited, to try a 300 DPI bitmap on your own.

Speed + Tables

Here you can see how fast VPEngine builds a report with a size of about 100 - 120 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 VPEngine how to build the report.

Since it is random data, the number of pages differs from 100-120 pages. **Note**, that this demo prints the number of generated pages finally 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 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 button, 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 and a pie chart on the second page. The source code shows very fine, how easy creating reports is.

Auto Text Break

This demo shows the powerful capability of VPE, to render text automatically over multiple pages. The demo reads the whole file "vpedemo.cpp" into memory and formats it into the document with a single VpeWriteBox() command.

3 How To Use VP Engine

3.1 General

The source codes of the several demonstration programs have been created very careful, considering the specific characteristics of each programming language. They show all basic and advanced techniques of how to use and control VPE from each programming language.

Source code tells much more in much shorter time, than abstract descriptions do, and we understand it as a substantial part of the documentation. So we strongly recommend, that you study the documentation as well as the source codes.

3.1.1 Basics

All objects in VPE are positioned and sized with 1/10 mm exactness.

$$10 * 0.1\text{mm} = 1\text{mm}$$

$$100 * 0.1\text{mm} = 10\text{mm} = 1\text{cm}$$

$$1\text{ inch is } 2.54\text{cm, so } 5\text{ inch} = 5 * 2.54\text{ cm} = 12.7\text{cm}$$

Most output functions need a starting coordinate (x, y), and some need an ending coordinate (x2, y2).

X and X2 is the offset to the left page margin in 1/10 mm.

Y and Y2 is the offset to the top page margin in 1/10 mm.

To draw a line starting at 1cm from the right and 1cm from the top, ending at 5cm from the right and 5cm from the top, you enter: `Line(100, 100, 500, 500)`

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.

3.2 VPE DLL

The common sequence of function calls is:

- Open a virtual document with the function "VpeOpenDoc()"
- Use all possible output calls
- Use "VpePageBreak()" to generate new pages
- Use "VpePreviewDoc()" to show the preview to the user, this is optional
- Use "VpePrintDoc()" to print the document
- Close the document with "VpeCloseDoc()"

You may open as many documents simultaneously as you like. The documents are identified by a unique handle (long) with a value different from NULL, which is returned by "VpeOpenDoc". Use this handle in successive calls to the output functions.

Each document might send messages. The messages are received by the window, that is specified as parent-window in the first parameter of "VpeOpenDoc".

Normally, you don't need to care about closing the document, because it's automatically closed, when the parent window is destroyed.

Example in C, which can be easily translated to other programming languages:

```
void Doc(HWND hWndParent)
{
    long hDoc;

    hDoc = VpeOpenDoc( hWndParent, "Test", US_LETTER, -1, 0 );
    VpeLine( hDoc, 100, 100, 500, 500 );
    VpePreviewDoc( hDoc, NULL, VPE_SHOW_NORMAL );
}
```

This is very easy. The preview window is automatically closed, when the parent window is closed. But you have to take care, that **your parent window (and especially your application) may not be closed, while VPE is printing**. Otherwise this will cause a GPF. The following code in the window procedure takes care for this, because "VpeCloseDoc" returns False, while the document is printing:

```
case WM_CLOSE:
    if (!VpeCloseDoc(hDoc)) // can't close, because printing ???
    {
        DestroyWindow(hWnd); // no, job is NOT printing, we can close
        return 1;
    }
    MessageBox(hWnd, "Can't close, job is printing!", "WARNING:", MB_OK);
    return 0;

case WM_DESTROY:
    PostQuitMessage(0);
    return 0;
```

VERY IMPORTANT:

After calling "VpePrintDoc", code execution is suspended until the document has been completely printed. So the following command sequence is absolutely valid:

```
VpePrintDoc(hDoc, True or False);  
VpeCloseDoc(hDoc);
```

The document will be closed AFTER it has been printed (or, if the print has been aborted by the user, which can be realized with the "VPE_PRINT" message event). But due to the event-driven mechanism of Windows, your application is still "alive" and other actions can take place (like button clicks), and the related code will be executed.

The closing of the document fires the message event "VPE_DESTROYWINDOW" to the parent window, which can be used in several ways to have control about what's happening. For example you can set hDoc to NULL, at application startup and when receiving this message. So you can always check with hDoc != NULL, if a specific document is open.

3.3 The Object-Oriented Style

VPE knows the following objects:

- Pen Style-Object
- Background Style-Object
- Hatch Style-Object
- Foreground Style-Object (accessed with VpeSetTextColor())
- Line
- Polyline
- Polygon
- Ellipse
- Frame
- Picture
- Box
- Barcode
- Text

Attributes for one object inherit to the others.

The inheritance order is:

Examples:

- 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 (where the bars and the add-on text is drawn with the actual text-color).
- A text(box) is drawn with the settings for the box (background fill color, hatchstyle and hatchcolor) and the actual pen.

So if you set the pensize to 0, no frame is drawn around any object.

Note: "Frame" is a virtual object, you cannot access it (i.e. there is no function VpeFrame or so), use VpeBox() and set with VpeSetTransparentMode() the background transparency to "on".

3.4 Important Note About Pens, Lines, Frames, Circles and Ellipses

Pens draw always with their thickness (i.e. pensize) around the coordinates. All objects using the pen, will have this behaviour. So 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.

In the following drawing the white dot in the middle of the arrows is the x, y starting coordinate. The arrows show the extent by the pensize.

The sense is, that if you draw a table (as in the demo "Speed + Tables"), the lines of two neighboring boxes will exactly overlap, so that no double (bold) lines will appear.

For circles and ellipses this means, that the radius has an extent by $\frac{1}{2}$ of the pensize:

3.5 Dynamic Positioning

VPEngine's concept of "Dynamic Positioning" is very important for its efficient use.

It's a very powerful concept, which helps to position and size objects relative to each other. Since dimensions of images and variable text can not be known in advance, VPEngine offers you several mechanisms to determine them during the creation of the document, that is: during runtime.

We strongly recommend, that you read the following sections careful, and understand them.

3.5.1 The Basic Concept

Until version 1.2, objects could only be absolute positioned and sized. There were always two coordinate pairs x, y and x_2, y_2 to set the position of the top left corner and the bottom right corner:

Rectangle	example for a circle, which is a rectangular object in VPE
x, y	x, y
x_2, y_2	x_2, y_2

Now the values for x_2 and y_2 may be negative and are then interpreted as width and height relative to the upper left corner. But x_2 and y_2 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 `VpePictureXXX` functions 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 called V-Flags will help:

- VFREE:** a flag for indicating, that VPE shall compute a width/height. It can only be used as y_2 (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 x_2 (width) AND / OR y_2 (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 on the current page
- VRIGHT:** the x_2 -coordinate of the last inserted object on the current page
- VTOP:** the y -coordinate of the last inserted object on the current page
- VBOTTOM:** the y_2 -coordinate of the last inserted object on the current page

Note, that VPEngine keeps track of these coordinates for each page separately.

Examples:

```
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 available 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.

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 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 output rectangle can be defined and retrieved.
- Each page in a document can have its individual output rectangle.
- After creating a Document with VpeOpenDoc[File]() the default-rectangle is set to:
top = 200, left = 200, right = page_width - 200, bottom = page_height - 200
- The flags to access the output rectangle are:
- **VLEFTMARGIN, VRIGHTMARGIN, VTOPMARGIN, VBOTTOMMARGIN**
- On an initial blank page, VLEFT is VLEFTMARGIN, VRIGHT is VRIGHTMARGIN, VTOP is VTOPMARGIN and **VBOTTOM is VTOPMARGIN (!)**.

All explained V-Flags except VFREE can be used for ALL objects.

3.5.2 Advanced Concept

Imagine, you created a report. Later you want to change the width of one field in a table. Would you really 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 coordinates with the functions **VpeGet** and **VpeSet**.

Instead of the DLL functions VpeGet and VpeSet, the OCX and VCL offer you the following properties:

nLeft, nTop, nRight, nBottom, nLeftMargin, nTopMargin, nRightMargin, nBottomMargin

They relate to the V-Flags, but you can modify and compute with them. They make the use of the Dynamic Positioning much more easy and transparent.

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

OCX / VCL: `<Object>.Write(VLEFT, <Object>.nBottom + 100, VRIGHT, VFREE, "text3");`

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

Example, DLL: `VpeSet(hDoc, VLEFTMARGIN, 300);`

OCX / VCL: `<Object>.nLeftMargin = 300;`

Sets the left margin of the output rectangle for the current page to 3 cm.

3.6 Automatic Text Break

This is a very powerful feature! VPE is able to render text to the bottom of the page (to the defined bottom of the output rectangle) and to insert the remaining text automatically onto successive pages.

VPEngine has three different Auto Break modes:

AUTO_BREAK_ON

The Auto Break will only work, if $y_2 = VFREE$, or $y > VBOTTOMMARGIN$.

Remaining text is broken onto the next page(s) at

$NewX = OldX$ and $NewY = \text{top of the output rectangle of the new page}$.

AUTO_BREAK_OFF

Same behaviour as **AUTO_BREAK_ON** (limited positioning / rendering to the bottom of the output rectangle is active), but remaining text is NOT broken onto next page(s). It is cut.

AUTO_BREAK_NO_LIMITS

Remaining text is NOT broken onto the next page(s), but it can be placed anywhere on the paper with no limits.

Use this switch to have the full compatible behaviour of VPE v1.4 with all previous versions.

Example, DLL: `VpeSetAutoBreak(hDoc, AUTO_BREAK_OFF);`

OCX / VCL: `<Object>.AutoBreak = AUTO_BREAK_OFF`

The default after calling OpenDoc is AUTO_BREAK_ON. For a detailed example take a look at the Auto Render Demo in the source code of VPEDEMO.

3.6.1 Rotation of Text, Images and Barcodes

VPE is able to rotate text, images and barcodes freely in 90 degree steps.

Rotation is done clockwise, the <angle> parameter is specified in 1/10 degrees.

So the value for rotating by 90 degrees is 900.

Example, DLL: `VpeSetRotation(hDoc, 900);`

OCX / VCL: `<Object>.Rotation = 900`

Important:

When images or text are rotated, the given starting coordinate (x, y) stays unchanged.

But x2 and y2 are **transformed** into x2', y2'.

Look at the following simple example for images:

0 degrees:	90 degrees:	180 degrees:	270 degrees:
x, y	x, y	x, y	x, y
	x2, y2	x2', y2'	x2', y2'
			x2', y2'

This example is simple, because the width and the height is the same, so rotation doesn't matter.

The same for text, but now width and height aren't identical:

0 degrees:	180 degrees:
x, y	x, y
x2, y2	x2', y2'

90 degrees:	270 degrees:
x, y	x, y
x2', y2'	x2', y2'

How rotation is performed:

Say, you insert the text above at position (0, 0, 100, 50). The command would be:

VpeWriteBox(hdoc, 0, 0, 100, 50, "Hello World!")

Now you want to rotate it by 90 degrees. The commands would be:

VpeSetRotation(hdoc, 900)

VpeWriteBox(hdoc, 0, 0, 100, 50, "Hello World!")

Why the same coordinates? - Because VPE transforms them!

Before rotating an object, VPE computes the width and the height of the object.

Regardless of the angle, the width and height will always stay the same. This makes rotation for you much easier (especially if you let VPE render the height of a text / image object), if you keep this in mind and use only relative coordinates for x2, y2 (negative signs).

Now the same example again, but with relative coordinates, and the starting position (350, 220):

VpeSetRotation(hdoc, 900)

VpeWriteBox(hdoc, 350, 220, -100, -50, "Hello World!")

Do you get the idea?

The "-100" is always the width, and the "-50" is always the height.

What about the rendering features of VPE? Of course it will work!

The command: **VpeWriteBox(hdoc, 350, 220, -100, VFREE, "Hello World!")** will do fine with every rotation angle.

Now, after this explanation, keep the following two rules in mind:

Since x_2 and y_2 are transformed, the best way to use rotation is, not to use absolute coordinates for x_2, y_2 but relative = width and height (this means: with a negative sign).

x_2' / y_2' is the final right / bottom position of the rotated object. These are the coordinates you will retrieve with VRIGHT / VBOTTOM.

See Also: "" on page

3.7 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 "" on page).

A gap may occur between the picture and the frame. If you use frames that are too fat, they will overlap the picture (see "" on page). This doesn't matter if you have pictures with a white background such as the logo.bmp, but the best solution is to have the frame in the image-files itself. Or - for the problem with the gap - make the frame a bit thicker, so that a possible gap will be hidden.

All metafiles (like WMF and DXF) are always converted to bitmaps in this version of VPE.

Metafiles (also those that are imported through Microsoft / Adobe 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 / Adobe import filters is an add-on feature for the 16-bit 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 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 PIC_xyz flags to change this standard behaviour - to have a faster processing and more memory usage, or less memory usage and slower processing.

3.8 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 comparison 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, especially when using a different scaling than 1:1 in the preview, because of performance reasons.

But be assured: VPE is a system that does its best to be WYSIWYG.

This chapter is only for people who want it very exact.

(Also some problems arise out of printer- or video-driver bugs. See "" on page .)

3.9 Positioning On the Printer

The correctly positioned output depends on the mechanical capabilities and quality of the printer. When paper is fed into the printer, there are tolerances of about 1 mm where the paper is misplaced in the y-direction. Also there are tolerances of about 0.1mm to 0.5mm in the x-direction.

3.10 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" <pen-size>

Sets the pensize. <pen-size> 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>.

"HSNone", "HSN"

"HSHorizontal", "HSH"

"HSVertical", "HSV"

"HSFDiagonal", "HSFD"

"HSBDiagonal", "HSBD"

"HSCross", "HSC"

"HSDiagCross", "HSDC"

Set the hatch-style. For styles see function "" on page .

"HatchColor", "HC" <color-name>

Sets the hatchcolor. <color-name> is one of the color-strings above.

"HatchColorRGB", "HCRGB" <red> <green> <blue>

Sets the hatchcolor with RGB-values <red> <green> <blue>. Example: "HCRGB 200 210 30"

"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.

"AutoBreak", "A"

Sets the AutoPageBreak-Mode to ON.

"AutoBreakOff", "AO"

Sets the AutoPageBreak-Mode to OFF.

"AutoBreakNoLimits", "ANL"

Sets the AutoPageBreak-Mode to NO LIMITS.

"Rotate", "Rot" <angle>

Sets rotation to the specified angle. (clockwise, currently only 90 degree steps allowed, angle in 1/10 degrees)

Example: "Rotate 900" = Rotate by 90 degrees clockwise

"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.

3.11 Predefined Color Constants

```
#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 representable solid color. On all other output devices, the color will be what you have chosen.

4 Important Notes & Troubleshooting

4.1 General Troubleshooting

- 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.
- Frames drawn around objects (Barcodes, Images, etc.) can be eliminated by setting the pensize to zero
- 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.
- For users of programming languages, which don't provide some datatypes:
TRUE is the value 1 and FALSE is the value 0.
COLORREF is a long-integer.
- 1 inch is 2.54cm, so 5 inch = 5 * 2.54 cm = 12.7cm
- Due to some printer- and video-drivers, the WYSIWYG capabilities of VPE are in some cases distorted.

4.2 Printer Troubleshooting

Some printer drivers (especially for Win95) have bugs. If you cover a problem with printing, please try first another printer driver, or another printer. Win95 has currently problems with some HP Laserjet Drivers. They are slow and work with problems. Please report bugs regarding Win95 HP Laserjet Drivers to Microsoft or HP.

If your output looks garbaged: Enable the option "Print True-Type as graphics" in the printer-options dialog.

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 other than `VFREE` and the last line of the text is printed in part on the screen, this last line might be printed completely on the printer. **The solution is to set y2 smaller, so that the whole line is clipped.**

Printing colors: Some printer drivers may not print colored text, printing nothing instead. This may also occur with lines and all other graphics depending on the chosen color.

Printing hatch styles: Some printer drivers may not print hatch styles correctly.

4.3 Video Troubleshooting

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
- scrolling objects with hatch styles may lead into misalignment of the hatching
- bitmaps are not shown when color-resolution of video-adaptor is higher than the bitmap-resolution (i.e., video-adaptor = 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).

4.4 If You Need Technical Assistance

- Make sure, you're a registered user of VPE. When you contact IDEAL Software for support, please include your serial number (see below).
- The preferred way is via e-mail. Normally, help will be provided within 24 hours (due to possible timezone differences). Please include "VPE" in the subject of your e-mail.
- IDEAL Software's technical support policy is as follows:
 - We offer unlimited support for program installation.
 - We offer two free support incidents relating to program use.
 - Additional support questions are \$25 (DM 35,-) each
 - Questions related to bugs or software problems are always free. You are only charged for "how-to" questions or questions that are covered in the documentation.

Provide the following information:

- Your complete registration number
- The version no. of VP Engine (also version no. of OCX or VCL)
- Programming language / tool you're working with (version no.)
- Platform (16 bit / 32 bit / name / version no.)
- Printer-Driver (name / version no.)
- Video-Card (manufacturer / version no.)
- Detailed description of your problem / symptoms, which are reproducible

5 Redistributing VPEngine

5.1 List of File Dependencies

The files you may redistribute are listed in the License Agreement.

The following diagram gives you an overview about the dependencies between the controls and DLL's:

- VPE doesn't need DAVINCI.DLL (DAV32.DLL) until a picture-function is called, but VPE can import BMP-files (not RLE) without DAVINCI.DLL (DAV32.DLL).
- VPE doesn't need LEON.DLL (LEON32.DLL) until an image is rotated.
- VPE doesn't need EASYBAR.DLL (EZBAR32.dll) until a barcode-function is called.

Keep in mind, that the DLL's you ship have to be accessible. This means: they need to be either in the directory of the calling application, or they are located in a directory which is included in the PATH environment variable.

6 DLL API (In Functional Order)

6.1 Messages Generated by VPEngine - DLL

VPEngine sends several notification messages to your application, so that you have always total control about what's happening. The messages are sent via "SendMessage()". They are sent to VPEngine's parent-window, which has been identified in the first parameter of the VpeOpenDoc() call.

6.1.1 VPE_DESTROYWINDOW

This is sent when the preview window was closed by the user (**the document is also closed!**):

WPARAM: unused

LPARAM: contains the document-handle, so you can determine which preview/document was closed (if you had more than one open).

NEVER CALL VpeCloseDoc() IN RESPONSE TO VPE_DESTROYWINDOW. THE DOCUMENT IS ALREADY CLOSED AND YOUR DOCUMENT HANDLE IS INVALID.

6.1.2 VPE_PRINT

This is sent when the user starts printing, or when printing has ended:

WPARAM: PRINT_MSG_ABORT = 0, // User aborted

PRINT_MSG_START = 1, // Print started

PRINT_MSG_END = 2, // Print ended

LPARAM: contains the document-handle, so you can determine which preview / document had send the message (if you have more than one open)

NEVER CALL VpeCloseDoc() IN RESPONSE TO VPE_PRINT. YOU WOULD TERMINATE A MODULE THAT IS STILL WORKING (OTHERWISE IT COULDN'T HAVE SENT THE MESSAGE).

6.1.3 VPE_HELP

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

WPARAM: unused

LPARAM: unused

6.2 Management Functions

Management Functions deal with the control of VPEngine itself. With the creation of virtual documents, storing and retrieving them from / to file, handling the preview, etc.

6.2.1 long VpeOpenDoc(HWND hwndParent, LPCSTR title, int page_width, int page_height, long flags)

Action: Create a new document with one initial blank page.

Parameters:

HWND hwndParent	a window of the calling application VPE exchanges messages with and the window where VPE connects its preview to, if embedded
LPCSTR 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

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

You may create an unlimited number of pages per document and an unlimited number of documents simultaneously, but both is limited by available memory. How much memory is needed, depends on the number of objects you insert and what type of objects you insert (a bitmap for example needs much more memory than a single line). So we can't give clear guidelines about memory usage. In case of doubt use a monitoring tool, to view how much memory is needed for your specific kind of document(s). For example one page in the "Speed + Tables" demo needs about 10 Kbyte of memory. This is really low, but 100 pages together need about 1MB of memory.

If the memory usage is too high, we recommend to use File Swapping (see "" on page).

The page size is free definable:

- 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

For page_width = DIN_A_4 = -1, standard DIN A 4 size (21cm x 29.7 cm) is taken.

For page_width = US_LETTER = -2, standard US Letter size (8.5 x 11 inch) is taken.

Don't worry, if you use VPE_LANDSCAPE! 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

16 Bit: there, computations may not exceed the number-range of a 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.

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_INFOREGROUND    2      // grid in foreground (default is background)
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 invisible 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 can't scale with the MOUSE
VPE_NO_SCALEBTN         512    // No Scale Buttons in Toolbar
VPE_NO_USER_SCALE       768    // User can't scale
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         131072  // Info-Button invisible

```

Most of the flags can be combined using the boolean OR operator (or simply the "+" operator). 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_TOOLBARBUTTON | VPE_GRID_VISIBLE)
VPE_GRID_OFF    0

```

Embedding the window:

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:

Use the flag VPE_EMBEDDED in VpeOpenDoc()

In the window-procedure of the parent window (where the preview shall be embedded), give the following response to the WM_SIZE message:

Call MoveWindow(VpeWindowHandle(hDoc), 0, 0, LOWORD(IParam), HIWORD(IParam), FALSE)

this will size the VPE preview window correctly with the parent window.

In the window-procedure of the parent window, respond to the WM_KEYDOWN message with:

SendMessage(VpeWindowHandle(hDoc), WM_KEYDOWN, wParam, lParam)

this will route the keyboard messages 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 GPF's might occur.

6.2.2 long VpeOpenDocFile(HWND hWndParent, LPCSTR file_name, LPCSTR 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. VPE's File Swapping is VERY fast (for example compared to Windows).
- Parameters:**
- | | |
|-----------------|---|
| HWND hWndParent | a window of the calling application VPE exchanges messages with and the window where VPE connects its preview to, if embedded |
| LPCSTR 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 |
- Returns:** The handle (=identifier) to the virtual document. This handle has to be provided to all other VPE calls.

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:

- calling VpePageBreak()
- 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. This means, the 16-bit format is fully upwards compatible, the 32-bit files can also read by the 16-bit engine, but there may no objects be bigger than 64 Kbyte, for example polyline and polygon arrays and also text objects.

6.2.3 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

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

Action: Opens the preview window

Parameters: Document Handle

rc: position of the preview window, if it is NULL, it is ignored. For use with interpreter languages, you can set rc.right = -1 then it is also ignored.

show_hide can be one of the following predefined constants:

VPE_SHOW_NORMAL = 1: show window normal

VPE_SHOW_MAXIMIZED = 2: show window maximized

VPE_SHOW_HIDE = 3: hide the preview window

Returns: -

6.2.5 void VpePreviewDocSP(long hDoc, int x, int y, int x2, int y2, int show_hide)

Action: The same as VpePreviewDoc(). "SP" stands for single-parameters, since you don't specify a RECT structure. This is very usable for interpreters, which don't support the RECT structure.

Parameters: Document Handle

position of the preview window, for x = -1 all coordinates are ignored.

show_hide can be one of the following predefined constants:

VPE_SHOW_NORMAL = 1: show window normal

VPE_SHOW_MAXIMIZED = 2: show window maximized

VPE_SHOW_HIDE = 3: hide the preview window

Returns: -

6.2.6 void VpeCenterPreview(long hDoc, int width, int height, HWND parent_window)

Action: The preview-window is centered in the middle of the parent window.

Parameters: Document Handle

width, height: the width and height of the preview window in pixels, they can take the following special values:

For width = -1: the previews width is kept

For height = -1: the previews height is kept

For width = -2: the previews width is set to the width of the parent window (or desktop)

For height = -2: the previews height is set to the height of the parent window (or desktop)

parent_window: a window, which will be used as reference point, if it is NULL (0), the preview is centered on the desktop (screen)

Returns: -

6.2.7 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 0.1mm.

Parameters: Document Handle

rulers_measure, can be one of the following predefined constants: "CM=0" or "INCH=1"

Returns: -

Default: CM

6.2.8 void VpeSetScale(long hDoc, double scale)

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

Parameters: Document Handle

Scale, for example: 1.0 is 1:1 0.25 is 1:4 4.0 is 4:1

Returns: -

Default: 1.0

6.2.9 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.

Parameters: Document Handle, yes_no (TRUE or FALSE)

Returns: -

Default: FALSE (no immediate update)

6.2.10 void VpeRefreshDoc(long hDoc)

Action: When the preview is open, this call will refresh the preview 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: -

Returns: -

6.2.11 void VpeWriteDoc(long hDoc, LPCSTR file_name)

Action: Store the actual document in the file named <file-name>. If the file already exists - and it's identified as a document-file by VPE - the actual document is appended to the end of that file. This even works, if the current document was opened by VpeOpenDocFile(), e.g. it is itself a file!

This allows merging of multiple documents. But be careful: if you used @PAGE in headers or footers, the numbers will not be correct, since they are already computed and stored in the document files. The best solution is, not to use the @PAGE function, but to store the documents without any page-number information. Then for merging with headers and footers, open a new document (file or RAM), define headers and footers, and then start to merge the documents with VpeReadDoc(). This will work very fine! Or, after you have merged all documents together in RAM, you can easily insert page numbers by calling VpeGotoPage() from 1 to VpePageCount() and each time insert the actual page number. But for this action you NEED to hold the document in RAM, because insertion of objects into existing pages of a file-document is impossible.

Parameters: Document Handle, file_name

Returns: -

6.2.12 void VpeReadDoc(long hDoc, LPCSTR file_name)

Action: Reads a document from the file named <file-name> and appends it to the current document (this even works, if the current document was opened by VpeOpenDocFile(), e.g. it is itself a file!)

The document file may have been created before with VpeOpenDocFile() or VpeWriteDoc().

Parameters: Document Handle, file_name

Returns: -

6.2.13 HWND VpeWindowHandle(long hDoc)

Action: Returns the window handle of the preview. This is a standard window handle, you can use for manipulating the preview. It is also needed, when working with embedded windows, to control the position and size of the window, and to route (send) keyboard-messages (WM_CHAR) to it.

Parameters: Document Handle

Returns: The window handle of the preview

6.2.14 int VpeGetVersion()

Action: Return the current version number.

Parameters: -

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

6.3 Printing Functions

These functions deal with setting-up the printer, controlling print-options and printing itself.

6.3.1 int VpeSetupPrinter(long hDoc, LPCSTR 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 a file (optional). Otherwise, the setting is only remembered for the lifetime of the document. NOT for the lifetime of the VPE control!

The 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: Document Handle

file_name (path and) file name of the file where to restore or store the setup or NULL

dialog_control:

PRINTDLG_NEVER = 0: never show setup-dialog (if file_name is NULL, the last setting or the setting of the default-printer will be taken)

PRINTDLG_ONFAIL = 1: show setup-dialog only, if file-read fails

PRINTDLG_ALWAYS = 2: show setup-dialog always

Returns: 0 = Ok

1 = User pushed cancel button in dialog

2 = I/O problems during read of setup file (only if dialog_control = PRINTDLG_NEVER = 0)

3 = I/O problems during write of setup file

6.3.2 int VpeSetPrintOptions(long hDoc, long flags)

Action: Allows to set print options. Currently there is only support for setting the print range regarding even, odd or all pages. This preset is also active, if the user pushes the print-button in the toolbar.

Parameters: Document Handle

flags:

PRINT_ALL 0 // print all pages

PRINT_EVEN 1 // print only even pages

PRINT_ODD 2 // print only odd pages

Returns: -

Default: PRINT_ALL

6.3.3 void VpePrintDoc(long hDoc, int with_setup)

Action: Prints the document. You must not close your application or the document **until** VPE finishes all print-jobs. (See message VPE_PRINT in chapter "" on page , the function "" on page and also function "" on page).

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 (look at the source for VPEDEMO).

Parameters: Document Handle, show setup-dialog before (TRUE) or don't (FALSE)

Returns: -

6.3.4 int VpelsPrinting(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. The VPE control will ignore all function calls while the document is printed.

Parameters: Document Handle

Returns: TRUE = is currently printing, else FALSE

6.4 Layout Functions

These functions offer you powerful ways of controlling the layout of the document, of dynamically placing and sizing objects and to define page boundaries for every individual page within the virtual document.

See also: "" on page .

6.4.1 void VpePageBreak(long hDoc)

Action: Adds a new blank page to the end of the document. Then VPE executes internally a VpeGotoPage() statement to this page, so that all further output-calls will draw onto this new page.

Parameters: Document Handle

Returns: -

6.4.2 int VpeGetPageCount(long hDoc)

Action: Retrieve the numbers of pages in the document

Parameters: Document Handle

Returns: The numbers of pages in the document

6.4.3 int VpeGetCurrentPage(long hDoc)

Action: Retrieve the number of the current active page

Parameters: Document Handle

Returns: The number of the current active page

6.4.4 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 "" on page).

Parameters: Document Handle, page number

Returns: The number of the current active page

6.4.5 void VpeStoreSet(long hDoc, int id)

Action: All current flag-settings (pen-size, alignment, colors, font, etc.) are stored in a buffer. You can create as much buffers as you like (only limited by available memory). To have access to the different buffers, you need to specify a unique ID for each. This is useful if you need the actual settings again later.

Parameters: Document Handle, id: the id under which you store the flags

Returns: -

6.4.6 void VpeUseSet(long hDoc, int id)

Action: This resets all flags to the stored values.
Parameters: Document Handle, id: the id under which you stored the flags
Returns: -

6.4.7 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: -

6.4.8 int VpeGet(long hDoc, int what)

Action: Returns the coordinate specified by one of the V-Flags provided in parameter "what".
Parameters: Document Handle, V-Flag
Returns: The coordinate specified by parameter "what".

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

Action: Sets the coordinate specified by one of the V-Flags provided in parameter "what".
Parameters: Document Handle, V-Flag, new value of the coordinate
Returns: -

6.4.10 void VpeSetDefOutRect(long hDoc, RECT *r)

Action: Sets the **default** output rectangle (will be used after each page break). The output rectangle is very important for use with the AutoPageBreak option of VPE for rendering text. See "" on page .
Parameters: Document Handle, output rectangle
Returns: -

6.4.11 void VpeSetDefOutRectSP(long hDoc, int x, int y, int x2, int y2)

Action: The same as VpeSetDefOutRect(). Sets the **default** output rectangle (will be used after each page break). The output rectangle is very important for use with the AutoPageBreak option of VPE for rendering text. See "" on page .
"SP" stands for single-parameters, since you don't specify a RECT structure. This is very usable for interpreters, which don't support the RECT structure.
Parameters: Document Handle, output rectangle
Returns: -

6.4.12 void VpeSetOutRect(long hDoc, RECT *r)

Action: Sets the **actual** output rectangle for the current active page. There is no "SP" function, use VpeSet() instead.

Parameters: Document Handle, output rectangle

Returns: -

6.4.13 void VpeGetOutRect(long hDoc, RECT *r)

Action: Gets the **actual** output rectangle for the current active page. There is no "SP" function, use VpeGet() instead.

Parameters: Document Handle, output rectangle

Returns: -

6.4.14 void VpeSetPosRect(long hDoc, RECT *r)

Action: Sets a dummy position as the last inserted object. All calls with V-Flags parameters will use these values. There is no "SP" function, use VpeSet() instead.

Parameters: Document Handle, dummy position rectangle

Returns: -

6.4.15 void VpeGetPosRect(long hDoc, RECT *r)

Action: Retrieves the rectangle of the last inserted object. There is no "SP" function, use VpeGet() instead.

Parameters: Document Handle, dummy position rectangle

Returns: -

6.4.16 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: -

6.4.17 void VpeRestorePos(long hDoc)

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

Parameters: Document Handle

Returns: -

6.4.18 void VpeSetRotation(long hDoc, int angle)

Action: Sets the new rotation angle for text, images and barcodes. Rotation is done clockwise. Angles are given in 1/10 degrees, so the command for rotating by 90 degrees is: VpeSetRotation(hdoc, 900)

The x, y position will always be the same. It is not modified by rotating. Internally VPE computes the width and height of the object, and then rotates it. So rotation is easier to use, if you work with width and height (negative signs for x2 and y2) instead of absolute values.

VPE needs LEON.DLL (LEON32.DLL) to rotate images.

Parameters: Document Handle, angle

Returns: -

Default: 0 degrees

For detailed explanation see "" on page .

6.5 Drawing Functions

This section deals with the basic drawing functions, e.g. lines, polylines, polygons, boxes, circles, etc.; and the basic style settings for these objects, which will inherit to the "" on page in the next section.

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

Action: Sets the style of the pen - all properties at once.

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 0.1mm
pen_style = one of the windows pen styles (PS_SOLID, etc.)
pen_color = RGB(x, y, z)

Returns: -

Default: 0.3 mm, PS_SOLID, Black

6.5.2 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. The pen can be made visible by setting the pensize different from 0.

Parameters: Document Handle

Returns: -

6.5.3 void VpePenSize(long hDoc, int pen_size)

Action: Sets the pen size

Parameters: Document Handle, pen_size in 0.1mm

Returns: -

Default: 0.3 mm

6.5.4 void VpePenStyle(long hDoc, int pen_style)

Action: Sets the style of the pen. 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_style = one of the windows pen styles (PS_SOLID, etc.)

Returns: -

Default: PS_SOLID

6.5.5 void VpePenColor(long hDoc, COLORREF color)

Action: Sets the color of the pen

Parameters: Document Handle, pen_color = RGB(x, y, z)

Returns: -

Default: Black

6.5.6 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, starting and ending coordinates

Returns: -

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

Action: Creates a polyline object 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().

Structure of the array <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()).

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

Action: Adds a new point to the polyline object specified in <p>.

Parameters: Document Handle, polyline object handle, coordinate of new point

Returns: -

6.5.9 void VpeSetBkgColor(long hDoc, COLORREF color)

Action: Sets the background color. The background color is the color inside of an object.

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

Returns: -

Default: White (but Transparent Mode is activated!).

6.5.10 void VpeSetTransparentMode(long hDoc, int on_off)

Action: Sets background color to transparent / not transparent. The background color is the color inside of an object.

Parameters: Document Handle, on_off (TRUE=1=ON or FALSE=0=OFF)

Returns: -

Default: ON

6.5.11 void VpeSetHatchStyle(long hDoc, int style)

Action: Sets the hatch style. All objects - except (of course) barcodes and images - can be hatched with the predefined windows hatch styles.

Parameters: Document Handle, style

Returns: -

Default: HS_NONE, which means NO hatching

Possible styles are:

6.5.12 void VpeSetHatchColor(long hDoc, COLORREF color)

Action: Sets the color of the hatching

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

Returns: -

Default: Black

6.5.13 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. The actual penstyle and background color is used.

Parameters: Document Handle, coordinates of the box

Returns: -

6.5.14 **long VpePolygon(long hDoc, POINT *p, unsigned int size)**

Action: Creates a polygon object with the current pen, background and hatchstyle.

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 VpeAddPolygonPoint().

Structure of the array <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 VpeAddPolygonPoint()).

6.5.15 **void VpeAddPolygonPoint(long hDoc, long p, int x, int y)**

Action: Adds a new point to the polygon object specified in <p>.

Parameters: Document Handle, polygon object handle, coordinate of new point

Returns: -

6.5.16 **void VpeEllipse(long hDoc, int x, int y, int x2, int y2)**

Action: Draws an ellipse or circle within the given rectangle.

x, y

x2, y2

Parameters: Document Handle, surrounding rectangle of the ellipse

Returns: -

See "" on page

6.5.17 **void VpePie(long hDoc, int x, int y, int x2, int y2, int begin_angle, int end_angle)**

Action: This function is identical to VpeEllipse(), but it draws a pie from begin_angle to end_angle.

Parameters: Document Handle, surrounding rectangle of the ellipse, begin_angle and end_angle (angles in 0.1 degree, clockwise)

Returns: -

6.6 Text Functions

These methods and properties deal with text formatting and output.

6.6.1 void VpeSelectFont(long hDoc, LPCSTR 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 in 0.1mm units!**)

Returns: -

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

Action: Sets the font-attributes - all at once.

Parameters: Document Handle, font-name, size in points (**not in 0.1mm units!**)

alignment:

ALIGN_LEFT = 0

ALIGN_RIGHT = 1

ALIGN_CENTER = 2

ALIGN_JUSTIFIED = 3

Returns: -

Default: ALIGN_LEFT, FALSE, FALSE, FALSE

6.6.3 void VpeSetAlign(long hDoc, int alignment)

Action: Sets the text alignment

Parameters: Document Handle

alignment: ALIGN_LEFT, ALIGN_RIGHT, ALIGN_CENTER, ALIGN_JUSTIFIED

Returns: -

Default: ALIGN_LEFT

6.6.4 void VpeSetBold(long hDoc, int bold)

Action: Set text bold on / off

Parameters: Document Handle, bold: TRUE or FALSE

Returns: -

Default: OFF

6.6.5 void VpeSetUnderlined(long hDoc, int underlined)

Action: Set text underlined on / off
Parameters: Document Handle, underlined: TRUE or FALSE
Returns: -
Default: OFF

6.6.6 void VpeSetItalic(long hDoc, int italic)

Action: Set text italic on / off
Parameters: Document Handle, italic: TRUE or FALSE
Returns: -
Default: OFF

6.6.7 void VpeSetTextColor(long hDoc, COLORREF color)

Action: Set the text color. This is the foreground color which also applies to barcodes.
Parameters: Document Handle, color coded as RGB triple
Returns: -
Default: Black

6.6.8 int VpeWrite(long hDoc, int x, int y, int x2, int y2, LPCSTR 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.
Parameters: Document Handle, position and dimensions, the string to output
Returns: The bottom y-coordinate generated by the output.

6.6.9 int VpeWriteBox(long hDoc, int x, int y, int x2, int y2, LPCSTR s)

Action: Same as VpeWrite, but pen- and box-settings are used. 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.
Parameters: Document Handle, position and dimensions, the string to output
Returns: The bottom y-coordinate generated by the output.

6.6.10 **int VpePrint(long hDoc, int x, int y, LPCSTR 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, the string to output

Returns: The bottom y-coordinate generated by the output.

6.6.11 **int VpePrintBox(long hDoc, int x, int y, LPCSTR s)**

Action: Like VpePrint, but pen- and box-settings are used. You need no box-coordinates. 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, the string to output

Returns: The bottom y-coordinate generated by the output.

6.6.12 **void VpeSetAutoBreak(long hDoc, int mode)**

Action: Controls, if and how text is handled, that overflows the bottom of the current output rectangle.

Parameters: Document Handle

mode:

AUTO_BREAK_ON: The Auto Page Break will only work, if $y_2 = VFREE$, or $y > VBOTTOMMARGIN$. Remaining text is broken onto the next page(s) at

$NewX = OldX$ and $NewY = \text{top of the output rectangle of the new page}$.

AUTO_BREAK_OFF: Same behaviour as AUTO_BREAK_ON (limited positioning / rendering to the bottom of the output rectangle is active), but remaining text is NOT broken onto next page(s). It is cut.

AUTO_BREAK_NO_LIMITS: Remaining text is NOT broken onto the next page(s), but it can be placed anywhere on the paper with no limits.

Returns: -

Default: AUTO_BREAK_ON

6.6.13 void VpeDefineHeader(long hDoc, int x, int y, int x2, int y2, LPCSTR 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: -

6.6.14 void VpeDefineFooter(long hDoc, int x, int y, int x2, int y2, LPCSTR 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: -

6.7 Picture Functions

These methods and properties deal with the import and presentation of image files.

6.7.1 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 of VpePicture() is VFREE: x2 OR y2.
- Parameters:** Document Handle, on_off (TRUE=1=ON or FALSE=0=OFF)
- Returns:** -
- Default:** ON

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

- Action:** Some bitmap-files do not contain the DPI resolution information 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. If the resolution information cannot be found in the image, VPE will use the default values you specify with this method.
- Parameters:** Document Handle, default x- and y-resolution in DPI
- Returns:** -
- Default:** 96 by 96 DPI which is the resolution of the screen.

6.7.3 void VpeGetPictureTypes(int with_filters, LPSTR s, int size)

- Action:** Returns a string with the extensions of all supported file-formats.
- Parameters:** Document Handle
- with_filters: TRUE = include the Adobe Image Import filter support, FALSE = exclude them
- s: the receive string with the type-list
- size: the maximum number of bytes, the string s can receive
- Returns:** A list of picture types in s of the following form: "*.WMF;*.BMP;*.TIF;*.JPG;*.PCX;*.TIF"

Currently the Adobe Image Import filters are only supported in the 16-bit version of VPEngine.

6.7.4 void VpePicture(long hDoc, int x, int y, int x2, int y2, LPCSTR 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. See also flag PIC_BESTFIT below.

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

Returns: -

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 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 PIC_KEEPIIMAGE 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 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 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.

```
#define PIC_BESTFIT           16
```

Keeps a bitmaps aspect and allows for the largest zoom within a given rectangle. i.e. autocalculates width and height and then decides, which to use in the final print. If the bitmap is wider than it is high, then X2 will be the limiter (and Y2 will be calculated regarding the bitmaps aspect). If the bitmap is higher than it is wide, then Y2 will be the limiter (and X2 will be calculated regarding the bitmaps aspect).

IMPORTANT: requires that all coordinates of the rectangle are given!

```
#define PIC_IN_FILE           32
```

When working with VpeOpenDocFile(), or VpeWriteDoc(), this flag forces the storage of the whole image to be directly in the file, instead of the pathname link (which will need much less space). This flag is useful, if the location of the image changes, or if you transport a document file to a system, where the image is not available.

Note, that for each VpePicture()-call with PIC_IN_FILE specified, the image is stored in the file. So, if you use the same bitmap on every page, it will be stored for every page separately in the file.

Currently images are stored in the file as decompressed bitmaps.

6.7.5 void VpePictureResID(long hDoc, int x, int y, int x2, int y2, int hInstance, unsigned int res_id, int flags)

Action: Takes the image out of the resource of
 the calling application, if hInstance = 0; or
 the loaded DLL, with hInstance = handle to the instance of the loaded DLL

WARNING: If the PIC_IN_FILE flag is NOT used, VPE stores the resource-link in it's document files. If you open this document file with another application that does not contain these resources with same ID's / names, then no images will be shown. Since DLL Instance-Handles change from load to load, VPE stores ALL images where hInstance is not null ALWAYS with the flag PIC_IN_FILE automatically set in the document files.

Parameters: Document Handle, rectangle, Instance-Handle, resource-id, flags (as explained for Picture)

Returns: -

6.7.6 void VpePictureResName(long hDoc, int x, int y, int x2, int y2, int hInstance, LPCSTR res_name, int flags)

Action: The same as VpePictureResID(), but instead of a numeric ID the resource is identified by name.
 Takes the image out of the resource of
 the calling application, if hInstance = 0; or
 the loaded DLL, with hInstance = handle to the instance of the loaded DLL

WARNING: If the PIC_IN_FILE flag is NOT used, VPE stores the resource-link in it's document files. If you open this document file with another application that does not contain these resources with same ID's / names, then no images will be shown. Since DLL Instance-Handles change from load to load, VPE stores ALL images where hInstance is not null ALWAYS with the flag PIC_IN_FILE automatically set in the document files.

Parameters: Document Handle, rectangle, Instance-Handle, resource-name, flags (as explained for Picture)

Returns: -

6.7.7 void VpePictureDIB(long hDoc, int x, int y, int x2, int y2, HGLOBAL hDIB, int flags)

Action: Takes the image from a handle - you must not delete the global memory block, but keep it unlocked. The flag PIC_IN_FILE is ALWAYS automatically set. Note, that this functions expects a handle to a DIB, not to a BITMAP.

This function was only implemented, to give you all possibilities in hands. Never use it to work with resources, since they are managed much better by VPE with the PictureRes-functions (locking, unlocking, conversion, etc.).

Parameters: Document Handle, rectangle, handle to DIB, , flags (as explained for Picture)

Returns: -

6.8 Barcode Functions

These methods and properties deal with the generation and presentation of the various barcode formats, VPEngine offers.

6.8.1 void VpeSetBarcodeParms(long hDoc, int top_bottom, int add_top_bottom)

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

Parameters: Document Handle

top_bottom: text on top (True) or on bottom of barcode (False)

add_top_bottom: add-on text on top (True) or on bottom of barcode (False)

Returns: -

Default: Bottom, Bottom

6.8.2 void VpeBarcode(long hDoc, int x, int y, int x2, int y2, int code_type, LPCSTR code, LPCSTR 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.

The color of bars and add-on text is set by "" on page .

Parameters: Document Handle, position and dimensions

code_type, one of the following constants (defined in vpecomon.h):

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

string with the code (i.e. "123456")

string with the add-on code, if add-on barcode type chosen, else NULL (0)

Returns: -

7 Release Notes

7.1 Release 1.1 (11/95)

Barcodes added:

- 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.

7.2 Release 1.2 (01/96)

- was only an internal release

7.3 Release 1.3 (03/96)

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_KEEPIIMAGE, VPE_PIC_DISCARD_DIB_DRAW, VPE_PIC_KEEP_DIB_PAGE

Page size is freely definable:

- 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:

- 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 (a VCL is currently under development and will soon be available)
- Borland Pascal

7.4 Release 1.4 (06/96)

Windows 95 look for toolbar buttons, better icons

Automatic Text Break:

The magic border is broken!!!

VPE is now able to render text to the bottom of the page (output rectangle) and to insert the remaining text automatically onto successive pages. (look at the new vpedemo.exe / .cpp)

For compatibility with previous versions and different behaviour, see new function: VpeSetAutoBreak()
(new embedded flags 'AutoBreakXXX', also see chapter "Dynamic Positioning")

VpeSetAutoBreak() Modes:

AUTO_BREAK_ON

The Auto Page Break will only work, for y2 = VFREE. Remaining text is broken onto the next page(s) at the same x-position and y = top of the output rectangle of the following page.

AUTO_BREAK_OFF

Same behaviour as AUTO_BREAK_ON (limited positioning / rendering), but remaining text is NOT broken onto next page(s).

AUTO_BREAK_NO_LIMITS

Remaining text is NOT broken onto the next page(s), but it can be placed anywhere on the paper with no limits. Use this switch to have the full compatible behaviour of VPE v1.4 with all previous versions.

"AutoBreak", "A"

"AutoBreakOff", "AO"

"AutoBreakNoLimits", "ANL"

The default is ON.

Rotation:

Text and images now can be rotated in 90 degree steps. This also affects barcodes, please see the changed parameter list for VpeSetBarcodeParms().

- New function VpeSetRotation().

(see chapter "Rotating Text, Images and Barcodes", new embedded flag 'Rotate')

Rotation is done clockwise. Angles are given in 1/10 degrees:

So the command for rotating by 90 degrees is: VpeSetRotation(hdoc, 900)

The x, y position will always be the same. It is not modified by rotating text / images.

Internally VPE computes the width and height of the object, and then rotates it.

So rotation is easier to use, if you work with width and height (negative signs for x2 and y2) instead of absolute values.

"Rotate", "Rot" <angle>

VPE needs LEON.DLL (LEON32.DLL) to rotate images.

The default is 0 degrees.

New Functions:

- VpeEllipse() to draw circles and ellipses.
- VpePie() to draw pies.
- VpePolygon() to draw filled polygons.
- VpeAddPolygonPoint() to add points to polygons.
- VpePictureResID() and VpePictureResName() take the image out of the resource of the calling application, if hInstance = 0; or the loaded DLL, with hInstance = handle to the Instance of the loaded DLL
WARNING: if the PIC_IN_FILE flag is NOT used, VPE stores the resource-link in it's document files.
If you open this document file with another application that does not contain these resources with same ID's / names, then no images will be shown. Since DLL Instance-Handles change from load to load, VPE stores ALL images where hInstance is not null ALWAYS with the flag PIC_IN_FILE automatically set.
- VpePictureDIB() takes the image from a handle - you must not delete the global memory block, but keep it unlocked. The flag PIC_IN_FILE is ALWAYS automatically set.
- VpeCenterPreview() centers the preview on the desktop or on a given window.
- VpeSetPrintOptions() allows setting the printing only for even, or for odd or for all pages. This preset is also active, if the user pushes the print-button in the toolbar.

New flags:

```
#define PRINT_ALL      0    // print all pages
#define PRINT_EVEN    1    // print only even pages
#define PRINT_ODD     2    // print only odd pages
```

The default is PRINT_ALL.

Hatching:

All objects - except (of course) barcodes and images - can be hatched now with the predefined windows hatch styles.

New functions:

- VpeSetHatchStyle() The default is HS_NONE
- VpeSetHatchColor() The default is COLOR_BLACK

(new embedded flags 'HSxxx' and 'HatchColor', see chapter "Embedded Flag Setting")

"HSNone", "HSN"

"HSHorizontal", "HSH"

"HSVertical", "HSV"

"HSFDiagonal", "HSFD"

"HSBDiagonal", "HSBD"

"HSCross", "HSC"

"HSDiagCross", "HSDC"

For styles see function VpeSetHatchStyle()

"HatchColor", "HC" <color-name>

Sets the hatchcolor. <color-name> is one of the color-strings above.

"HatchColorRGB", "HCRGB" <red> <green> <blue>

Sets the hatchcolor with RGB-values <red> <green> <blue>.

Example: "HCRGB 200 210 30"

Improved File Management:

- VpeWriteDoc() writes the current document into a file or appends the current document to another document stored in a file (this even works, if the current document was opened by VpeOpenDocFile(), e.g. it is itself a file!)
- VpeReadDoc() reads a documents from file into RAM and appends it to the current document (this even works, if the current document was opened by VpeOpenDocFile(), e.g. it is itself a file!)

These two new function allow much more flexibility in file handling than v1.3 and even merging of multiple documents. But be careful: if you used @PAGE in headers or footers, the numbers will not be correct, since they are already computed and stored in the document files. The best solution is, not to use the @PAGE function, but to store the documents without any page-number information. Then for merging with headers and footers, open a new document (file or RAM), define headers and footers, and then start to merge the documents with VpeReadDoc(). This will work very fine! Or, after you have merged all documents together in RAM, you can easily insert page numbers by calling VpeGotoPage() from 1 to VpePageCount() and each time insert the actual page number. But for this action you NEED to hold the document in RAM, because insertion of objects into existing pages of a file-document is impossible.

New flag PIC_BESTFIT:

Keeps a bitmaps aspect and allows for the largest zoom within a given rectangle.

i.e. autocalculates width and height and then decides, which to use in the final print. If the bitmap is wider than it is high, then X2 will be the limiter (and Y2 will be calculated regarding the bitmaps aspect). If the bitmap is higher than it is wide, then Y2 will be the limiter (and X2 will be calculated regarding the bitmaps aspect). IMPORTANT: requires that all coords. of the rectangle are given!

New flag PIC_IN_FILE:

In v1.3 only the pathnames of the images were stored as a reference in the files created by VpeOpenDocFile(). Now it is possible, to store the image inside the file (this flag is automatically set, when using VpePictureDIB()). Currently images are stored in the file as decompressed bitmaps.

Bitmaps:

Some programs write curious BMP file formats. VPE's format checking is now more relaxed and even bad formats (regarding the standard) can be read.

VPE_PIC_xyz flags renamed:

All VPE_PIC_xyz flags have been renamed to PIC_xyz.

The following functions have been renamed:

- VpeSetDefaultOutputRect() into VpeSetDefOutRect()
- VpeSetOutputRect() into VpeSetOutRect()
- VpeGetOutputRect() into VpeGetOutRect()

The following functions have been added for use with interpreters, who have problems with the RECT structure:

The suffix "SP" stands for "Single Parameters".

- VpeSetDefOutRectSP()
- VpePreviewDocSP()

For VpeSetOutRect() and VpeSetPosRect() you can use VpeSet().

For VpeGetOutRect() and VpeGetPosRect() you can use VpeGet().

For the following functions parameter types have been changed for use with interpreters, which do not support all datatypes:

- VpePrintDoc(long hDoc, int with_setup)
previously: BOOL with_setup
- void VpeGetPictureTypes(int with_filters, LPCSTR s, int size)
previously: returned String-Pointer; parameter BOOL with_filters only;
now receives string in s, size is the maximum size of the string
- VpeSetBarcodeParms(long hDoc, int top_bottom, int add_top_bottom)
previously: int orientation, BYTE top_bottom, BYTE add_top_bottom

The parameter "orientation" has been removed, to gain a consistent interface (due to the new function VpeSetRotation()). Use this function!

Flags have been defined for VpeSetupPrinter(), their values didn't change:

- PRINTDLG_NEVER
- PRINTDLG_ONFAIL
- PRINTDLG_ALWAYS

Fixed problem, that VPE didn't print under Win95 when called from Visual-C.

New interfaces:

- FoxPro / Visual FoxPro
- Demo for use with Micro Focus COBOL for Windows

8 Index