IDEAL Software

Programmer's Manual v4.00

VIRTUAL PRINT ENGINE

DLL Reference

DLL Reference
Virtual Print Engine

Contents

1 Introduction

1.1 Introduction

This manual is the complete **reference** for all events, properties and methods provided by the VPE DLL. It does not explain any programming techniques.

The techniques on using VPE are explained in the "Programmer's Manual" chapter 3 "Programming Techniques".

1.1.1 Datatypes

The API (Application Programming Interface) of VPE defines two important datatypes:

- *VpeHandle*, which is used as a handle to documents and other types of internal objects. On 32-bit platforms, this is a 32-bit integer and on 64-bit platforms this is a 64-bit integer.
  (In fact it is typedef'd as a void pointer "void *").
- *VpeCoord*, which is used to specify coordinates as parameters to functions. *VpeCoord* is of type *double*.

2 Messages Generated by VPE - DLL

2.1 Messages Generated by VPE - DLL

VPE sends several notification messages to your application, so you have always total control about what's happening. The messages are sent via the WIN API methods "SendMessage()" or PostMessage(). They are sent to VPE's parent window, which you supplied in the first parameter of the VpeOpenDoc() call.

If you want to execute VPE on servers or in batch jobs where your application is windowless, you can install a message callback function. In this case VPE will not send events via SendMessage() or PostMessage() to your application window, instead it will call your callback function.
For details, see VpeSetMsgCallback().

The messages generated by VPE (e.g. VPE_DESTROYWINDOW, etc.) are globally registered in the windows system to avoid conflicts with other controls and applications. Therefore it is unknown, what value equals each message and you need to map the messages with the method VpeMapMessage() in your window-procedure.

If your tool is not able to process dynamically window-messages, or if you want to gain the old behavior of VPE prior to v3.0 (having fixed values for each message), use the new flag VPE_FIXED_MESSAGES in VpeOpenDoc(). If you use VPE_FIXED_MESSAGES, VPE will directly send the VPE_xyz message constants to the window procedure of your parent window, instead of sending the globally registered messages. For a detailed discussion on how to handle messages see VpeMapMessage().

For the names of the registered messages, please see Names of the Registered Window Messages.

For information on how to use the registered messages with Visual C++ and the Microsoft Foundation Classes (MFC), please see How to handle VPE Events using the MFC.

NOTE: You may not call VpeCloseDoc() while processing any event fired by VPE - except it is explicitly said in the decription of a particular event that it is allowed.

2.2 VPE_DESTROYWINDOW

[Windows platform only]

Is sent when the preview window was destroyed - for example closed by the user - and EnableAutoDelete is True (the default).

The document is also closed (removed from memory).

VPE_DESTROYWINDOW
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
unused

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

Do not call any VPE function when processing this event. The document is already closed and not accessible.

This message is sent, if EnableAutoDelete is True and the user closes the preview. Otherwise the message VPE_CLOSEWINDOW is sent.

Your application should return zero if it processes this message.

2.3 VPE_CANCLOSE

[Windows platform only]

VPE requests for confirmation of your application if the preview can be closed.

VPE_CANCLOSE
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
unused

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

Your application should return:

Value	Description
1	no, the preview may not be closed
0	Otherwise

2.4 VPE_CLOSEWINDOW

[Windows platform only]

Is sent in case the preview window was closed - for example by the user - and EnableAutoDelete is False, which means, that the document is not destroyed if the preview is closed.

VPE_CLOSEWINDOW
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
unused

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:
Your application should return zero if it processes this message.

See also:
VPE_DESTROYWINDOW

2.5 VPE_BEFORE_OPEN_FILE

[Windows platform only]

Is sent when the user clicked the Open File button in the toolbar (or pushed the corresponding key) and before the open file dialog is shown.

VPE_BEFORE_OPEN_FILE
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
unused

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

Your application should return:

Value	Description
0	continue operation
1	cancel operation, i.e. do not show the open dialog

Cancelling the operation allows you to display your own open dialog and to import or create your own document, for example from a database via a memory stream.

See also:

VpeGetOpenFileName

2.6 VPE_AFTER_OPEN_FILE

[Windows platform only]

Is sent after the file open operation has been completed.

VPE_AFTER_OPEN_FILE
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
the error code of the operation, which is identical to LastError. For example, if the user clicked onto the Cancel-Button in the open file dialog, wParam will be VERR_CANCELLED.

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:
Your application should return zero if it processes this message.

See also:
VpeGetOpenFileName

2.7 VPE_BEFORE_SAVE_FILE

[Windows platform only]

Is sent when the user clicked the Save File button in the toolbar (or pushed the corresponding key) and before the save dialog is shown.

VPE_BEFORE_SAVE_FILE
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
unused

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

Your application should return:

Value	Description
0	continue operation
1	cancel operation, i.e. do not show the save dialog

Cancelling the operation allows you to display your own save dialog and to export your own document, for example to a memory stream and from there to a database.

See also:

VpeGetSaveFileName

2.8 VPE_AFTER_SAVE_FILE

[Windows platform only]

Is sent after the file save operation has been completed.

VPE_AFTER_SAVE_FILE
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
the error code of the operation, which is identical to LastError. For example, if the user clicked onto the Cancel-Button in the save file dialog, wParam will be VERR_CANCELLED.

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:
Your application should return zero if it processes this message.

See also:
VpeGetSaveFileName

2.9 VPE_HELP

[Windows platform only]

Is sent if the property EnableHelpRouting is True and the user clicked the Help-Button in the toolbar or pushed the corresponding key (F1 by default).

This is one of the few events where VpeCloseDoc() may be called while processing it.

VPE_HELP
 WPARAM *wParam*
 LPARAM *lParam*

Parameters:

wParam
 unused

lParam
 contains the document-handle, so you can determine which document has sent the message

Remarks:
 Your application should return zero if it processes this message.

2.10 VPE_AUTOPAGEBREAK

Is sent, when a Auto Page Break occurred (see "Automatic Text Break" in the Programmer's Manual). The engine is on the new page. This is either a newly generated page, or an already existing page (if the text was inserted on a page, that has already pages following).

This gives you additional control over the layout. When the event is fired, VPE already moved to the next page (or generated one) and you can modify the Output Rectangle (NOT the Default Output Rectangle) to control the layout, or insert manually Headers and Footers, etc. (see "Headers and Footers" in the Programmer's Manual).

VPE_AUTOPAGEBREAK
  WPARAM *wParam*
  LPARAM *lParam*

Parameters:

wParam
  unused

lParam
  contains the document-handle, so you can determine which document has sent the message

Remarks:

Your application should return zero if it processes this message.

If your application is not able to receive events, or if you don't want to process this event for some reason, you can test for AutoBreak (this means, check if an AutoBreak occurred after inserting a text object) with the following technique:

Store the current page-number in a variable. After the output, compare this variable to the current page-number. If it is different, an AutoBreak had occurred.

Code example:

```
n = VpeGetCurrentPage(hDoc)    // remember the current page number
VpePrint(hDoc, 1, 1, "... very long text ...")
if n <> VpeGetCurrentPage(hDoc)
   catched auto break!
   The formula "VpeGetCurrentPage(hDoc) - n" returns the number of
   automatically generated pages.
```

NOTES:
If you are inserting text objects into the document while processing this event, make sure they will not cause again an Auto Page Break event - otherwise this will cause an endless recursion, ending up in a stack overflow with a GPF.
To avoid Auto Page Breaks, set AutoBreakMode = AUTO_BREAK_NO_LIMITS.

If you are modifiying any properties, as for example the AutoBreakMode or FontSize etc., it will not be reset when you code exits the event handler. If you are changing properties, you need to save and restore them by code yourself. **Do this by calling VpeStoreSet() in the beginning of you event handler and by calling VpeUseSet() followed by VpeRemoveSet() at the end.**
Example: if a Print() statement in your main code causes an AutoBreak and if your AutoBreak Handler sets the PenSize to something different than zero, then the rest of the text which is output by the previous Print() command of your main code will have a box around it.

Solution: at the very first beginning of your AutoBreak Handler call StoreSet() and on the last line call UseSet() and then RemoveSet(). Do this only if required, you can also check which single properties have been changed and change them back, because StoreSet() and UseSet() eat performance!

YOU MAY NOT USE NoHold of the embedded flags in the AutoBreak-Event Handler. This will cause an internal recursion and overwrite settings done in the main code. Explanation: When using the flag NoHold in your main code in a Print(Box)() or Write(Box)() statement which causes an AutoBreak, the current settings are stored in a temporary memory block (to be restored after the command has been executed). Afterwards your AutoBreak Event-Handler is executed and if you use the NoHold Flag again, the temporary memory block is overwritten, so that its initial settings are lost.

2.11 VPE_PRINT

[Windows platform only]

Is sent to inform the application about the several stages during the printing process.

VPE_PRINT
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam

Action	Value	Comment
PRINT_MSG_ABORT	0	User aborted while printing
PRINT_MSG_START	1	Print started
PRINT_MSG_END	2	Print ended
PRINT_MSG_SETUPABORT	3	User aborted Setup-Dialog
PRINT_MSG_SETUPSTART	4	Setup-Dialog started
PRINT_MSG_SETUPEND	5	Setup-Dialog ended

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

Do not call VpeCloseDoc() when processing this event. You would terminate a module that is working.

Your application should return **PRINT_ACTION_OK** (zero) if it processes this message, except for wParam = PRINT_MSG_SETUPSTART, where your application may return in addition **PRINT_ACTION_ABORT** (= 1) to abort the print job. PRINT_MSG_SETUPSTART is sent, when the user clicked the print button in the preview (or pushed the corresponding key). You have the option to abort the job, for you can then create and print internally a new document which is completely different to the preview.

Another use for the PRINT_MSG_SETUPSTART message is, to pre-initialize the Device Control Properties at this stage, before the printer setup dialog will be shown to the user.

2.12 VPE_PRINT_NEWPAGE

[Windows platform only]

Is sent only while printing exactly before printing a new page. The event is useful to change the Device Control Properties on-the-fly during printing. You may change all properties, **except the device itself**.

VPE_PRINT_NEWPAGE
  WPARAM *wParam*
  LPARAM *lParam*

Parameters:

wParam
  current page number that will be printed

lParam
  contains the document-handle, so you can determine which document has sent the message

Remarks:

Your application should return **PRINT_ACTION_OK** (zero) if it processes this message without changing a printing device's properties.

If a *Device Control Property* was changed, your application must return **PRINT_ACTION_CHANGE** (= 1).

- If you change the *Device Control Properties* during the print job, you must reset them to the original values when the print job has finished (see VPE_PRINT: with wParam = PRINT_MSG_ABORT or wParam = PRINT_MSG_END).
- Changing the properties (like DevBin, DevOrientation, DevPaperFormat, etc.) during the print job doesn't work with some (buggy) printer drivers, for example some fax drivers and the HP4 M Plus driver (the HP4 PS driver should work with the HP4 M Plus printer!).
- Some properties and methods don't work with some printer drivers. For example "DevTTOption" doesn't work with our HP4 and HP5 printer drivers on WfW 3.11 and NT 3.51, but it works with both drivers on Win95. This is a driver problem.
- Printer drivers are manufactured by vendors independent of IDEAL Software; we make no warranty, implied or otherwise, regarding these product's performance or reliability.
- Win32s is not officially supported by VPE. The *Device Control Properties* do not work under Win32s.

Known Problems:

Changing any Device Control Properties during the print job disables Duplex Printing on PCL Printers

SYMPTOMS

When you change a Device Control Property during the print job, it appears to disable Duplex (double-sided) printing when the target printer is a (Hewlett Packard) PCL printer.

CAUSE

PCL printers treat a change in paper size as a new print job that requires the printer to be initialized. This causes the printer to eject any page that is currently in the printer. The PCL printer drivers for Windows assume that the page size has been changed when a Device Control Property is changed during the print job, unless the orientation of the page has changed.

RESOLUTION

To prevent having a page ejected when changing a Device Control Property during the print job, make sure that the function is called only between individual sheets of paper. Changing a Device Control Property before printing odd-numbered pages is sufficient for most applications that use duplex printing. However, some applications require that you change the page orientation on a page-by-page basis. In this case, you can change Device Control Properties between individual sheets of paper if the orientation has changed.

STATUS

This behavior is by design.

MORE INFORMATION

Note that when this problem occurs the print job continues and the sheets of paper are passed through the printer's duplexer, but the sheets are only printed on one side.

Because of the page size initialization requirement for PCL printers, Windows PCL drivers treat the change of Device Control Properties differently. These drivers allow only the orientation to change between the front and back pages of a sheet of paper. This means that the change of a Device Control Property will eject the page unless the orientation (and only the orientation) has changed from the previous page. Returning PRINT_ACTION_CHANGE although no Device Control Property has been changed causes the printer to eject the page.

Returning PRINT_ACTION_CHANGE although no Device Control Property has been changed is unnecessary. By doing so, you risk having a page ejected from the printer, which has a high probability of occurring.

See also:

VPE_PRINT_DEVDATA

2.13 VPE_PRINT_DEVDATA

[Windows platform only]

Is sent only while printing, exactly before printing a new page and immediately after VPE_PRINT_NEWPAGE has been sent. The only use for this event is to call VpeDevSendData() in response.

VpeDevSendData() enables your application to send escape sequences to the printing device. So it is possible to select for example an **output paper bin** by code (or whatever other functionality is provided by the connected printer).

VPE_PRINT_DEVDATA
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
current page number that will be printed

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:
Your application should return **PRINT_ACTION_OK** (zero) if it processes this message without calling VpeDevSendData(). If it called VpeDevSendData(), your application must return **PRINT_ACTION_CHANGE** (= 1).

See also:
VPE_PRINT_NEWPAGE

2.14 VPE_BEFORE_MAIL

[Windows platform only]

Is sent, when the user clicked the eMail-button in the preview (or pushed the corresponding key). The e-mail was not sent yet, therefore your application has now the option to set receivers, attachments, etc. by code.

VPE_BEFORE_MAIL
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
unused

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:
Your application should return:

Value	Description
0	continue operation
1	cancel operation i.e. do not mail the document

Cancelling the operation allows you to display your own mail dialog and/or to execute your own mailing code.

See also:
E-Mail Functions

2.15 VPE_AFTER_MAIL

[Windows platform only]

Is sent, after the user clicked the eMail-button in the preview (or pushed the corresponding key) and the e-mail has already been sent.

VPE_AFTER_MAIL
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
the status of the e-mail action, one of the VERR_xyz error codes

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:
Your application should return zero if it processes this message.

See also:
E-Mail Functions

2.16 VPE_OBJECTCLICKED

[Windows platform only, Professional Edition and above]

A clickable object with an assigned ObjectID has been clicked with the mouse.

This is one of the few events where VpeCloseDoc() may be called while processing the event.

VPE_OBJECTCLICKED
  WPARAM *wParam*
  LPARAM *lParam*

Parameters:

wParam
  unused

lParam
  contains the document-handle, so you can determine which document has sent the message

Remarks:
  Retrieve the ObjectID with VpeGetObjectID(lParam).

  Your application should return zero if it processes this message.

See also:
  Clickable Objects

2.17 VPE_UDO_PAINT

[Windows platform only, Professional Edition and above]

Is sent as a notification from a User Defined Object (UDO). The object needs to be painted to the output device.

VPE_UDO_PAINT
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
unused

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

See the description of the User Defined Objects for information on how to process this event.

Your application should return zero if it processes this message.

2.18 VPE_CTRL_AFTER_ENTER

[Windows platform only, Interactive Edition and above]

A Control received the focus.
Normally, this event is sent if the user presses the Tab or Backtab key, or clicks with the mouse onto a control. This event can be used to re-format the content of a control.

VPE_CTRL_AFTER_ENTER
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
contains the object-handle of the object, which fired the event

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

Your application should return zero if it processes this message.

See Also:

Interactive Documents in the Programmer's Manual.

2.19 VPE_CTRL_CAN_EXIT

[Windows platform only, Interactive Edition and above]

The user wishes to remove the focus from the currently focused Control. In response to this event your application can evalute the value of the Control and decide, whether the current value is valid and the Control may lose the focus or not.

VPE_CTRL_CAN_EXIT
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
contains the object-handle of the object, which fired the event

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

Your application should return:

Value	Description
0	yes, the control may loose the focus
1	no, the control may not loose the focus

See Also:

Interactive Documents in the Programmer's Manual.

2.20 VPE_CTRL_AFTER_EXIT

[Windows platform only, Interactive Edition and above]

The currently focused Control lost the focus.
Normally, this event is sent if the user presses the Tab or Backtab key, or clicks with the mouse onto another control. When receiving this message, it is possible for you to force the focus to be set explicitly to a specific control by calling SetFocusControlByName() or SetFocus(). It is also possible to enable and disable other controls of the current form while processing this event.
In addition this event can be used to re-format the content of a control.

VPE_CTRL_AFTER_EXIT
    WPARAM *wParam*
    LPARAM *lParam*

Parameters:

wParam
    contains the object-handle of the object, which fired the event

lParam
    contains the document-handle, so you can determine which document has sent the message

Remarks:

    Your application should return zero if it processes this message.

See Also:

    Interactive Documents in the Programmer's Manual.

2.21 VPE_CTRL_AFTER_CHANGE

[Windows platform only, Interactive Edition and above]

A ***Control*** changed its value, i.e. the content of a Control was edited by the user or the value of an associated Field was changed by code.
Evaluating this event means, that your application is informed about every single keystroke or mouse-click, which modifies a Control's content.
If you are working with Fields that are associated with controls - as recommended - your application should not take care of this event.
This is one of the few events where CloseDoc() may be called while processing the event.
The event is not fired, if you set the value of a ***Control*** by code.

VPE_CTRL_AFTER_CHANGE
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
contains the object-handle of the object, which fired the event

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

If you change the value of a Control by code, the AfterControlChange event is not fired. But if the Control is associated with a Field, the event AfterFieldChange is fired.
Vice versa, if you change the value of a Field by code, the AfterFieldChange event is not fired, but any Controls associated with the Field will fire the event AfterControlChange.

Your application should return zero if it processes this message.

See Also:

Interactive Documents in the Programmer's Manual.

2.22 VPE_FIELD_AFTER_CHANGE

[Windows platform only, Interactive Edition and above]

A *Field* (not a control!) changed its value, because the associated Control was edited by the user or the content of any associated Control was changed by code.
This event is very interesting, because a Field will change its value each time the user makes a change. Evaluating this event means, that your application is informed about every single keystroke or mouse-click, which modifies a Control's content.
This is one of the few events where CloseDoc() may be called while processing the event.
The event is not fired, if you set the value of a ***Field*** by code.

VPE_FIELD_AFTER_CHANGE
WPARAM *wParam*
LPARAM *lParam*

Parameters:

wParam
contains the object-handle of the Field object, which fired the event

lParam
contains the document-handle, so you can determine which document has sent the message

Remarks:

If you change the value of a Control by code, the AfterControlChange event is not fired. But if the Control is associated with a Field, the event AfterFieldChange is fired.
Vice versa, if you change the value of a Field by code, the AfterFieldChange event is not fired, but any Controls associated with the Field will fire the event AfterControlChange.

Your application should return zero if it processes this message.

See Also:

Interactive Documents in the Programmer's Manual.

3 Management Functions

3.1 Management Functions

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

3.2 VpeOpenDoc

Creates a new document with one initial blank page.

VpeHandle VpeOpenDoc(
    HWND *hwndParent*,
    LPCSTR *title*,
    long *flags*
)

HWND hwndParent
    a window of your calling application that will be the parent window of the VPE Preview Window. VPE exchanges messages with it. If the Preview is embedded, this will also be the host window of the Preview.
    This parameter may be NULL, for example to use VPE in windowless applications like server processes or batch jobs. In this case you can install a message callback function, in order to receive events generated by VPE. For details, see VpeSetMsgCallback().
    For non-Windows platforms this parameter must be NULL.

LPCSTR title
    title of the Preview Window. The title is also used by VPE to compose the default JobName of the print job.

long flags
    controls the style of the Preview and the behavior of VPE (see below)

Returns:
    The handle (=identifier) to the virtual document. This handle has to be provided to all other VPE calls. In case of an error, NULL (0) is returned.
    On 32-bit platforms the handle is a 32-bit integer, on 64-bit platforms it is a 64-bit integer.

Remarks:
    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. 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 KB of memory. This is really low, but 100 pages need about 1MB of memory.

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

A VPE document can exist without showing a preview. But if a preview is shown, the document is closed and removed from memory by default, when the preview is closed by the user or when the parent window is closed. If you call VpeEnableAutoDelete(hDoc, false), the document is not closed when the preview is closed.

On non-Windows platforms you must always call VpeCloseDoc() to remove a document from memory.

Windows platform: Embedding the preview into a host 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:

1. Use the flag VPE_EMBEDDED in VpeOpenDoc()
2. In the window-procedure of your parent window (where the preview shall be embedded), give the following response to the WM_SIZE message:

   ```
   MoveWindow(VpeGetWindowHandle(hDoc), 0, 0, LOWORD(lParam), HIWORD(lParam),
              TRUE)
   ```

   this will size the VPE preview window accordingly with the parent window.
3. In the window-procedure of the parent window, respond to the WM_SETFOCUS message with:

   ```
   SetFocus(VpeGetWindowHandle(hDoc))
   ```

   this will route the keyboard and mouse-wheel 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 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.

Flags:

If you want to customize the preview or behavior of VPE you can specify one or more flags. VPE has been configured in a way that you can use 0 (zero) as flags-value for a default behavior.

Constant	Value	Description
VPE_NO_TOOLBAR	1	Toolbar NOT visible
VPE_NO_PRINTBUTTON	8	Print-Button invisible, VpePrintDoc() works
VPE_NO_MAILBUTTON	16	Mail-Button invisible
VPE_NO_SCALEBTNS	32	No Scale Buttons in Toolbar
VPE_GRIDBUTTON	64	Grid Toolbar-Button visible
VPE_NO_MOVEBTNS	128	Move Buttons invisible
VPE_NO_HELPBUTTON	256	Help-Button invisible

VPE_NO_INFOBUTTON	512	Info-Button invisible
VPE_NO_USER_CLOSE	1024	User can't close VPE: Close-Button INVISIBLE and Sys-Menu disabled (if not embedded) - VpeCloseDoc() works!
VPE_NO_STATBAR	2048	Statusbar invisible
VPE_NO_PAGESCROLLER	4096	Page Scroller in Statusbar invisible
VPE_NO_STATUSSEG	8192	Status Segment (where you can show messages and the Progress Bars) in Statusbar invisible
VPE_NO_RULERS	16384	Rulers invisible
VPE_EMBEDDED	32768	Preview is made child window of parent window specified in VpeOpenDoc()
VPE_DOCFILE_READONLY	65536	Document file is opened with read-only permission
VPE_FIXED_MESSAGES	131072	Messages are not registered by the Windows System
VPE_NO_OPEN_BUTTON	262144	No File Open Button
VPE_NO_SAVE_BUTTON	524288	No File Save Button

The flag **VPE_DOCFILE_READONLY:** see VpeOpenDocFile()

The flag **VPE_FIXED_MESSAGES:** if you use VPE_FIXED_MESSAGES, VPE will directly send the VPE_xyz message constants to the window procedure of your parent window, instead of sending the globally registered messages. For a detailed discussion on how to handle messages see VpeMapMessage().

Example:

```
VpeHandle hDoc;
hDoc = VpeOpenDoc(hwndParent, "Report", VPE_NO_STATUSSEG + VPE_NO_RULERS);
if (hDoc == NULL)
   return; // error
VpePreviewDoc(hDoc, 0, VPE_SHOW_MAXIMIZED);
```

Creates a document where the preview will have the title "Report". In this example the preview will have no status segment and no rulers. Note that the preview is not automatically shown after calling VpeOpenDoc() - it just creates the document, which is then ready for inserting objects and adding pages. The preview is shown by calling VpePreviewDoc().

Community Edition:

For the Community Edition the GUI elements can not be changed, i.e. the preview has always a toolbar, rulers, scrollers and a statusbar. The open, save, grid and help buttons are not available, the other buttons of the toolbar can not be made invisible. In addition embedding the preview is not supported by the Community Edition. The only flags accepted by the Community Edition are VPE_DOCFILE_READONLY and VPE_FIXED_MESSAGES.

3.3 VpeOpenDocFile

The same as VpeOpenDoc(), but instead of storing all document pages in memory, only the current page is held in memory. All other pages are swapped to a VPE document file. This implies minimum memory usage at very high performance and allows to create huge documents. VPE's file swapping is VERY fast.

Even for file-based documents you can add new pages to the end of a document at any point in time.

Editions below the Professional Edition: after a page has been swapped to file, you can not modify the page, i.e. add new objects to it.

The Professional Edition and higher allow to add new objects to pages which have already been written to file and to clear, insert and delete pages at any position in a document file.

A page is swapped to file after:

- Adding a new blank page by calling VpePageBreak()
- Moving to a different page by calling VpeGotoPage()

For details about creating and using VPE document files, please see the "Programmer's Manual", chapter "Programming Techniques", subchapter "VPE Document Files".

VpeHandle VpeOpenDocFile(
HWND *hWndParent*,
LPCSTR *file_name*,
LPCSTR *title*,
long *flags*
)

HWND hwndParent
a window of your calling application that will be the parent window of the VPE Preview Window. VPE exchanges messages with it. If the Preview is embedded, this will also be the host window of the Preview.
This parameter may be NULL, for example to use VPE in windowless applications like server processes or batch jobs. In this case you can install a message callback function, in order to receive events generated by VPE. For details, see VpeSetMsgCallback().
For non-Windows platforms this parameter must be NULL.

LPCSTR file_name
name of file to open or create
it is very important that all VPE document files have the suffix ".vpe". Always use this suffix, because "VPE View" (the document browser) is associated with this suffix.

LPCSTR title
title of the Preview Window

long flags
controls the style of the Preview and the behavior of VPE (see VpeOpenDoc())

Returns:
The handle (=identifier) to the virtual document. This handle has to be provided to all other VPE calls. In case of an error, NULL (0) is returned.
On 32-bit platforms the handle is a 32-bit integer, on 64-bit platforms it is a 64-bit integer.

You can check for error conditions - for example, if there is not enough free space left on disk for the SwapFile - by testing the property VpeGetLastError() after calling VpeOpenDocFile() and each time after calling VpePageBreak().

Remarks:

When you create a new document with this function, compression is always activated.

Example:

```
VpeHandle hDoc;
long count;
hDoc = VpeOpenDocFile(hwndParent, "c:\docs\report1.vpe", "Example", 0);
if (hDoc == NULL)
   return; // error
count = VpeGetPageCount(hDoc);
VpePageBreak(hDoc);
VpePrint(hDoc, 1, 1, "Added a new page.");
VpePreviewDoc(hDoc, NULL, VPE_SHOW_NORMAL);
```

If the VPE document file "c:\docs\report1.vpe" is already existing, the file will be opened and the first page is read into memory. If the document file is not existing, VPE will create it with an initial blank page. The variable "count" is assigned the number of pages the document contains. VPE will add a new page at the end of the document and insert the text "Added a new page." there. Then the preview is shown.

The flag **VPE_DOCFILE_READONLY:**

If you use this flag, the Document file is opened with read-only permission. A VPE document file can not be created if this flag is specified. You can only use it to open an existing file for read-only purposes.

If a VPE document file is opened for read / write (the default), no other application can open this file - even if it tries to open the file with read-only permission. Multiple applications can open the file at the same time only, if all applications use ReadOnly = True.

Example:

```
VpeHandle hDoc;
long count;
hDoc = VpeOpenDocFile(hwndParent, "c:\docs\report1.vpe", "Example",
                      VPE_DOCFILE_READONLY);
if (hDoc == NULL)
   return; // error
count = VpeGetPageCount(hDoc);
VpeGotoVisualPage(hDoc, count);
VpePreviewDoc(hDoc, NULL, VPE_SHOW_NORMAL);
```

If the VPE document file "c:\docs\report1.vpe" is not existing, VpeOpenDocFile() will return NULL (= error). Otherwise the file will be opened in read-only mode and the first page is read into memory. The variable "count" is assigned the number of pages the document contains. The preview will show the last page contained in the document.

When calling VpeOpenDocFile(), the JobName is set automatically to the file name.

See Also:

VpeWriteDoc() and VpeReadDoc()

3.4 VpeCloseDoc

Closes the specified document (and also the preview, if open).

int VpeCloseDoc(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:

Value	Description
True	Ok
False	couldn’t close, because the document is currently printed

Remarks:
  You may call this method with hDoc = NULL. In this case the function will return True. This is the only method of VPE where hDoc may be NULL.

Note: when a window which contains the VPE Control is being closed, it is **very important** that you call *VpeCloseDoc().* You need to call *VpeCloseDoc()* exactly at the time when receiving the event which asks for confirmation if the form can be closed. (For Windows applications this is the **WM_CLOSE** event.)

In case *VpeCloseDoc()* should return False in that moment, you need to cancel the process of closing the window, i.e. the window must not be closed.

Example for C/C++:

```
case WM_CLOSE:
   if ( !VpeCloseDoc(hDoc1) || !VpeCloseDoc(hDoc2))
   {
         Msg("The application cannot terminate until all jobs have finished
              printing.");
         return 0;
   }
   DestroyWindow(hwnd);
   return 1;
```

In the above examples the window controls two VPE documents named "hDoc1" and "hDoc2". For both documents it is checked whether their previews can be closed or not.

3.5 VpeSetMsgCallback

Installs a message callback function. If you install a callback function, VPE will not send events via SendMessage() or PostMessage() to your application window, instead it will call your callback function. This is very useful, if you want to run VPE on servers or in batch jobs in windowless applications.

void VpeSetMsgCallback(
VpeHandle *hDoc*,
VPE_MSG_CALLBACK *callback*
)

VpeHandle hDoc
Document Handle

VPE_MSG_CALLBACK callback
The pointer to your application's callback function. In case of an event, VPE will call this function with one of the events listed in the chapter "Messages Generated by VPE - DLL".

Remarks:
The callback function VPE_MSG_CALLBACK is defined in vpiface.h as follows:

```
typedef LRESULT (EXPO *VPE_MSG_CALLBACK)(UINT Msg, WPARAM wParam, LPARAM
lParam);
```

The parameters are equal to a normal Windows-Procedure. The parameter *Msg* contains the VPE message code, like for example VPE_DESTROYWINDOW. The message codes are not transformed, i.e. you must not call VpeMapMessage().

The values which should be returned by your callback function depend on the VPE message code. If not otherwise explained for each individual message, zero should be returned.

3.6 VpeSetEditProtection

[Professional Edition and above]

The Visual Designer *dycodoc* can read and edit VPE Document files which have been created with the VPE Professional Edition or any higher edition.

If you want to protect your VPE Document files so they can not be read by *dycodoc*, call this function. In both cases - either if a document was opened with VpeOpenDocFile() or if you call VpeWriteDoc() - the created VPE document file will be protected.

Once you have called this function, it is impossible to unprotect the document.

[Interactive Edition only:]

If the EditProtection is enabled, the interactive objects stored in VPE Document files are not editable within VPE, nor *VPEView*.

void VpeSetEditProtection(
VpeHandle *hDoc*,
int *reserved*
)

VpeHandle hDoc
Document Handle

int reserved
You must assign the value "1" to this property! Different values are reserved for future extensions. Do not assign any different value!

Default:
0 = the current document is not protected from being edited with *dycodoc*

Remarks:
Once you have called this function, it is impossible to unprotect the document.

For security reasons, *dycodoc* can not read VPE Document files which have been created with any release prior to VPE v3.20.

Example:

```
VpeSetEditProtection(hDoc, 1);
VpeWriteDoc(hDoc, "my_file.vpe");
```

Activates the edit protection and writes the current document as protected file to disk.

3.7 VpeGetEditProtection

[Professional Edition and above]

Retrieves the edit protection status from the current VPE Document.

int VpeGetEditProtection(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
0	the current document is not protected from being edited with *dycodoc*
1	the current document is protected

Example:

```
VpeSetEditProtection(hDoc, 0);
VpeWriteDoc(hDoc, "my_file.vpe");
```

Activates the edit protection and writes the current document as protected file to disk.

3.8 VpeLicense

Licences the given document within the engine as full version, so the demo banners disappear.

void VpeLicense(
VpeHandle *hDoc*,
LPCSTR *serial1*,
LPCSTR *serial2*
)

VpeHandle hDoc
Document Handle

LPCSTR serial1, serial2
the two serial strings provided by IDEAL Software when you license VPE

Example:

If the license key has the following form:

VPE-A1234-123456
ABCD-EFGH

The the method is called with:

```
VpeLicense(hDoc, "VPE-A1234-123456", "ABCD-EFGH")
```

Remarks:

If you obtained additional license keys for add-on products, call this method for each License Key separately. It is necessary that the VPE module is licensed **first** before any add-on module is licensed. Otherwise the licensing of the add-on modules will fail.

If you are using a Server License, it is necessary that you call License() for all available add-on modules first, before setting the property *EnableMultiThreading* = *true*.

3.9 VpeEnableMultiThreading

By default, VPE is not thread-safe. If you are using VPE in a multi-threaded environment, set the property *EnableMultiThreading = True*, to activate the thread-safe code of VPE.

On some platforms it might be required to purchase and install a special server license (for each server), before this property can be used.

void VpeEnableMultiThreading(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	VPE's thread-safe code is activated
False	VPE's thread-safe code is not active

Default:
False

Remarks:
VPE documents must be created and used separately per thread, i.e. each thread must create a VPE document itself by calling OpenDoc(), and one thread must not use the document handle of another thread for calls to the VPE API.

VPE will operate slower, if the thread-safe code is activated, due to acquiring and releasing thread-locks.

Once the thread-safe code has been activated, it is activated for **all** VPE documents that are currently open - and that will be opened later - by the calling application instance. Furthermore the thread-safe code can not be deactivated by the application instance.

The trial versions of VPE as well allow to activate the thread-safe code for testing purposes.

However, if you activated mutli-threading and call later the VpeLicense() method, the licensing will fail. You must license VPE by calling VpeLicense() **before** you activate the thread-safe code.

3.10 VpeGetLastError

Returns the error state of the last VPE function call.

long VpeGetLastError(
 VpeHandle *hDoc*
)

VpeHandle hDoc
 Document Handle

Returns:
 one of the VERR_xyz constants (see below)

Remarks:
 Not all functions do set / clear the error state. Only the functions which set the error state will also clear it, in case that no error occurred. All other functions keep the error state untouched.

 The "Remarks" section of each function described in this manual will clearly indicate, if the function will modify the LastError state.

Error Codes:

Constant Name	Value	Description
VERR_OK	0	no error
VERR_COMMON	1	common error
VERR_CANCELLED	2	the user cancelled an operation, for example the Open or Save file dialog
VERR_MEMORY	100	out of memory
VERR_FILE_OPEN	200	error opening a file; occurs when calling functions like OpenDoc() with SwapFileName set, ReadDoc(), WriteDoc(), ReadPrinterSetup(), Picture(), WriteRTFFile(), etc. Meaning: The specified file could not be opened, it is either not existing, the path is not existing, it is locked, incorrect logical structure (for example corrupted file)
VERR_FILE_DOCVERSION	201	Document file has the wrong (higher) version and can not be opened / read
VERR_FILE_CREATE	202	error creating file; occurs when calling functions like WriteDoc(), OpenDoc() with SwapFileName set, WritePrinterSetup(), SetupPrinter() (only during write), etc. Meaning: Illegal path / filename, file locked, disk full
VERR_FILE_ACCESS	203	Access denied (no permission); see also DocFileReadOnly
VERR_FILE_READ	204	error during file read operation
VERR_FILE_WRITE	205	error during file write operation
VERR_PIC_IMPORT	300	image could not be imported Meaning: File or resource not found, file not accessible, or image structure unreadable / corrupted, or not enough memory

VERR_PIC_NOLICENSE	301	No license for image access (e.g. TIFF / GIF image is LZW compressed and flag PIC_ALLOWLZW not used). This error code is obsolete. The LZW patend has expired. Since v4.00 VPE imports LZW compressed images without specifying PIC_ALLOWLZW.
VERR_PIC_DXFCOORD	302	For DXF formats, x2 and y2 may not be VFREE at the same time, either x2 or y2 must be <> VFREE
VERR_PIC_EXPORT	350	image could not be exported Meaning: File could not be created, or not enough memory
VERR_MOD_GRAPH_IMP	400	Error loading Graphics Import Library (Dav3_32.DLL)
VERR_MOD_GRAPH_PROC	401	Error loading Graphics Processing Library (Leon3_32.DLL)
VERR_MOD_BARCODE	402	Error loading Barcode Library (BarVisD.DLL)
VERR_MOD_CHART	403	Error loading Chart Library (VChart32.DLL)
VERR_MOD_ZLIB	404	Error loading ZLIB Library (VZLIB32.DLL)
VERR_MOD_VPDF	405	Error loading PDF Export Library (VPDF32.DLL)
VERR_MOD_VBAR2D	406	Error loading 2D Barcode Library (VBAR2D32.DLL)
VERR_MAIL_LOAD_MAPI	450	Could not load MAPI
VERR_MAIL_CREATE	451	Could not create temporary file
VERR_MAIL_USER_ABORT	452	
VERR_MAIL_FAILURE	453	
VERR_MAIL_LOGON_FAILURE	454	
VERR_MAIL_DISK_FULL	455	
VERR_MAIL_INSUFFICIENT_MEMORY	456	
VERR_MAIL_ACCESS_DENIED	457	
VERR_MAIL_RESERVED	458	
VERR_MAIL_TOO_MANY_SESSIONS	459	
VERR_MAIL_TOO_MANY_FILES	460	
VERR_MAIL_TOO_MANY_RECIPIENTS	461	
VERR_MAIL_ATTACHMENT_NOT_FOUND	462	
VERR_MAIL_ATTACHMENT_OPEN_FAILURE	463	
VERR_MAIL_ATTACHMENT_WRITE_FAILURE	464	
VERR_MAIL_UNKNOWN_RECIPIENT	465	
VERR_MAIL_BAD_RECIPTYPE	466	
VERR_MAIL_NO_MESSAGES	467	
VERR_MAIL_INVALID_MESSAGE	468	
VERR_MAIL_TEXT_TOO_LARGE	469	
VERR_MAIL_INVALID_SESSION	470	
VERR_MAIL_TYPE_NOT_SUPPORTED	471	
VERR_MAIL_AMBIGUOUS_RECIPIENT	472	
VERR_MAIL_MESSAGE_IN_USE	473	
VERR_MAIL_NETWORK_FAILURE	474	
VERR_MAIL_INVALID_EDITFIELDS	475	
VERR_MAIL_INVALID_RECIPS	476	
VERR_MAIL_NOT_SUPPORTED	477	

VERR_ZLIB_STREAM	500	Stream Inconsistent
VERR_ZLIB_DATA	501	Data Corrupt
VERR_ZLIB_BUFFER	502	Internal Buffer Error
VERR_ZLIB_VERSION	503	Wrong Version of VZLIB32.DLL
VERR_VBAR2D_FORMAT_OUT_OF_RANGE	550	
VERR_VBAR2D_UNDEFINED_ID	551	
VERR_VBAR2D_FORMAT_TOO_LONG	552	
VERR_VBAR2D_FORMAT_OUT_OF_MEMORY	553	
VERR_VBAR2D_FORMAT_DATA_INVALID	554	
VERR_VBAR2D_FORMAT_NOT_ALLOWED	555	
VERR_VBAR2D_DATA_WRONG_LENGTH	556	
VERR_VBAR2D_DATA_ZERO_LENGTH	557	
VERR_VBAR2D_DATA_TOO_SHORT	558	
VERR_VBAR2D_DATA_TOO_LONG	559	
VERR_VBAR2D_INVALID_DATA	560	
VERR_VBAR2D_SQUARE_EDGE_TOO_SMALL	561	
VERR_VBAR2D_SQUARE_TOO_LARGE	562	
VERR_VBAR2D_EDGE_OVER_FORCED	563	
VERR_VBAR2D_SQUARE_ASPECT_SMALL	564	
VERR_VBAR2D_SQUARE_ASPECT_LARGE	565	
VERR_VBAR2D_SQUARE_EVEN_ODD_MATCH	566	
VERR_VBAR2D_INVALID_EDGE	567	
VERR_VBAR2D_SQUARE_EDGE_TOO_LARGE	568	
VERR_VBAR2D_INVALID_ECC	569	
VERR_VBAR2D_INVALID_BORDER	570	
VERR_VBAR2D_SELF_TEST_FAILED	571	
VERR_RTF_BRACES	1000	RTF: unbalanced braces "{}"
VERR_RTF_OVERFLOW	1001	RTF: only 16-bit version; generated internal structure > 64 KB
VERR_RTF_FONTTBL	1002	RTF: error parsing font table
VERR_RTF_COLORTBL	1003	RTF: error parsing color table
VERR_TPL_OWNERSHIP	2000	Template: tried to dump the template to a foreign VPE document (where the template was not loaded into).
VERR_TPL_PAGE_ALREADY_DUMPED	2001	Template: the page contains interactive objects and already had been dumped. A page containing interactive objects may only be dumped once.
VERR_TPL_AUTHENTICATION	2002	Template: the template has an authentication key and it has not been validated successfully

3.11 VpePreviewDoc

[Windows platform only]

Opens the preview window. The preview shows the page specified by VpeGotoVisualPage() (by default, this is the first page).

void VpePreviewDoc(
VpeHandle *hDoc*,
RECT **rc*,
int *show_hide*
)

VpeHandle hDoc
Document Handle

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

int show_hide
can be one of the following predefined constants:

Value	**Description**
VPE_SHOW_NORMAL	1: show window normal
VPE_SHOW_MAXIMIZED	2: show window maximized
VPE_SHOW_HIDE	3: hide the preview window

3.12 VpePreviewDocSP

[Windows platform only]

The same as VpePreviewDoc(), opens the preview window. The preview shows the page specified by VpeGotoVisualPage() (by default, this is the first 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.

void VpePreviewDocSP(
  VpeHandle *hDoc*,
  int *x*,
  int *y*,
  int *x2*,
  int *y2*,
  int *show_hide*
)

VpeHandle hDoc
  Document Handle

int x, y, x2, y2
  position of the preview window in pixels, for x = -1 all coordinates are ignored.

int show_hide
  can be one of the following predefined constants:

Value	**Description**
VPE_SHOW_NORMAL	1: show window normal
VPE_SHOW_MAXIMIZED	2: show window maximized
VPE_SHOW_HIDE	3: hide the preview window

3.13 VpeCenterPreview

[Windows platform only]

The preview-window is centered in the middle of the parent window or desktop.

void VpeCenterPreview(
VpeHandle *hDoc*,
int *width*,
int *height*,
HWND *parent_window*
)

VpeHandle hDoc
Document Handle

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

Value	Description
Width	1: the preview's width is kept
Height	1: the preview's height is kept
Width	2: the preview's width is set to the width of the parent window (or desktop)
Height	2: the preview's height is set to the height of the parent window (or desktop)

HWND parent_window
a window, which will be used as reference. If it is NULL (0), the preview is centered on the desktop (screen)

3.14 VpeSetPreviewCtrl

[Windows platform only]

Defines the behavior of the preview for page forward / backward action. It specifies the vertical positioning of the page in the preview after the user moved a page forward / backward.

void VpeSetPreviewCtrl(
VpeHandle *hDoc*,
int *setting*
)

VpeHandle hDoc
Document Handle

int setting
possible values are:

Constant Name	Value	Comment
PREVIEW_STAY	0	vertical position unaffected
PREVIEW_JUMPTOP	1	vertical position on top of page (default)

Default:
PREVIEW_JUMPTOP

Example:

```
VpeSetPreviewCtrl(hDoc, PREVIEW_STAY)
```

3.15 VpeClosePreview

[Windows platform only]

Closes the preview. The document is **NOT** closed (= removed from memory), regardless of the setting of EnableAutoDelete. So you can re-open the preview with the method VpePreviewDoc() after a call to this method.

void VpeClosePreview(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

3.16 VpeIsPreviewVisible

[Windows platform only]

Returns the visibility state of the Preview.

int VpeIsPreviewVisible(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
True	the Preview is visible
False	the Preview is invisible (hidden)

3.17 VpeGetVisualPage

[Windows platform only]

Retrieve the number of the currently visible page in the preview. This page is independently from the currently active page, where you can insert objects on (see VpeGetCurrentPage() and VpeGotoPage()).

int VpeGetVisualPage(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  The number of the currently visible page.

Default:
  1 = the first page, after a call to VpeOpenDoc()

Example:

```
VpeGotoVisualPage(hDoc, 5)          // moves the preview to page 5
n = VpeGetVisualPage(hDoc)          // returns the value 5
```

3.18 VpeGotoVisualPage

[Windows platform only]

Moves the preview to the specified page. VPE has internally two working pages: one your application is working on and one - the visual page - which is currently viewed by the user (if the preview is open). With this property you can set the position of the visual page.

This page is independently from the currently active page, where you can insert objects on (see VpeGetCurrentPage() and VpeGotoPage())

int VpeGotoVisualPage(
VpeHandle *hDoc*,
int *page*
)

VpeHandle hDoc
Document Handle

int page
page number

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set.

If you want to show the preview and let the user work with it (e.g. scroll and print) while your application is still generating the document, see "Multipage Documents" in the Programmer's Manual for details.

Example:

```
// retrieve the page that is currently shown in the preview:
n = GetVisualPage(hDoc)

// move the preview to the next page:
VpeGotoVisualPage(hDoc, n + 1)
```

3.19 VpeDispatchAllMessages

[Windows platform only]

Needs to be called regularly to allow the user to browse in the preview through an open document while your application is still generating the report (i.e. adding objects and pages).

See "Multipage Documents" in the Programmer's Manual for details.

int VpeDispatchAllMessages(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

The method returns True, if the preview - or the parent window of the preview - are closed by the user. In such a case you should perform the necessary cleanup (for example close tables and databases) and exit the function that creates the report immediately. **You may not call any VPE function of the related document, if this function returns True.**

The method returns False, if the preview or the parent window of the preview are NOT closed by the user.

Remarks:

Never call VpeDispatchAllMessages() while processing a VPE Event!

3.20 VpeMapMessage

[Windows platform only]

This method maps a globally registered VPE Message into one of the VPE_xyz message constants.

Working with the VPE Messages is not necessary, but it makes the use of VPE and the interaction between your application, VPE and the end user much more well-appointed.

The messages generated by VPE (e.g. VPE_DESTROYWINDOW, etc.) are globally registered in the windows system to avoid conflicts with other controls and applications. As it is unknown, what value equals each message, you need to map the messages with this method in your window-procedure.

If your tool is not able to process dynamically window-messages, or if you want to gain the old behavior of VPE prior to v3.0 (having fixed values for each message), use the new flag **VPE_FIXED_MESSAGES** in VpeOpenDoc(). If you use VPE_FIXED_MESSAGES, VPE will directly send the VPE_xyz message constants to the window procedure of your parent window, instead of sending the globally registered messages. You may not call VpeMapMessage(), if you are using VPE_FIXED_MESSAGES.

UINT VpeMapMessage(
  VpeHandle *hDoc*,
  UINT *message*
)

VpeHandle hDoc
  Document Handle

UINT message
  the message received in the window procedure of your parent window

Returns:
  One of the VPE_xyz message constants, see Messages Generated by VPE - DLL.

Remarks:
  To make things easier if you have multiple open documents, VpeMapMessage() assumes for hDoc = NULL that no fixed messages are used and directly maps the message into the dynamic message map. You can only make use of this feature, if ALL VPE-Documents were created with dynamic message mapping.

C/C++ Example with fixed messages VpeOpenDoc(..., VPE_FIXED_MESSAGES):

```
// =====================================================================
//                              WndProc
// =====================================================================
long FAR PASCAL _export WndProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
  switch (message)
  {
  case WM_LBUTTONDOWN:
       if (!hDoc)
            MiniDemo(hwnd);    // creates doc, sets global hDoc-variable
       return 0;

  case VPE_DESTROYWINDOW:
       hDoc = 0;
       PostMessage(hwnd, WM_CLOSE, 0, 0);     // only, if preview is
                   embedded!
       return 0;

  case WM_CLOSE:
       if (!VpeCloseDoc(hDoc))
       {
            Msg("The application cannot terminate until all jobs have
                finished printing.");
            return 0;
       }
       DestroyWindow(hwnd);
       return 1;

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

  return DefWindowProc(hwnd, message, wParam, lParam);
}
```

C/C++ Example with dynamic message mapping:

```
// =====================================================================
//                              WndProc
// =====================================================================
long FAR PASCAL _export WndProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
  switch (message)
  {
  case WM_LBUTTONDOWN:
       if (!hDoc)
            MiniDemo(hwnd);    // creates doc, sets global hDoc-variable
       return 0;

  case WM_CLOSE:
       if (!VpeCloseDoc(hDoc))
       {
            Msg("The application cannot terminate until all jobs have
                 finished printing.");
            return 0;
       }
       DestroyWindow(hwnd);
       return 1;

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

  if (hDoc)
  {
       switch (VpeMapMessage(hDoc, message))
       {
       case VPE_DESTROYWINDOW:
            hDoc = 0;
            PostMessage(hwnd, WM_CLOSE, 0, 0);   // only, if preview is
                       embedded!
            return 0;
       }
  }

  return DefWindowProc(hwnd, message, wParam, lParam);
}
```

C/C++ Example with dynamic message mapping and multiple documents:

```
// ==================================================================
//                              WndProc
// ==================================================================
long FAR PASCAL _export WndProc(HWND hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
  switch (message)
  {

  .
  .
  .

  case WM_CLOSE:
       if (!VpeCloseDoc(hDocRTF) ||
            !VpeCloseDoc(hDocUDO) ||
            !VpeCloseDoc(hDocScale2Gray))
       {
            Msg("The application cannot terminate until all jobs have
                finished printing.");
            return 0;
       }
       DestroyWindow(hwnd);
       return 1;

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

  switch (VpeMapMessage(NULL, message))
  {
  case VPE_DESTROYWINDOW:
       if (hDocRTF == lParam)      // identify document, the message is
                                      for...
            hDocRTF = 0;
       else if (hDocUDO == lParam)
            hDocUDO = 0;
       else if (hDocScale2Gray == lParam)
            hDocScale2Gray = 0;
       return 0;

  case VPE_PRINT: // the following MessageBoxes will appear for all Docs
       switch (wParam)
       {
       case PRINT_MSG_SETUPABORT:  // User aborted Setup-Dialog
            MessageBox(hwnd, "Caught event from VPE: Setup aborted.",
                       szAppName, MB_OK);
            break;

       case PRINT_MSG_SETUPSTART: // Setup-Dialog started
            MessageBox(hwnd, " Caught event from VPE: Setup started.",
                       szAppName, MB_OK);
            break;
       }
       return 0;
  }
  return DefWindowProc(hwnd, message, wParam, lParam);
}
```

3.21 VpeRefreshDoc

[Windows platform only]

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

void VpeRefreshDoc(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

3.22 VpeSetDocExportType

Sets the current Document Export Type. The value of this property controls, what type of document is created by the method VpeWriteDoc().

void VpeSetDocExportType(
VpeHandle *hDoc*,
int *doc_type*
)

VpeHandle hDoc
Document Handle

int doc_type
possible values are:

Constant Name	Value	Comment
VPE_DOC_TYPE_AUTO	0	WriteDoc() will create VPE, PDF or HTML files, depending on the file suffix of the supplied file name
VPE_DOC_TYPE_VPE	1	WriteDoc() will create VPE files, regardless of the file suffix
VPE_DOC_TYPE_PDF	2	WriteDoc() will create PDF files, regardless of the file suffix
VPE_DOC_TYPE_HTML	3	WriteDoc() will create HTML files, regardless of the file suffix

Default:
VPE_DOC_TYPE_AUTO

3.23 VpeGetDocExportType

Returns the current setting of the Document Export Type.

int VpeGetDocExportType(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

3.24 VpeSetCompression

[Not supported by the Community Edition]

Sets the current mode for compression. This affects writing native VPE document files as well as exporting PDF files.

void VpeSetCompression(
VpeHandle *hDoc*,
int *compression*
)

VpeHandle hDoc
Document Handle

int compression
possible values are:

Constant Name	Value	Comment
DOC_COMPRESS_NONE	0	no compression
DOC_COMPRESS_FLATE	1	flate (ZLIB) compression

Default:
DOC_COMPRESS_FLATE

Remarks:
When creating a VPE document file using VpeOpenDocFile(), compression is always activated.

The trial version of VPE will always use compression for exported PDF files.

For the Community Edition, compression is not available.

3.25 VpeGetCompression

Returns the current setting of the compression mode.

int VpeGetCompression(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    possible values are:

Constant Name	**Value**	**Comment**
DOC_COMPRESS_NONE	0	no compression
DOC_COMPRESS_FLATE	1	flate (ZLIB) compression

3.26 VpeOpenFileDialog

[Windows platform only; not supported by the Community Edition]

Displays a file-open dialog, so the user can select a VPE document file for reading.

void VpeOpenFileDialog(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Remarks:
    You can preset a file name for the file-open dialog with the method VpeSetOpenFileName().

3.27 VpeSaveFileDialog

[Windows platform only; not supported by the Community Edition]

Displays a file-save dialog, so the user can select a file name under which the current VPE document file is written.

void VpeSaveFileDialog(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Remarks:
    You can preset a file name for the file-save dialog with the method VpeSetSaveFileName().

3.28 VpeSetOpenFileName

[Windows platform only]

Sets the default path and file name, which will be shown in the Open File Dialog.

void VpeSetOpenFileName(
    VpeHandle *hDoc*,
    LPCTSTR *file_name*
)

VpeHandle hDoc
    Document Handle

LPCTSTR file_name
    the file name

Default:
    empty string

3.29 VpeGetOpenFileName

[Windows platform only]

Returns the file name, which was set by VpeSetOpenFileName() or the Open File Dialog.

After the Open File Dialog has been confirmed with OK, this property holds the path and file name selected by the user.

void VpeGetOpenFileName(
  VpeHandle *hDoc*,
  LPTSTR *file_name,*
  UINT **size,*
)

VpeHandle hDoc
  Document Handle

LPTSTR file_name
  Pointer to a buffer that receives the file name string. This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
  Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the *file_name* parameter. When the function returns, this variable contains the number of bytes copied to *file_name* - including the size of the terminating null character.
  If *file_name* is NULL, and size is non-NULL, the function returns TRUE and stores the size of the required buffer, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the *file_name* 's data.

Returns:

Value	Description
True	Value retrieved successfully
False	Value could not be retrieved (for example out of memory)

3.30 VpeSetSaveFileName

[Windows platform only]

Sets the default path and file name, which will be shown in the Save File Dialog.

void VpeSetSaveFileName(
  VpeHandle *hDoc*,
  LPCTSTR *file_name*
)

VpeHandle hDoc
  Document Handle

LPCTSTR file_name
  the file name

Default:
  empty string

3.31 VpeGetSaveFileName

[Windows platform only]

Returns the file name, which was set by VpeSetSaveFileName() or the Save File Dialog.

After the Save File Dialog has been confirmed with OK, this property holds the path and file name selected by the user.

void VpeGetSaveFileName(
VpeHandle *hDoc*,
LPTSTR *file_name,*
UINT **size,*
)

VpeHandle hDoc
Document Handle

LPTSTR file_name
Pointer to a buffer that receives the file name string. This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the *file_name* parameter. When the function returns, this variable contains the number of bytes copied to *file_name* - including the size of the terminating null character.
If *file_name* is NULL, and size is non-NULL, the function returns TRUE and stores the size of the required buffer, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the *file_name* 's data.

Returns:

Value	Description
True	Value retrieved successfully
False	Value could not be retrieved (for example out of memory)

3.32 VpeWriteDoc

Writes the currently open document to an external file.

Depending on the file suffix, the file is written to the following format:

- .vpe – Native VPE document file format
- .pdf – PDF file format
- .htm or .html – HTML file format (requires Professional Edition or higher)

For details about creating and using VPE document files, please see the "Programmer's Manual", chapter "Programming Techniques", subchapter "VPE Document Files".

For details about creating PDF files, please see the "Programmer's Manual", chapter "The PDF Export Module".

For details about creating PDF files, please see the "Programmer's Manual", chapter "The HTML Export Module".

int VpeWriteDoc(
    VpeHandle *hDoc*,
    LPCSTR *file_name*
)

VpeHandle hDoc
    Document Handle

LPCSTR file_name
    the path and filename

Returns:

Value	Description
True	Success
False	Failure

Remarks:

In case of an error, LastError is set.

Depending on the setting of the property DocExportType, you can instruct VPE to write the file to a desired file format (VPE, PDF or HTML), regardless of the supplied file name's suffix.

The property Compression controls, whether a written document (VPE or PDF file format) is compressed.

Protection
VPE Document files can be read, edited and saved using our visual designer *dycodoc*.
If you want to protect your files from being edited, use the property EditProtection.

3.33 VpeWriteDocPageRange

[Professional Edition and above]

Identical to VpeWriteDoc(), but writes only the given range of pages to a document file.

int VpeWriteDocPageRange(
  VpeHandle *hDoc*,
  LPCSTR *file_name*,
  int *from_page*,
  int *to_page*
)

VpeHandle hDoc
  Document Handle

LPCSTR file_name
  the path and filename

int from_page
  starting page of the page range

int to_page
  ending page of the page range

Returns:

Value	Description
True	Success
False	Failure

3.34 VpeWriteDocStream

[Professional Edition and above]

Identical to VpeWriteDoc(), but writes the document to a stream. Currently the only type of stream offered by VPE is a memory stream.

int VpeWriteDocStream(
VpeHandle *hDoc*,
VpeHandle *hStream*
)

VpeHandle hDoc
Document Handle

VpeHandle hStream
The handle of the stream where the document is written to. The stream must have been created before by calling VpeCreateMemoryStream().

Returns:

Value	Description
True	Success
False	Failure

3.35 VpeWriteDocStreamPageRange

[Professional Edition and above]

Identical to VpeWriteDocStream(), but writes only the given range of pages to a stream.

int VpeWriteDocStreamPageRange(
    VpeHandle *hDoc*,
    VpeHandle *hStream*,
    int *from_page*,
    int *to_page*
)

VpeHandle hDoc
    Document Handle

VpeHandle hStream
    The handle of the stream where the document is written to. The stream must have been created before by calling VpeCreateMemoryStream().

int from_page
    starting page of the page range

int to_page
    ending page of the page range

Returns:

Value	**Description**
True	Success
False	Failure

3.36 VpeReadDoc

Reads a VPE 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 must have been created before with VpeOpenDocFile() or VpeWriteDoc().

int VpeReadDoc(
 VpeHandle *hDoc*,
 LPCSTR *file_name*
)

VpeHandle hDoc
 Document Handle

LPCSTR file_name
 the path and filename

Returns:

Value	Description
True	Success
False	Failure

Remarks:

In case of an error, LastError is set.

Tip: If you want to preview or print an already created VPE-Document file, DON'T USE VpeReadDoc(). Use OpenDocFile() instead. This guarantees minimum memory usage with the same performance. (Believe us, our high-performance swapping technology is really FAST.)

This method can not import PDF or any other document files, except VPE document files.

3.37 VpeReadDocPageRange

[Professional Edition and above]

Identical to VpeReadDoc(), but reads only the given range of pages from a document file.

int VpeReadDocPageRange(
    VpeHandle *hDoc*,
    LPCSTR *file_name*,
    int *from_page*,
    int *to_page*
)

VpeHandle hDoc
    Document Handle

LPCSTR file_name
    the path and filename

int from_page
    starting page of the page range

int to_page
    ending page of the page range

Returns:

Value	Description
True	Success
False	Failure

Remarks:
    In case of an error, LastError is set.

3.38 VpeReadDocStream

[Professional Edition and above]

Identical to VpeReadDoc(), but reads the document from a stream. Currently the only type of stream offered by VPE is a memory stream.

int VpeReadDocStream(
VpeHandle *hDoc*,
VpeHandle *hStream*
)

VpeHandle hDoc
Document Handle

VpeHandle hStream
The handle of the stream where the document shall be read from. The stream must have been created before by calling VpeCreateMemoryStream() and of course it must hold a valid VPE document.

Returns:

Value	Description
True	Success
False	Failure

3.39 VpeReadDocStreamPageRange

[Professional Edition and above]

Identical to VpeReadDocStream(), but reads only the given range of pages from a stream.

int VpeReadDocStreamPageRange(
  VpeHandle *hDoc*,
  VpeHandle *hStream*,
  int *from_page*,
  int *to_page*
)

VpeHandle hDoc
  Document Handle

VpeHandle hStream
  The handle of the stream where the document is written to. The stream must have been created before by calling VpeCreateMemoryStream().

int from_page
  starting page of the page range

int to_page
  ending page of the page range

Returns:

Value	Description
True	Success
False	Failure

3.40 VpeSetDocFileReadOnly

If you read an existing document file with VpeReadDoc() or VpeReadDocPageRange() into memory, you should set the read-only mode to True. In this case the document file is not opened in exclusive-mode and therefore other applications and users can browse and print it simultaneously.

See also the flag VPE_DOCFILE_READONLY under the description of VpeOpenDoc.

void VpeSetDocFileReadOnly(
  VpeHandle *hDoc*,
  int *yes_no*
)

VpeHandle hDoc
  Document Handle

int yes_no

Value	Description
True	read-only: file can be shared
False	exclusive access

Default:
  False

3.41 VpeEnableAutoDelete

[Windows platform only; not supported by the Community Edition]

Specifies, if the document shall be closed (removed from memory) when the user closes the preview.

void VpeEnableAutoDelete(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	yes, close the document also
False	No

Default:
True (the document is deleted, if the user closes the preview)

3.42 VpeEnablePrintSetupDialog

[Windows platform only; not supported by the Community Edition]

If set to False, no printer setup dialog will appear when the print button (or the corresponding key) in the preview is pushed.

void VpeEnablePrintSetupDialog(
VpeHandle *hDoc*,
int *enabled*
)

VpeHandle hDoc
Document Handle

int enabled

Value	**Description**
True	Enabled
False	Disabled

Default:
enabled (Setup Dialog will appear)

3.43 VpeEnableMailButton

[Windows platform only; not supported by the Community Edition]

Setting this property to True / False will enable / disable the e-mail button in the toolbar.

void VpeEnableMailButton(
VpeHandle *hDoc*,
int *enabled*
)

VpeHandle hDoc
Document Handle

int enabled

Value	Description
True	Enabled
False	Disabled

Default:
enabled

3.44 VpeEnableCloseButton

[Windows platform only; not supported by the Community Edition]

Specifies, whether the Close Button in the preview is enabled (= activated, so the user can push it).

If the preview is not embedded (ExternalWindow = True), the close controls of the window (Close Button including the system-menu) are also disabled.

You can use this property to temporarily lock the preview from being closed by the user.

void VpeEnableCloseButton(
    VpeHandle *hDoc*,
    int *enabled*
)

VpeHandle hDoc
    Document Handle

int enabled

Value	Description
True	enabled
False	disabled

Default:
    enabled (= Close Button is enabled)

Remarks:
    This property does not control, whether the button is visible in the toolbar or not. (see VpeOpenDoc-Flag: VPE_NO_USER_CLOSE)

3.45 VpeEnableHelpRouting

[Windows platform only; not supported by the Community Edition]

If enabled and the Help-Button is visible, pushing the Help-Button or pressing the Help key will cause the event VPE_HELP to be sent to the parent window, no matter if the VPE-Preview window is embedded or not. With this option you are able to present context-sensitive help to your user. If this property is False, VPE's build-in small help dialog will be shown.

void VpeEnableHelpRouting(
VpeHandle *hDoc*,
int *enabled*
)

VpeHandle hDoc
Document Handle

int enabled

Value	Description
True	enabled (help event will be sent)
False	disabled (VPE's build-in small help dialog will be shown)

Default:
False (VPE's build-in small help dialog will be shown)

3.46 VpeSetGridMode

[Windows platform only; not supported by the Community Edition]

Specifies, whether the Layout Grid that can be optionally shown in the Preview is drawn in foreground or background. This property does not control, if the grid is drawn. It controls how the grid is drawn.

void VpeSetGridMode(
VpeHandle *hDoc*,
int *in_foreground*
)

VpeHandle hDoc
Document Handle

int in_foreground

Value	Description
True	in foreground
False	in background

Default:
True

3.47 VpeSetGridVisible

[Windows platform only; not supported by the Community Edition]

Specifies, if the Layout Grid is drawn.

void VpeSetGridVisible(
  VpeHandle *hDoc*,
  int *yes_no*
)

VpeHandle hDoc
  Document Handle

int yes_no

Value	Description
True	is drawn
False	is not drawn

Default:
  False

3.48 VpeSetPreviewWithScrollers

[Windows platform only; not supported by the Community Edition]

If set to False, the scrollbars in the Preview are forced to be hidden. Use this property with care.

void VpeSetPreviewWithScrollers(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	scrollbars are visible
False	scrollbars are invisible

Default:
True

3.49 VpeSetPaperView

[Windows platform only; not supported by the Community Edition]

If set to True, the document in the preview is drawn like a sheet of paper, reflecting the page-dimensions.

void VpeSetPaperView(
  VpeHandle *hDoc*,
  int *on_off*
)

VpeHandle hDoc
  Document Handle

int on_off

Value	Description
True	document is drawn like a sheet of paper
False	the window background is drawn in white, without borders

Default:
  True

3.50 VpeSetPageScrollerTracking

[Windows platform only; not supported by the Community Edition]

When activated (set to true), moving the page scroller in the statusbar will immediately update the preview. Otherwise the preview will be updated if the scrolling has finished.

void VpeSetPageScrollerTracking(
VpeHandle *hDoc*,
int *on_off*
)

VpeHandle hDoc
Document Handle

int on_off

Value	Description
True	enabled
False	disabled

Default:
True

3.51 VpeWriteStatusbar

[Windows platform only; not supported by the Community Edition]

Writes the given text into the Statusbar. If text is NULL (0), the default "Ready" in the current GUI-Language will be displayed.

void VpeWriteStatusbar(
VpeHandle *hDoc*,
LPCSTR *text*
)

VpeHandle hDoc
Document Handle

LPCSTR text
The text that shall be written into the Statusbar.

Remarks:
The Progress Bar overlays the text segment in the Statusbar, therefore text will not be visible until the Progress Bar is closed.

3.52 VpeOpenProgressBar

[Windows platform only; not supported by the Community Edition]

Creates a Progress Bar in the Statusbar with the initial value of zero (0%).

void VpeOpenProgressBar(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Remarks:
  The Progress Bar hides the text segment in the Statusbar, therefore text displayed there with VpeWriteStatusbar will not be visible until the Progress Bar is closed.

Example:

```
VpeOpenProgressBar(hDoc)
for n = 0 to 100
  VpeSetProgressBar(hDoc, n)
next n
VpeCloseProgressBar(hDoc)
```

3.53 VpeSetProgressBar

[Windows platform only; not supported by the Community Edition]

Sets the Progress Bar to the given percentage value.

void VpeSetProgressBar(
  VpeHandle *hDoc*,
  int *percent*
)

VpeHandle hDoc
  Document Handle

int percent
  a number between 0 and 100, specifying the percentage value, the progress bar shall display

Example:

```
VpeOpenProgressBar(hDoc)
for n = 0 to 100
  VpeSetProgressBar(hDoc, n)
next n
VpeCloseProgressBar(hDoc)
```

3.54 VpeCloseProgressBar

[Windows platform only; not supported by the Community Edition]

Closes the Progress Bar.

void VpeCloseProgressBar(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

3.55 VpeSetBusyProgressBar

[Windows platform only, Enhanced Edition and above]

When this property is enabled and you are exporting a document, for example to PDF or XML, there will be shown a Progress Bar in the text segment of the Statusbar.

If you are also showing the private Progress Bar (see VpeOpenProgressBar()), **both** Progress Bars will be shown simultaneously. They are drawn in a way that overlapping parts are shown in a mixture of the colors each Progress Bar is using.

void VpeSetBusyProgressBar(
    VpeHandle *hDoc*,
    int *visible*
)

VpeHandle hDoc
    Document Handle

int visible

Value	Description
True	Enabled
False	Disabled

Default:
    True

3.56 VpeSetRulersMeasure

[Windows platform only]

Sets the measurement for the rulers. This does not affect the coordinate system of the API. To change the coordinate system of the API, use VpeSetUnitTransformation().

void VpeSetRulersMeasure(
VpeHandle *hDoc*,
int *rulers_measure*
)

VpeHandle hDoc
Document Handle

int rulers_measure
possible values are:

Value	Description
CM	0: rulers are shown in metric units
INCH	1: rulers are shown in inch units

Default:
CM

3.57 VpeSetScale

[Windows platform only; not supported by the Community Edition]

Sets the scale for the preview (**not for printing**).

void VpeSetScale(
    VpeHandle *hDoc*,
    double *scale*
)

VpeHandle hDoc
    Document Handle

double scale
    for example: 1.0 = 1:1   0.25 = 1:4   4.0 = 4:1

Default:
    Variable, the initial scale is computed so that the current page fits with its width into the preview.

3.58 VpeGetScale

[Windows platform only; not supported by the Community Edition]

Returns the current scale of the preview.

double VpeGetScale(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current scale

3.59 VpeSetScalePercent

[Windows platform only; not supported by the Community Edition]

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

void VpeSetScalePercent(
    VpeHandle *hDoc*,
    int *scale*
)

VpeHandle hDoc
    Document Handle

int scale
    for example: 100 = 100% (1:1)   25 = 25% (1:4)   400 = 400% (4:1)

Default:
    Variable, the initial scale is computed so that the current page fits with its width into the preview.

3.60 VpeGetScalePercent

[Windows platform only; not supported by the Community Edition]

Returns the current scale of the preview in percent.

int VpeGetScalePercent(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current scale in percent

3.61 VpeSetMinScale

[Windows platform only; not supported by the Community Edition]

Specifies the minimum possible preview scale factor that can be set by the user. MinScale may not be less than 0.25 (= 25 %).

void VpeSetMinScale(
    VpeHandle *hDoc*,
    double *min_scale*
)

VpeHandle hDoc
    Document Handle

double min_scale
    for example: 1.0 = 1:1   0.25 = 1:4

Default:
    0.25

3.62 VpeSetMinScalePercent

[Windows platform only; not supported by the Community Edition]

Specifies the minimum preview scale factor that can be set by the user. MinScale may not be less than 25 (25 %)

void VpeSetMinScalePercent(
VpeHandle *hDoc*,
int *min_scale_percent*
)

VpeHandle hDoc
Document Handle

int min_scale_percent
for example: 100 = 1:1   25 = 1:4

Default:
25

3.63 VpeSetMaxScale

[Windows platform only; not supported by the Community Edition]

Specifies the maximum preview scale factor that can be set by the user. MaxScale may not be greater than 32.0 (3200 %).

void VpeSetMaxScale(
  VpeHandle *hDoc*,
  double *max_scale*
)

VpeHandle hDoc
  Document Handle

double max_scale

for example: 1.0 = 100%	0.25 = 25%	6.0 = 600%

Default:
  32.0 (3200 %)

3.64 VpeSetMaxScalePercent

[Windows platform only; not supported by the Community Edition]

Specifies the maximum preview scale factor that can be set by the user. MaxScale may not be greater than 3200 (3200 %).

void VpeSetMaxScalePercent(
    VpeHandle *hDoc*,
    int *max_scale_percent*
)

VpeHandle hDoc
    Document Handle

int max_scale_percent
    for example: 100 = 100%   25 = 25%   600 = 600%

Default:
    3200 (3200 %)

3.65 VpeSetScaleMode

[Windows platform only; not supported by the Community Edition]

Sets the scale mode of the preview.

void VpeSetScaleMode(
    VpeHandle *hDoc*,
    int *mode*
)

VpeHandle hDoc
    Document Handle

int mode
    possible values are:

Constant Name	Value	Comment
VSCALE_MODE_FULL_PAGE	0	show full page
VSCALE_MODE_PAGE_WIDTH	1	fit page width
VSCALE_MODE_ZOOM_TOOL	2	zoom tool activated
VSCALE_MODE_FREE	3	any scale allowed

Default:
    VSCALE_MODE_PAGE_WIDTH

3.66 VpeGetScaleMode

[Windows platform only; not supported by the Community Edition]

Returns the current scale mode of the preview.

int VpeGetScaleMode(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    possible return values are:

Constant Name	Value	Comment
VSCALE_MODE_FULL_PAGE	0	show full page
VSCALE_MODE_PAGE_WIDTH	1	fit page width
VSCALE_MODE_ZOOM_TOOL	2	zoom tool activated
VSCALE_MODE_FREE	3	any scale allowed

3.67 VpeZoomPreview

[Windows platform only; not supported by the Community Edition]

Zooms a specified rectangle of the current page into the preview.

void VpeZoomPreview(
VpeHandle *hDoc*,
int *left*,
int *top*,
int *right*,
int *bottom*
)

VpeHandle hDoc
Document Handle

int left, top, right, bottom
the rectangle of the current page in metric or inch coordinates

Remarks:
This method requires a visible preview. If the preview is not visible, the method does nothing.

3.68 VpeZoomIn

[Windows platform only; not supported by the Community Edition]

Enlarges the view. Switches to the next larger scale of the internal hard-coded scaling table.

void VpeZoomIn(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Remarks:
This method requires a visible preview. If the preview is not visible, the method does nothing.

3.69 VpeZoomOut

[Windows platform only; not supported by the Community Edition]

Reduces the view. Switches to the next smaller scale of the internal hard-coded scaling table.

void VpeZoomOut(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Remarks:
This method requires a visible preview. If the preview is not visible, the method does nothing.

3.70 VpeSetPreviewPosition

[Windows platform only; not supported by the Community Edition]

Scrolls the preview into the given position.

void VpeSetPreviewPosition(
    VpeHandle *hDoc*,
    int *left*,
    int *top*
)

VpeHandle hDoc
    Document Handle

int left, top
    the position within the current page in metric or inch coordinates

Remarks:
    This method requires a visible preview. If the preview is not visible, the method does nothing.

3.71 VpeDefineKey

[Windows platform only; not supported by the Community Edition]

This method allows you to define your own key for **each** available keyboard function.

void VpeDefineKey(
VpeHandle *hDoc*,
int *function*,
int *key_code*,
int *add_key_code1*,
int *add_key_code2*
)

VpeHandle hDoc
Document Handle

int function
All available keyboard functions are enumerated in the VKEY_xyz constants (see below).

int key_code, add_key_code1, add_key_code2
Virtual keycode constants (defined in the Windows SDK) for the keys that need to be pressed at once for the specified function.

Remarks:

A keyboard function can be set to "undefined" = no key is associated with this keyboard function by setting all key codes to zero.

Example:

```
VpeDefineKey(hDoc, VKEY_CLOSE, VK_ESCAPE, 0, 0)
```

Defines the key <ESC> to close the preview when pressed.

```
VpeDefineKey(hDoc, VKEY_CLOSE, VK_ESCAPE, VK_SHIFT, 0)
```

Defines the key pair <SHIFT> + <ESC> to close the preview when pressed.

```
VpeDefineKey(hDoc, VKEY_PRINT, 0, 0, 0)
```

There is no key associated with the keyboard function VKEY_PRINT.

Possible values for the parameter "function" are:

Constant	Value	Default Key(s)
VKEY_SCROLL_LEFT	0	Arrow Left
VKEY_SCROLL_PAGE_LEFT	1	Ctrl-Arrow Left
VKEY_SCROLL_RIGHT	2	Arrow Right
VKEY_SCROLL_PAGE_RIGHT	3	Ctrl-Arrow Right
VKEY_SCROLL_UP	4	Arrow Up
VKEY_SCROLL_PAGE_UP	5	Ctrl-Arrow Up
VKEY_SCROLL_DOWN	6	Arrow Down
VKEY_SCROLL_PAGE_DOWN	7	Ctrl-Arrow Down

VKEY_SCROLL_TOP	8	Pos1
VKEY_SCROLL_BOTTOM	9	End
VKEY_PRINT	10	F2
VKEY_MAIL	11	F3
VKEY_FULL_PAGE	12	Ctrl-Ins
VKEY_PAGE_WIDTH	13	Ctrl-Del
VKEY_ZOOM_IN	14	Ins
VKEY_ZOOM_OUT	15	Del
VKEY_GRID	16	"G"
VKEY_PAGE_FIRST	17	Ctrl-Page Up
VKEY_PAGE_LEFT	18	Page Up
VKEY_PAGE_RIGHT	19	Page Down
VKEY_PAGE_LAST	20	Ctrl-Page Down
VKEY_HELP	21	F1
VKEY_INFO	22	"I"
VKEY_CLOSE	23	<not defined>
VKEY_GOTO_PAGE	24	ENTER
VKEY_OPEN	25	Ctrl-O
VKEY_SAVE	26	Ctrl-S
VKEY_ESCAPE	27	Esc

See Also:

VpeSendKey

3.72 VpeSendKey

[Windows platform only; not supported by the Community Edition]

Simulates a keystroke of the user. This allows your application for example to scroll the preview.

void VpeSendKey(
 VpeHandle *hDoc*,
 int *vkey*
)

VpeHandle hDoc
 Document Handle

int vkey
 All available keyboard functions are enumerated in the VKEY_xyz constants:

Constant	Value
VKEY_SCROLL_LEFT	0
VKEY_SCROLL_PAGE_LEFT	1
VKEY_SCROLL_RIGHT	2
VKEY_SCROLL_PAGE_RIGHT	3
VKEY_SCROLL_UP	4
VKEY_SCROLL_PAGE_UP	5
VKEY_SCROLL_DOWN	6
VKEY_SCROLL_PAGE_DOWN	7
VKEY_SCROLL_TOP	8
VKEY_SCROLL_BOTTOM	9
VKEY_PRINT	10
VKEY_MAIL	11
VKEY_FULL_PAGE	12
VKEY_PAGE_WIDTH	13
VKEY_ZOOM_IN	14
VKEY_ZOOM_OUT	15
VKEY_GRID	16
VKEY_PAGE_FIRST	17
VKEY_PAGE_LEFT	18
VKEY_PAGE_RIGHT	19
VKEY_PAGE_LAST	20
VKEY_HELP	21
VKEY_INFO	22
VKEY_CLOSE	23
VKEY_GOTO_PAGE	24
VKEY_OPEN	25

VKEY_SAVE	26
VKEY_ESCAPE	27

Example:

```
VpeSendKey(hDoc, VKEY_SCROLL_DOWN)
```

Scrolls the preview down by one cm.

See Also:

VpeDefineKey

3.73 VpeSetGUITheme

[Windows platform only; not supported by the Community Edition]

Specifies the look and feel for the GUI (Graphical User Interface) of the VPE Preview.

void VpeSetGUITheme(
VpeHandle *hDoc*,
int *theme*
)

VpeHandle hDoc
Document Handle

int theme
available themes are:

Constant	Value	Comment
VGUI_THEME_OFFICE2000	0	
VGUI_THEME_OFFICE2003	1	
VGUI_THEME_WHIDBEY	2	

Default:
VGUI_THEME_WHIDBEY

Remarks:
On platforms other than Windows XP, there is no visual difference between the Office2003 and the Whidbey theme. On Windows XP the Office2003 theme uses different color schemes depending on the color theme chosen in the control panel of Windows XP (Blue, Olive or Silver).

On systems with a color resolution below 16-bits it is only possible to select the Office2000 theme.

3.74 VpeSetGUILanguage

[Windows platform only]

By default, VPE chooses the language for its GUI (Graphical User Interface) by querying the current chosen language in the control panel of the system. With this property you can override that default behavior and force VPE to use a specific language.

void VpeSetGUILanguage(
 VpeHandle *hDoc*,
 int *language*
)

VpeHandle hDoc
 Document Handle

int language
 available languages are:

Constant	Value	Comment
VGUI_LANGUAGE_USER_DEFINED	-1	
VGUI_LANGUAGE_ENGLISH	0	
VGUI_LANGUAGE_GERMAN	1	
VGUI_LANGUAGE_FRENCH	2	
VGUI_LANGUAGE_DUTCH	3	
VGUI_LANGUAGE_SPANISH	4	
VGUI_LANGUAGE_DANISH	5	
VGUI_LANGUAGE_SWEDISH	6	
VGUI_LANGUAGE_FINNISH	7	
VGUI_LANGUAGE_ITALIAN	8	
VGUI_LANGUAGE_NORWEGIAN	9	
VGUI_LANGUAGE_PORTUGUESE	10	

Default:
 depends on the chosen language in the control panel of the system

Remarks:
 If you set the *GUILanguage* to *UserDefined*, you can specify freely any text in any language for each element of the GUI using VpeSetResourceString().

Example:

```
VpeSetGUILanguage(hDoc, VGUI_LANGUAGE_SPANISH)
```

forces VPE to use spanish as language for the GUI

3.75 VpeSetResourceString

[Windows platform only; not supported by the Community Edition]

If you set the GUILanguage to *UserDefined*, you can specify freely any text in any language for each element of the GUI.

void VpeSetResourceString(
VpeHandle *hDoc*,
int *resource_id*,
LPCTSTR *text*
)

VpeHandle hDoc
Document Handle

int resource_id
possible values are:

Constant	Value	Comment
VRSCID_COPY_OF	0	"Copy %u of %u"
VRSCID_PAGE_OF	1	"Page %u of %u"
VRSCID_BAND	2	"Band %u" (i.e. "stripe" = "band")
VRSCID_ERROR	3	"Error"
VRSCID_WARNING	4	"Warning"
VRSCID_PRINTING	5	"Printing"
VRSCID_CANCEL	6	"Cancel"
VRSCID_OK	7	"Ok"
Tooltips:		
VRSCID_PRINT_DOC	8	"Print the Document"
VRSCID_FULL_PAGE	9	"Show Full Page"
VRSCID_PAGE_WIDTH	10	"Fit Page Width"
VRSCID_ZOOM_IN	11	"Zoom in"
VRSCID_ZOOM_OUT	12	"Zoom out"
VRSCID_FIRST_PAGE	13	"Go to First Page"
VRSCID_PREV_PAGE	14	"Go to Previous Page"
VRSCID_NEXT_PAGE	15	"Go to Next Page"
VRSCID_LAST_PAGE	16	"Go to Last Page"
VRSCID_HELP	17	"Help on Control"
VRSCID_INFORMATION	18	"Program Information"
VRSCID_CLOSE_PREVIEW	19	"Close Preview"
VRSCID_GRID	20	"Show / Hide Grid"

VRSCID_ENTER_PAGENO	21	"enter page# with mouse-click"
VRSCID_ZOOM_FACTOR	22	"Zoom Factor"
VRSCID_STATUS	23	"Status"
VRSCID_READY	24	"Ready"
VRSCID_EMAIL	25	"eMail"
VRSCID_OPEN	26	"Open"
VRSCID_SAVE	27	"Save"
Other:		
VRSCID_ERROR_OPEN	28	"Could not open file!"
VRSCID_ERROR_WRITE	29	"Could not write file!"
Usage Help Dialog:		
VRSCID_USAGE	30	"Usage" (caption of the help-dialog)
VRSCID_MOUSE	31	"Mouse"
VRSCID_USAGE_MOUSE	32	<all mouse usage text, lines separated by "\n" (carriage return or linefeed)>
VRSCID_KEYBOARD	33	"Keyboard"
VRSCID_USAGE_KEYBOARD	34	<all keyboard usage text, lines separated by "\n" (carriage return or linefeed)>

LPCTSTR text

The text that shall be used for the specified resource.

3.76 VpeGetWindowHandle

[Windows platform only]

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 an embedded Preview Window, to control the position and the size of the Preview and for routing (sending) keyboard-messages (WM_CHAR) to it.

HWND VpeGetWindowHandle(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the window handle of the preview

3.77 VpeWindowHandle

[Windows platform only]

The same as VpeGetWindowHandle.

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 an embedded Preview Window, to control the position and the size of the Preview and for routing (sending) keyboard-messages (WM_CHAR) to it.

HWND VpeWindowHandle(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the window handle of the preview

3.78 VpeGetVersion

Returns the current version number.

int VpeGetVersion(
)

Returns:

The current version number coded as follows:

Hi Byte	= major no.	(0 - 99)
Lo Byte	= minor no.	(0 - 99)

For Example: (hex) 0x0115 = 1.21

3.79 VpeGetReleaseNumber

Returns the current release number. The release number is incremented for each new service release.

int VpeGetBuildNumber(
)

Returns:

The current release number.

3.80 VpeGetBuildNumber

Returns the current build number. The build number is incremented with each new build.

int VpeGetBuildNumber(
)

Returns:

The current build number.

3.81 VpeGetEdition

Returns the Edition of the loaded Engine.

int VpeGetEdition(
)

Returns:

possible values are:

Constant	Value	Comment
VEDITION_COMMUNITY	500	
VEDITION_STANDARD	1000	
VEDITION_ENHANCED	2000	
VEDITION_PROFESSIONAL	3000	
VEDITION_ENTERPRISE	4000	
VEDITION_INTERACTIVE	5000	

4 Printing Functions

4.1 Printing Functions

[Windows platform only]

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

4.2 VpeSetupPrinter

[Windows platform only; not supported by the Community Edition]

This is a very central function for controlling the printer. You can create a setup for the printer separate from printing. The settings for the printer can be stored/retrieved in/from a file (optional), so the settings can be used permanently.

The method is very flexible, as your end-user may make individual printer-setups for *each* kind of document. So the user can not only specify an individual printer for each different type of document, but also the printer's properties, like the number of copies, collation, output paper bin and everything else. Just let the user make these definitions in an extra part of your application and store the different settings into different files. This will give you the best control possible, and you can easily implement a "Printer Setup" function in your applications menu using this method.

The setup and storage into a file is done with a single call to SetupPrinter(), where you provide the file name as parameter and set the parameter dialog_control = PRINTDLG_ALWAYS.

For example with the call: SetupPrinter("document_type_1.prs", PRINTDLG_ALWAYS)
VPE will try to read the file "document_type_1", and load it, if it exists. Next, a dialog with the settings stored in the file (or default settings, if no file is found) will appear and your user can make all settings available in the dialog(s). If the user then pushes the OK button, the settings are stored in the file "document_type_1.prs".

Later, when printing the document of "type 1", you call OpenDoc() and then SetupPrinter(hDoc, "document_type_1.prs", PRINTDLG_ONFAIL). The flag PRINTDLG_ONFAIL will let the setup-dialog only come up, if the file "document_type_1.prs" is not found. For example if your user hasn't done the setup yet (the specified file is not found), it will be done then and only once! Otherwise, if the file is found, no dialog is shown and the printer setup - using all previously stored settings from the file - is performed "silently".

Important: VPE saves *all* settings to the setup file: name of the printer, driver, port and in addition *all* other driver-specific private data. The private printer-driver data includes every special option a printer-driver provides, even those which can not be accessed through the standard Windows API. This means that your users can set, store and use just any option of the printer!

int VpeSetupPrinter(
VpeHandle *hDoc*,
LPCSTR *file_name*,
int *dialog_control*
)

VpeHandle hDoc
Document Handle

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

int dialog_control
possible values are:

Constant Name	Value	Comment
PRINTDLG_NEVER	0	never show setup-dialog (if file_name is "", 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
PRINTDLG_FULL	4	show FULL Dialog (with page range, collation, etc.). This is a flag, i.e. add it to one of the other PRINTDLG_xyz flags (example: PRINTDLG_ALWAYS + PRINTDLG_FULL).The flag is especially useful if you hide the Toolbar or the Print-Button and you want to provide to your end-user the possibility to make a full printer setup including a page-range, etc.

Returns:

Value	Description
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
4	problem in the printing system, for example the printer used in the setup file is no longer installed in the system

Remarks:

For standardization, Printer Setup Files created with VPE should have the suffix ".PRS".

Beside using SetupPrinter(), you can read Printer Setup Files into VPE by using VpeReadPrinterSetup() and you can write them using VpeWritePrinterSetup().

If there is a problem in the printing system, for example the printer used in the setup file is no longer installed in the system, SetupPrinter() will try to select the default printer. If this is not possible, SetupPrinter() will return the value "4".

VPE is **not** using the settings of a printer for the document (otherwise it wouldn't be printer-independent). In order to reflect the settings of the printer, let the user make a printer setup - or read an existing setup from a PRS file - and assign the Device Control Properties like DevPaperFormat and DevOrientation, etc. to the current VPE Document (see PageFormat and PageOrientation). For an example, see below.

The page range is not written to a PRS File:
It does not make sense to store the page range within a generic setup file, which is intended to be used for different documents.

The settings in the PRS file for the page orientation and page format are ignored:
The **Page Orientation** (Landscape / Portrait) chosen in the Printer Setup Dialog is ignored by VPE: this is by design, because VPE gives you - the developer - the control over the printout. That means: because you can set in a VPE Document the page orientation for *each page* separately by code, the printer setting for the orientation will be overridden by the settings in the VPE document during printing.

The same rule applies to the **Page Format** (or individual page dimensions): since you can specify different page formats for *each page* in a VPE Document separately, VPE ignores the settings in a PRS file and uses the settings of the document. This has the advantage, that VPE instructs the printer at each newly printed page, to use the page dimensions of the currently printed page. Many printers can react on this and chose automatically the right paper from a different paper input bin. If you want to stop VPE from this automatism, or if you need to retrieve the page format from the PRS file, you can specify the flag PRINT_NO_AUTO_PAGE_DIMS for the property PrintOptions.

Example:

```
hDoc = VpeOpenDoc(hwnd, "Test", 0);
// use PRINT_NO_AUTO_PAGE_DIMS, in order to be able to retrieve the values
// for the page dimensions from the PRS file:
VpeSetPrintOptions(hDoc, PRINT_ALL + PRINT_NO_AUTO_PAGE_DIMS);
VpeSetupPrinter(hDoc, "personal settings.prs", PRINTDLG_ONFAIL);
VpeSetPageWidth(hDoc, VpeGetDevPaperWidth(hDoc));
VpeSetPageHeight(hDoc, VpeGetDevPaperHeight(hDoc));
VpeSetPageOrientation(hDoc, VpeGetDevOrientation(hDoc));

// Switch PrintOptions back and let VPE control the printer's page
// dimensions, i.e. clear the flag PRINT_NO_AUTO_PAGE_DIMS
VpeSetPrintOptions(hDoc, PRINT_ALL);
```

You will notice that we are using *PageWidth* and *PageHeight* in the example above. The reason is, that this circumvents a bug in many printer drivers, which return wrong values for DevPaperFormat in case a user defined format has been assigned to the printer. Because of this, the construction "VpeSetPageFormat(hDoc, VpeGetDevPaperFormat(hDoc))" **can not be used reliably**. Instead, use the code from the example above.

Printing to a tractor printer using endless paper / labels:
The Windows operating system always generates a Form Feed after a page has been printed. In order to print on endless paper or labels with a tractor printer, set the PageFormat or PageWidth and PageHeight of the VPE Document correspondingly to the dimensions of the paper / label. Since VPE will instruct the printer to use the document's page format, this will force the printer to position exactly on the next page/label after a single page/label has been printed.

4.3 VpeSetPrintOptions

[Windows platform only; not supported by the Community Edition]

Allows to set several print options. This preset is also active if the user pushes the print-button in the toolbar, or if the method VpeSetupPrinter() is called.

void VpeSetPrintOptions(
VpeHandle *hDoc*,
long *flags*
)

VpeHandle hDoc
Document Handle

long flags
a combination of the following values:

Constant Name	Value	Comment
PRINT_ALL	0	print all pages
PRINT_EVEN	1	print only even pages
PRINT_ODD	2	print only odd pages
PRINT_NOABORTDLG	4	show no abort / progress dialog during printing
PRINT_NO_AUTO_PAGE_DIMS	16	VPE shall not instruct the printer to use the page dimensions of the currently printed page

Default:
PRINT_ALL

Example:
The several different options can be set by adding the values of each flag. When you assign a new value to *PrintOptions*, any previous settings will be overwritten:

```
VpeSetPrintOptions(hDoc, PRINT_EVEN);
```

print only even numbered pages

```
VpeSetPrintOptions(hDoc, PRINT_ODD + PRINT_NOABORTDLG);
```

print only odd numbered pages and do not show the print-progress dialog

```
VpeSetPrintOptions(hDoc, PRINT_ALL + PRINT_NOABORTDLG +
PRINT_NO_AUTO_PAGE_DIMS);
```

print all pages, do not show the print-progress dialog, do not instruct the printer to use the page dimensions of the currently printed page

```
VpeSetPrintOptions(hDoc, PRINT_ALL);
```

reset to default

Remarks:

PRINT_NO_AUTO_PAGE_DIMS:

By default, VPE instructs the printer during printing, to switch the paper dimensions automatically according to the page dimensions of the VPE Document's currently printed page. Many printers can react on this and chose automatically the right paper from a different paper input bin. If you want to stop VPE from this automatism, you can specify the flag PRINT_NO_AUTO_PAGE_DIMS for this property.

Use PRINT_NO_AUTO_PAGE_DIMS also, if you require that the SetupPrinter() method retrieves the settings for the page dimensions from a PRS file.

It has been reported, that some very few printer drivers have a bug and do not print in duplex mode, unless you use PRINT_NO_AUTO_PAGE_DIMS.

In order to let the printer accept the automatic selection of user defined page formats (i.e. you are **not** specifying PRINT_NO_AUTO_PAGE_DIMS), it might be necessary to define a form of the desired page format.

Example: with "Start Menu | Settings | Printers" the window with the installed printers will appear. Right-click on a blank area of the window and choose "Server Properties" from the pop-up menu. In the upcoming dialog, define a custom form of the desired dimensions. For example name it "Test" and set the width to 760 and height to 1280.

In your source code, select the printer and the desired page format like this:

```
VpeSetDevice(hDoc, "Epson LQ-550");
VpeSetPageWidth(hDoc, 760);
VpeSetPageHeight(hDoc, 1280);
```

That's it.

4.4 VpeSetPrintPosMode

[Windows platform only; not supported by the Community Edition]

Specifies, how objects are positioned on the printer.

```
void VpeSetPrintPosMode(
    VpeHandle hDoc,
    int  mode
)
```

VpeHandle hDoc

Document Handle

int mode

possible values for mode are:

Constant Name	Value	Comment
PRINTPOS_ABSOLUTE	0	absolute to the paper-edges (default)
PRINTPOS_RELATIVE	1	relative to the first printable position of the printer

Default:

PRINTPOS_ABSOLUTE

See also:

"Positioning On the Printer" in the Programmer's Manual and
VpeSetPrintOffset

4.5 VpeSetPrintOffset

[Windows platform only; not supported by the Community Edition]

Allows to set a printing offset by code. This offset will be added to all objects in the document while printing. It is especially useful for calibrating printers that return wrong information about their unprintable area.

void VpeSetPrintOffset(
  VpeHandle *hDoc*,
  VpeCoord *offset_x*,
  VpeCoord *offset_y*
)

VpeHandle hDoc
  Document Handle

VpeCoord offset_x, offset_y
  the offset in metric or inch units, values can be positive or negative

Default:

offset_x	= 0	(no offset)
offset_y	= 0	(no offset)

See also:

  "Positioning On the Printer" in the Programmer's Manual and
  VpeSetPrintPosMode()

4.6 VpeSetPrintOffsetX

[Windows platform only; not supported by the Community Edition]

Allows to set a printing offset by code. This offset will be added to all objects in the document while printing. It is especially useful for calibrating printers that return wrong information about their unprintable area.

void VpeSetPrintOffsetX(
    VpeHandle *hDoc*,
    VpeCoord *offset_x*
)

VpeHandle hDoc
    Document Handle

VpeCoord offset_x
    the x-offset in metric or inch units, values can be positive or negative

Default:
    0 (no offset)

See also:
    "Positioning On the Printer" in the Programmer's Manual and
    VpeSetPrintPosMode()

4.7 VpeGetPrintOffsetX

[Windows platform only; not supported by the Community Edition]

Returns the x-printing offset.

VpeCoord VpeGetPrintOffsetX(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

4.8 VpeSetPrintOffsetY

[Windows platform only; not supported by the Community Edition]

Allows to set a printing offset by code. This offset will be added to all objects in the document while printing. It is especially useful for calibrating printers that return wrong information about their unprintable area.

void VpeSetPrintOffsetY(
    VpeHandle *hDoc*,
    VpeCoord *offset_y*
)

VpeHandle hDoc
    Document Handle

VpeCoord offset_y
    the y-offset in metric or inch units, values can be positive or negative

Default:
    0 (no offset)

See also:
    "Positioning On the Printer" in the Programmer's Manual and
    VpeSetPrintPosMode()

4.9 VpeGetPrintOffsetY

[Windows platform only; not supported by the Community Edition]

Returns the y-printing offset.

VpeCoord VpeGetPrintOffsetY(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

4.10 VpeSetPrintScale

[Windows platform only; Professional Edition and higher only]

This property sets the scale for printing. It does not affect the scale in the preview.

Since the X- and Y- dimensions are scaled simultaneously, this property changes the width and the height of the printout at the same time. The change of the PrintScale implies a change of the positions of all object's contained in the document.

This might lead to problems regarding the positioning of the topmost / leftmost objects:
if for example a text is placed at the position (2, 2) and the PrintScale is set to 0.75 then the position of the text is changed to 0.75 * (2, 2) = (1.5, 1.5). If you have placed objects very near to the unprintable area, or if you have set the PrintScale to a very small value, objects might be moved into the unprintable area and clipped.

This can be corrected by setting the PrintOffsetX- and PrintOffsetY- properties. By using these properties you can align the printout anywhere on the paper, even on the center.

The PrintScale property is not stored in VPE Document files. Therefore VPEView - the Document Viewer - will print all documents unscaled.

void VpeSetPrintScale(
VpeHandle *hDoc*,
double *scale*
)

VpeHandle hDoc
Document Handle

double scale
the scaling factor, must be between 0.01 and 6.0

Default:
1.0 which means the printout is unscaled (the scaling factor is 1:1)

Remarks:
The setting for the PrintScale is not stored in VPE Document files.

Example:

```
VpeSetPrintScale(hDoc, 0.5);
```

Sets the print scale to 0.5 : 1, i.e. the printout is half the size than the original size.

```
VpeSetPrintScale(hDoc, 4.0);
```

Sets the print scale to 4 : 1, i.e. the printout is four times bigger than the original size.

See also:
"Positioning On the Printer" in the Programmer's Manual

4.11 VpeGetPrintScale

[Windows platform only; Professional Edition and higher only]

Returns the current value of the print scale.

double VpeGetPrintScale(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current value of the print scale

4.12 VpePrintDoc

[Windows platform only; not supported by the Community Edition]

Prints the document. You may not close your application or the document **until** VPE has finished all print-jobs. 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 window messages.

During printing, VPE will disable the parent window of the preview supplied as parameter in VpeOpenDoc(). If the parent window is not the main window of your application, or if your application is in any other way able to receive and to respond to messages, it is your responsibility to disable your application, or to take care that your print-functions (for example a button or a menu entry that invokes printing) are stopped from being reentered.

void VpePrintDoc(
VpeHandle *hDoc*,
int *with_setup*
)

VpeHandle hDoc
Document Handle

int with_setup

Value	Description
True	show printer setup-dialog
False	do not show printer setup-dialog

Remarks:
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 event). But due to the event-driven mechanism of Windows, your application window is still "alive" and other actions can take place (like button clicks), and the related code will be executed.

4.13 VpeIsPrinting

[Windows platform only]

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.

int VpeIsPrinting(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
True	the Engine is currently printing
False	the Engine is currently not printing

5 Device Control Properties

5.1 Device Control Properties

[Windows platform only; not supported by the Community Edition]

VPE offers a large set of properties and methods to give you all control over the printing device. By default, VPE selects now the default (standard) output device after a call to OpenDoc().

Remarks:

- Changing the properties (like DevBin, DevOrientation, DevPaperFormat, etc.) during the print-job doesn't work with some (buggy) printer drivers, for example some fax drivers and the HP4 M Plus driver (the HP4 PS driver should work with the HP4 M Plus printer!).
- Some properties and methods don't work with some printer drivers. For example " DevTTOption" doesn't work with our HP4 and HP5 printer drivers on WfW 3.11 and NT 3.51, but it works with both drivers on Win95. This is a driver problem.
- Printer drivers are manufactured by vendors independent of IDEAL Software; we make no warranty, implied or otherwise, regarding these product's performance or reliability.
- Win32s is not officially supported by VPE. The Sophisticated Printer Control does not work with Win32s.

5.2 VpeDevEnum

[Windows platform only; not supported by the Community Edition]

Initialize enumeration of all available output devices. After this method has been called, you can retrieve each single device name with the method VpeGetDevEntry().

int VpeDevEnum(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  the number of printing devices installed in the system

Remarks:
  Do not call VpeDevEnum() in a loop like "for x = 0 to VpeDevEnum(hDoc) - 1", because each time VPE will check for all available devices, which is time-consuming. Rather assign the value of VpeDevEnum() to a variable.

Example:
  see VpeGetDevEntry()

5.3 VpeGetDevEntry

[Windows platform only; not supported by the Community Edition]

After calling VpeDevEnum(), you are able to retrieve the names of all printing devices installed on the system with this method. With each call you retrieve one entry.

void VpeGetDevEntry(
VpeHandle *hDoc*,
int *index*,
LPSTR *device*,
int *size*
)

VpeHandle hDoc
Document Handle

int index
must be in the range between 0 and the value returned by VpeDevEnum() - 1

LPSTR device
receive string for the device name

int size
maximum size of the receive string in characters

Returns:
The name of the device specified by "index". If index is out of range, an empty string ("") will be returned.

Remarks:
This method works only correctly, if you called VpeDevEnum() before.

Example:

```
long i, count
char s[256]
count = VpeDevEnum(hDoc) - 1   // initialize enumeration
for i = 0 to count
   VpeGetDevEntry(hDoc, i, s, sizeof(s))
   MessageBox(s) // show each available device in a separate message box
next i
```

Note:
Do not call VpeDevEnum() repeatedly, because each time VPE will query all printers from the system.

Wrong Example:

```
char s[256];
for i = 0 to VpeDevEnum(hDoc) - 1
   VpeGetDevEntry(hDoc, i, s, sizeof(s))
   MessageBox(s) // show each available device in a separate message box
Next i
```

5.4 VpeSetDevice

[Windows platform only; not supported by the Community Edition]

Set the current output device. After a new output device is successfully selected, **all device settings (like DevPaperFormat(), etc.) are lost!** You are also able to select the default output device by setting device to NULL or empty ("").

int VpeSetDevice(
    VpeHandle *hDoc*,
    LPCSTR *device*
)

VpeHandle hDoc
    Document Handle

LPCSTR device
    the name of the output device

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set to VERR_COMMON. An error may occur in case the specified device is not present or not available for some reason.

You can enumerate the available devices with VpeDevEnum().

Example:

```
VpeSetDevice(hDoc, "Office Printer")
```

Selects the device installed as "Office Printer" in the system.

```
VpeSetDevice(hDoc, "") or VpeSetDevice(hDoc, NULL)
```

Selects the default device of the system.

5.5 VpeGetDevice

[Windows platform only; not supported by the Community Edition]

Retrieves the current output device.

void VpeGetDevice(
VpeHandle *hDoc*,
LPSTR *device*,
int *size*
)

VpeHandle hDoc
Document Handle

LPSTR device
the name of the output device

int size
Specifies the maximum number of characters to copy to the buffer (the string "device"). If the text exceeds this limit, it is truncated.

Remarks:
VPE selects by default the standard (default) output device when OpenDoc() is called.

In case of an error, LastError is set to VERR_COMMON. An error may occur in case a device is not present or not available for some reason.

5.6 VpeSetDevOrientation

[Windows platform only; not supported by the Community Edition]

Sets the orientation of the currently selected output device to portrait or landscape. Setting this property does NOT affect the setting of the preview (see VpeSetPageOrientation()).

int VpeSetDevOrientation(
  VpeHandle *hDoc*,
  int *orientation*
)

VpeHandle hDoc
  Document Handle

int orientation
  possible values are:

Constant Name	Value	Comment
VORIENT_PORTRAIT	1	
VORIENT_LANDSCAPE	2	

Returns:

Value	Description
True	success
False	failure

Remarks:

The DevOrientation property is available for total control. It is only useful if called while processing the VPE_PRINT_NEWPAGE event. Otherwise the DevOrientation property is always overridden by the PageOrientation property. **You should always use VpeSetPageOrientation() to change the orientation for the preview and the printer.**

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support changing the orientation.

Example:

```
VpeSetDevOrientation(hDoc, VORIENT_LANDSCAPE)
```

5.7 VpeGetDevOrientation

[Windows platform only; not supported by the Community Edition]

Retrieves the orientation of the currently selected output device.

int VpeGetDevOrientation(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Constant Name	Value	Comment
VORIENT_PORTRAIT	1	
VORIENT_LANDSCAPE	2	
False	0	failure

Remarks:
In case of an error, LastError is set to VERR_COMMON.

5.8 VpeSetDevPaperFormat

[Windows platform only; not supported by the Community Edition]

Sets the paper format of the currently selected output device to one of the predefined VPAPER_xyz formats. You may only use one of the VPAPER_xyz constants as a valid value. Setting this property does NOT affect the setting of the preview (see VpeSetPageFormat()).

Setting this property instructs a printer with multiple paper bins to select the input bin which contains paper of the appropriate paper format. This property is also useful for tractor printers with endless paper feeders.

int VpeSetDevPaperFormat(
    VpeHandle *hDoc*,
    int *format*
)

VpeHandle hDoc
    Document Handle

int format
    one of the predefined VPAPER_xyz constants (see below)

Returns:

Value	Description
True	success
False	failure

Remarks:

This property is available for total control. It is only useful if called while processing the VPE_PRINT_NEWPAGE event. Otherwise the DevPaperFomat property is always overridden by the PageFormat property. **You should always use VpeSetPageFormat() to change the page format for the preview and the printer.**

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support changing the paper size at all, or not to the specified format.

We experienced, that this property doesn’t work with some printer drivers. If your printer does not respond to these settings, it is a printer driver problem.

Example:

```
VpeSetDevPaperFormat(hDoc, VPAPER_CSHEET)     // C Sheet, 17- by 22-inches
```

***page_format* can be one of the following values**

Constant	Value	Comment
VPAPER_USER_DEFINED	0	User-Defined
VPAPER_A4	-1	A4 Sheet, 210- by 297-millimeters
VPAPER_LETTER	-2	US Letter, 8 1/2- by 11-inches
VPAPER_LEGAL	-3	Legal, 8 1/2- by 14-inches
VPAPER_CSHEET	-4	C Sheet, 17- by 22-inches

VPAPER_DSHEET	-5	D Sheet, 22- by 34-inches
VPAPER_ESHEET	-6	E Sheet, 34- by 44-inches
VPAPER_LETTERSMALL	-7	Letter Small, 8 1/2- by 11-inches
VPAPER_TABLOID	-8	Tabloid, 11- by 17-inches
VPAPER_LEDGER	-9	Ledger, 17- by 11-inches
VPAPER_STATEMENT	-10	Statement, 5 1/2- by 8 1/2-inches
VPAPER_EXECUTIVE	-11	Executive, 7 1/4- by 10 1/2-inches
VPAPER_A3	-12	A3 sheet, 297- by 420-millimeters
VPAPER_A4SMALL	-13	A4 small sheet, 210- by 297-millimeters
VPAPER_A5	-14	A5 sheet, 148- by 210-millimeters
VPAPER_B4	-15	B4 sheet, 250- by 354-millimeters
VPAPER_B5	-16	B5 sheet, 182- by 257-millimeter paper
VPAPER_FOLIO	-17	Folio, 8 1/2- by 13-inch paper
VPAPER_QUARTO	-18	Quarto, 215- by 275-millimeter paper
VPAPER_10X14	-19	10- by 14-inch sheet
VPAPER_11X17	-20	11- by 17-inch sheet
VPAPER_NOTE	-21	Note, 8 1/2- by 11-inches
VPAPER_ENV_9	-22	#9 Envelope, 3 7/8- by 8 7/8-inches
VPAPER_ENV_10	-23	#10 Envelope, 4 1/8- by 9 1/2-inches
VPAPER_ENV_11	-24	#11 Envelope, 4 1/2- by 10 3/8-inches
VPAPER_ENV_12	-25	#12 Envelope, 4 3/4- by 11-inches
VPAPER_ENV_14	-26	#14 Envelope, 5- by 11 1/2-inches
VPAPER_ENV_DL	-27	DL Envelope, 110- by 220-millimeters
VPAPER_ENV_C5	-28	C5 Envelope, 162- by 229-millimeters
VPAPER_ENV_C3	-29	C3 Envelope, 324- by 458-millimeters
VPAPER_ENV_C4	-30	C4 Envelope, 229- by 324-millimeters
VPAPER_ENV_C6	-31	C6 Envelope, 114- by 162-millimeters
VPAPER_ENV_C65	-32	C65 Envelope, 114- by 229-millimeters
VPAPER_ENV_B4	-33	B4 Envelope, 250- by 353-millimeters
VPAPER_ENV_B5	-34	B5 Envelope, 176- by 250-millimeters
VPAPER_ENV_B6	-35	B6 Envelope, 176- by 125-millimeters
VPAPER_ENV_ITALY	-36	Italy Envelope, 110- by 230-millimeters
VPAPER_ENV_MONARCH	-37	Monarch Envelope, 3 7/8- by 7 ½-inches
VPAPER_ENV_PERSONAL	-38	6 3/4 Envelope, 3 5/8- by 6 1/2-inches
VPAPER_FANFOLD_US	-39	US Std Fanfold, 14 7/8- by 11-inches
VPAPER_FANFOLD_STD_GERMAN	-40	German Std Fanfold, 8 1/2- by 12-inches
VPAPER_FANFOLD_LGL_GERMAN	-41	German Legal Fanfold, 8 1/2- by 13-inches
VPAPER_ISO_B4	-42	B4 (ISO) 250 x 353 mm
VPAPER_JAPANESE_POSTCARD	-43	Japanese Postcard 100 x 148 mm
VPAPER_9X11	-44	9 x 11 in
VPAPER_10X11	-45	10 x 11 in
VPAPER_15X11	-46	15 x 11 in
VPAPER_ENV_INVITE	-47	Envelope Invite 220 x 220 mm
VPAPER_RESERVED_48	-48	RESERVED--DO NOT USE
VPAPER_RESERVED_49	-49	RESERVED--DO NOT USE
VPAPER_LETTER_EXTRA	-50	Letter Extra 9 \275 x 12 in
VPAPER_LEGAL_EXTRA	-51	Legal Extra 9 \275 x 15 in
VPAPER_TABLOID_EXTRA	-52	Tabloid Extra 11.69 x 18 in

VPAPER_A4_EXTRA	-53	A4 Extra 9.27 x 12.69 in
VPAPER_LETTER_TRANSVERSE	-54	Letter Transverse 8 \275 x 11 in
VPAPER_A4_TRANSVERSE	-55	A4 Transverse 210 x 297 mm
VPAPER_LETTER_EXTRA_TRANSVERSE	-56	Letter Extra Transverse 9\275 x 12 in
VPAPER_A_PLUS	-57	SuperA/SuperA/A4 227 x 356 mm
VPAPER_B_PLUS	-58	SuperB/SuperB/A3 305 x 487 mm
VPAPER_LETTER_PLUS	-59	Letter Plus 8.5 x 12.69 in
VPAPER_A4_PLUS	-60	A4 Plus 210 x 330 mm
VPAPER_A5_TRANSVERSE	-61	A5 Transverse 148 x 210 mm
VPAPER_B5_TRANSVERSE	-62	B5 (JIS) Transverse 182 x 257 mm
VPAPER_A3_EXTRA	-63	A3 Extra 322 x 445 mm
VPAPER_A5_EXTRA	-64	A5 Extra 174 x 235 mm
VPAPER_B5_EXTRA	-65	B5 (ISO) Extra 201 x 276 mm
VPAPER_A2	-66	A2 420 x 594 mm
VPAPER_A3_TRANSVERSE	-67	A3 Transverse 297 x 420 mm
VPAPER_A3_EXTRA_TRANSVERSE	-68	A3 Extra Transverse 322 x 445 mm
Windows 2000 or Higher:		
VPAPER_DBL_JAPANESE_POSTCARD	-69	Japanese Double Postcard 200 x 148 mm
VPAPER_A6	-70	A6 105 x 148 mm
VPAPER_JENV_KAKU2	-71	Japanese Envelope Kaku #2
VPAPER_JENV_KAKU3	-72	Japanese Envelope Kaku #3
VPAPER_JENV_CHOU3	-73	Japanese Envelope Chou #3
VPAPER_JENV_CHOU4	-74	Japanese Envelope Chou #4
VPAPER_LETTER_ROTATED	-75	Letter Rotated 11 x 8 1/2 in
VPAPER_A3_ROTATED	-76	A3 Rotated 420 x 297 mm
VPAPER_A4_ROTATED	-77	A4 Rotated 297 x 210 mm
VPAPER_A5_ROTATED	-78	A5 Rotated 210 x 148 mm
VPAPER_B4_JIS_ROTATED	-79	B4 (JIS) Rotated 364 x 257 mm
VPAPER_B5_JIS_ROTATED	-80	B5 (JIS) Rotated 257 x 182 mm
VPAPER_JAPANESE_POSTCARD_ROTATED	-81	Japanese Postcard Rotated 148 x 100 mm
VPAPER_DBL_JAPANESE_POSTCARD_ROTATED	-82	Double Japanese Postcard Rotated 148 x 200mm
VPAPER_A6_ROTATED	-83	A6 Rotated 148 x 105 mm
VPAPER_JENV_KAKU2_ROTATED	-84	Japanese Envelope Kaku #2 Rotated
VPAPER_JENV_KAKU3_ROTATED -85	-85	Japanese Envelope Kaku #3 Rotated
VPAPER_JENV_CHOU3_ROTATED	-86	Japanese Envelope Chou #3 Rotated
VPAPER_JENV_CHOU4_ROTATED	-87	Japanese Envelope Chou #4 Rotated
VPAPER_B6_JIS	-88	B6 (JIS) 128 x 182 mm
VPAPER_B6_JIS_ROTATED	-89	B6 (JIS) Rotated 182 x 128 mm
VPAPER_12X11	-90	12 x 11 in
VPAPER_JENV_YOU4	-91	Japanese Envelope You #4
VPAPER_JENV_YOU4_ROTATED	-92	Japanese Envelope You #4 Rotated
VPAPER_P16K	-93	PRC 16K 146 x 215 mm
VPAPER_P32K	-94	PRC 32K 97 x 151 mm
VPAPER_P32KBIG	-95	PRC 32K(Big) 97 x 151 mm
VPAPER_PENV_1	-96	PRC Envelope #1 102 x 165 mm
VPAPER_PENV_2	-97	PRC Envelope #2 102 x 176 mm
VPAPER_PENV_3	-98	PRC Envelope #3 125 x 176 mm
VPAPER_PENV_4	-99	PRC Envelope #4 110 x 208 mm

VPAPER_PENV_5	-100	PRC Envelope #5 110 x 220 mm
VPAPER_PENV_6	-101	PRC Envelope #6 120 x 230 mm
VPAPER_PENV_7	-102	PRC Envelope #7 160 x 230 mm
VPAPER_PENV_8	-103	PRC Envelope #8 120 x 309 mm
VPAPER_PENV_9	-104	PRC Envelope #9 229 x 324 mm
VPAPER_PENV_10	-105	PRC Envelope #10 324 x 458 mm
VPAPER_P16K_ROTATED	-106	PRC 16K Rotated
VPAPER_P32K_ROTATED	-107	PRC 32K Rotated
VPAPER_P32KBIG_ROTATED	-108	PRC 32K(Big) Rotated
VPAPER_PENV_1_ROTATED	-109	PRC Envelope #1 Rotated 165 x 102 mm
VPAPER_PENV_2_ROTATED	-110	PRC Envelope #2 Rotated 176 x 102 mm
VPAPER_PENV_3_ROTATED	-111	PRC Envelope #3 Rotated 176 x 125 mm
VPAPER_PENV_4_ROTATED	-112	PRC Envelope #4 Rotated 208 x 110 mm
VPAPER_PENV_5_ROTATED	-113	PRC Envelope #5 Rotated 220 x 110 mm
VPAPER_PENV_6_ROTATED	-114	PRC Envelope #6 Rotated 230 x 120 mm
VPAPER_PENV_7_ROTATED	-115	PRC Envelope #7 Rotated 230 x 160 mm
VPAPER_PENV_8_ROTATED	-116	PRC Envelope #8 Rotated 309 x 120 mm
VPAPER_PENV_9_ROTATED	-117	PRC Envelope #9 Rotated 324 x 229 mm
VPAPER_PENV_10_ROTATED	-118	PRC Envelope #10 Rotated 458 x 324 mm

5.9 VpeGetDevPaperFormat

[Windows platform only; not supported by the Community Edition]

Retrieves the selected paper format of the currently selected output device.

int VpeGetDevPaperFormat(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    one of the predefined VPAPER_xyz constants
    or 1 = failure

Remarks:
    In case of an error, LastError is set to VERR_COMMON.

5.10 VpeSetDevPaperWidth

[Windows platform only; not supported by the Community Edition]

Sets an individual paper width for the currently selected output device to the specified value. Setting this property does NOT affect the setting of the preview. If you set the width, you should also set the height. If this call is successful, VpeGetDevPaperFormat() returns VPAPER_USER_DEFINED (=0).

int VpeSetDevPaperWidth(
    VpeHandle *hDoc*,
    VpeCoord *width*
)

VpeHandle hDoc
    Document Handle

VpeCoord width
    the paper width

Returns:

Value	Description
True	Success
False	Failure

Remarks:

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support changing the paper width.

This property is available for total control. It is only useful if called while processing the VPE_PRINT_NEWPAGE event. Otherwise this property is always overridden by the PageFormat or PageWidth property.

We experienced that this option doesn’t work with some printer drivers. We tested this for example on an Epson LQ-550 dot-matrix printer and it didn’t work on WfW 3.11 and NT 3.51, but it worked on Win95. But we heared about, that a LQ-510 printer driver will solve the problem for the LQ-550. So if your printer does not respond to this setting, it is a printer driver problem. In that case you can try to use another compatible printer driver, or see if the printer driver responds to setting the paper size with the property DevPaperFormat to a predefined paper format.

Example:

```
VpeSetDevPaperWidth(hDoc, 10.5)     // 10.5 cm / inch
```

5.11 VpeGetDevPaperWidth

[Windows platform only; not supported by the Community Edition]

Retrieves the paper width of the currently selected output device.

VpeCoord VpeGetDevPaperWidth(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the setting of the paper width for the current selected device
    or False (0) = failure

Remarks:
    In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support changing the paper width.

5.12 VpeSetDevPaperHeight

[Windows platform only; not supported by the Community Edition]

Sets an individual paper height for the currently selected output device to the specified value. Setting this property does NOT affect the setting of the preview. If you set the height, you should also set the width. If this call is successful, VpeGetDevPaperFormat() returns VPAPER_USER_DEFINED (=0).

Setting this property makes sense for tractor printers with endless paper feeders.

int VpeSetDevPaperHeight(
    VpeHandle hDoc,
    VpeCoord *height*
)

VpeHandle hDoc
    Document Handle

VpeCoord height
    the paper height

Returns:

Value	Description
True	Success
False	Failure

Remarks:

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support changing the paper height.

This property is available for total control. It is only useful if called while processing the VPE_PRINT_NEWPAGE event. Otherwise this property is always overridden by the PageFormat or PageHeight property.

We experienced that this option doesn't work with some printer drivers. We tested this for example on an Epson LQ-550 dot-matrix printer and it didn't work on WfW 3.11 and NT 3.51, but it worked on Win95. But we heared about, that a LQ-510 printer driver will solve the problem for the LQ-550. So if your printer does not respond to this setting, it is a printer driver problem. In that case you can try to use another compatible printer driver, or see if the printer driver responds to setting the paper size with the property DevPaperFormat to a predefined paper format.

Example:

```
VpeSetDevPaperHeight(hDoc, 5.8)     // 5.8 cm / inch
```

5.13 VpeGetDevPaperHeight

[Windows platform only; not supported by the Community Edition]

Retrieves the individual paper height of the currently selected output device.

VpeCoord VpeGetDevPaperHeight(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the setting of the paper width for the current selected device or False (= 0) = failure

Remarks:
    In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support changing the paper height.

5.14 VpeSetDevScalePercent

[Windows platform only; not supported by the Community Edition]

Sets the printing scale of the currently selected output device to the specified value. Setting this property does NOT affect the setting of the preview. The value you specify is in percent.

int VpeSetDevScalePercent(
VpeHandle *hDoc*,
int *scale*
)

VpeHandle hDoc
Document Handle

int scale
the scaling (e.g. 100 = 100%; 25 = 25%)

Returns:

Value	Description
True	success
False	failure

Remarks:
In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support changing the scale.

5.15 VpeGetDevScalePercent

[Windows platform only; not supported by the Community Edition]

Retrieves / sets the printing scale of the currently selected output device to the specified value. Setting this property does NOT affect the setting of the preview. The value you specify is in percent.

int VpeGetDevScalePercent(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  the scaling (e.g. 100 = 100%; 25 = 25%)
  or False (= 0) = failure

Remarks:
  In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support changing the scale.

5.16 VpeSetDevPrintQuality

[Windows platform only; not supported by the Community Edition]

Sets the print quality of the currently selected output device to the specified value or it sets the x-resolution in DPI.

int VpeSetDevPrintQuality(
VpeHandle *hDoc*,
int *quality*
)

VpeHandle hDoc
Document Handle

int quality
possible values are:

Constant Name	**Value**	**Comment**
VRES_DRAFT	-1	Draft Quality
VRES_LOW	-2	Low Quality
VRES_MEDIUM	-3	Medium Quality
VRES_HIGH	-4	High Quality

If a positive value is given, it specifies the number of dots per inch (DPI) for the x-resolution and is therefore device dependent. If you are able to set DevYResolution without error, this property should specify the x-resolution in DPI.

Returns:

Value	**Description**
True	success
False	failure

Remarks:

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support setting the print quality or x-resolution. You should read this property's value after setting it, to be sure the value has been accepted. Sometimes the value has not been accepted, but LastError returns no error state (= VERR_OK).

We experienced that some drivers do not allow setting the y-resolution, BEFORE the x-resolution had been changed and vice versa. Also some drivers only accept the same values for both resolutions.

Example:

```
VpeSetDevPrintQuality(hDoc, VRES_DRAFT) // Draft Mode
VpeSetDevPrintQuality(hDoc, 300)             // 300 DPI
```

5.17 VpeGetDevPrintQuality

[Windows platform only; not supported by the Community Edition]

Retrieves the print quality of the currently selected output device.

int VpeGetDevPrintQuality(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Value	**Description**
<>0	success
False (0)	failure

Remarks:

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support changing print quality.

5.18 VpeSetDevYResolution

[Windows platform only; not supported by the Community Edition]

Sets the y-resolution in dots per inch for the currently selected output device. If the output device initializes this property, the DevPrintQuality property specifies the x-resolution, in dots per inch (DPI), of the printer.

int VpeSetDevYResolution(
  VpeHandle *hDoc*,
  int *yres*
)

VpeHandle hDoc
  Document Handle

int yres
  the y-resolution in DPI

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support setting the y-resolution. You should read this property's value after setting it, to be sure the value has been accepted. Sometimes the value has not been accepted, but LastError returns no error state (= VERR_OK).

We experienced that some drivers do not allow setting the y-resolution, BEFORE the x-resolution had been changed and vice versa. Also some drivers only accept the same values for both resolutions.

Example:

```
VpeSetDevPrintQuality(hDoc, 300)    // 300 DPI
VpeSetDevYResolution(hDoc, 300)     // 300 DPI
```

5.19 VpeGetDevYResolution

[Windows platform only; not supported by the Community Edition]

Retrieves the y-resolution in dots per inch of the currently selected output device.

int VpeGetDevYResolution(
 VpeHandle *hDoc*
)

VpeHandle hDoc
 Document Handle

Returns:
 the y-resolution in DPI
 or False (0) = failure

Remarks:
 In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support setting the y-resolution.

5.20 VpeSetDevColor

[Windows platform only; not supported by the Community Edition]

Sets the color mode for the currently selected output device, only useful, if the output device is a color printer.

int VpeSetDevColor(
    VpeHandle *hDoc*,
    int *color*
)

VpeHandle hDoc
    Document Handle

int color
    possible values are:

Constant Name	Value	Comment
VCOLOR_MONOCHROME	1	
VCOLOR_COLOR	2	

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support setting the color mode.

Example:

```
VpeSetDevColor(hDoc, VCOLOR_MONOCHROME)
```

5.21 VpeGetDevColor

[Windows platform only; not supported by the Community Edition]

Retrieves the color mode for the currently selected output device.

int VpeGetDevColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    possible values are:

Constant Name	**Value**	**Comment**
VCOLOR_MONOCHROME	1	
VCOLOR_COLOR	2	
False	0	failure

Remarks:
    In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support setting the color mode.

5.22 VpeSetDevDuplex

[Windows platform only; not supported by the Community Edition]

Selects duplex (or double-sided) printing for the currently selected output device (if it is capable of duplex printing).

int VpeSetDevDuplex(
VpeHandle *hDoc*,
int *duplex*
)

VpeHandle hDoc
Document Handle

int duplex
possible values are:

Constant Name	Value	Comment
VDUP_SIMPLEX	1	
VDUP_VERTICAL	2	
VDUP_HORIZONTAL	3	

Returns:

Value	Description
True	success
False	failure

Remarks:
In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support setting the duplex mode.

Example:

```
VpeSetDevDuplex(hDoc, VDUP_VERTICAL)
```

5.23 VpeGetDevDuplex

[Windows platform only; not supported by the Community Edition]

Retrieves the duplex mode of the currently selected output device.

int VpeGetDevDuplex(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    possible values are:

Constant Name	Value	Comment
VDUP_SIMPLEX	1	
VDUP_VERTICAL	2	
VDUP_HORIZONTAL	3	
False	0	failure

Remarks:
    In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support setting the duplex mode.

5.24 VpeSetDevTTOption

[Windows platform only; not supported by the Community Edition]

Specifies how TrueType® fonts should be printed on the currently selected output device.

int VpeSetDevTTOption(
    VpeHandle *hDoc*,
    int *option*
)

VpeHandle hDoc
    Document Handle

int option
    possible values are:

Constant Name	Value	Comment
VTT_BITMAP	1	Prints TrueType fonts as graphics. This is the default action for dot-matrix printers.
VTT_DOWNLOAD	2	Downloads TrueType fonts as soft fonts. This is the default action for Hewlett-Packard printers that use Printer Control Language (PCL).
VTT_SUBDEV	3	Substitute device fonts for TrueType fonts. This is the default action for PostScript® printers.

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support setting the TTOption. Setting this property doesn't work with some printer drivers. For example it doesn't work with our HP4 and HP5 printer drivers on WfW 3.11 and NT 3.51, but it works with both drivers on Win95. This is a driver problem.

Example:

```
VpeSetDevTTOption(hDoc, VTT_BITMAP)
```

The following note is from Microsoft itself (MSDN column "Ask Dr. Gui")

These options are usually available in the printer properties dialog box under the **Fonts**, **Device Options**, or **Advanced Settings** tabs. Printer drivers for HP printers have an option called "Text As Graphics." This option, if turned on, prevents use of device fonts and draws the text using the operating system version of the font. PostScript printer drivers sometimes have options that are set per font and they usually have options to only download the font rather than rasterize it. Whenever these options are selected, the print job will get larger because the rasterized glyphs of the font are included within it.

It should save you lots of time and frustration to note that printer drivers are more different than alike, and that these settings are private settings for each printer. It may be necessary to explore the printer's settings to find the one that does the trick.

In theory, how a printer driver works with a TrueType font on the device is controllable by the member of a **DEVMODE** structure. In practice, however, Dr. GUI has diagnosed plenty of ill printer drivers that do not use this member of the **DEVMODE** structure correctly. This is a pity, because it places the burden of configuring the printer to use the operating system's fonts on the shoulders of an application's users.

5.25 VpeGetDevTTOption

[Windows platform only; not supported by the Community Edition]

Retrieves how TrueType® fonts are printed on the currently selected output device.

int VpeGetDevTTOption(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    possible values are:

Constant Name	Value	Comment
VTT_BITMAP	1	Prints TrueType fonts as graphics..
VTT_DOWNLOAD	2	Downloads TrueType fonts as soft fonts.
VTT_SUBDEV	3	Substitute device fonts for TrueType fonts.
False	0	failure

Remarks:
    In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support setting the TTOption.

5.26 VpeDevEnumPaperBins

[Windows platform only; not supported by the Community Edition]

Initialize enumeration of all available paper bins for the currently selected output device. After this function has been called, you can retrieve each single paper bin name with the method VpeGetDevPaperBinName().

int VpeDevEnumPaperBins(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  number of available paper bins of the currently selected output device

Remarks:
  In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support multiple paper bins.

  Do not call VpeDevEnumPaperBins() in a loop like "for x = 0 to VpeDevEnumPaperBins() - 1", because each time VPE will check for all available devices, which is time-consuming. Rather assign the value of VpeDevEnumPaperBins() to a variable.

  Not all of the bin options are available on every printer. Check the printer's documentation for more specific descriptions of these options.

Example:
  see VpeGetDevPaperBinName

5.27 VpeGetDevPaperBinName

[Windows platform only; not supported by the Community Edition]

Returns the paper bin name specified by "index" of the currently selected output device. If index is out of range, an empty string ("") will be returned. This method only works correctly if you called VpeDevEnumPaperBins() before.

void VpeGetDevPaperBinName(
  VpeHandle *hDoc*,
  int *index*,
  LPSTR *bin_name*,
  int *size*
)

VpeHandle hDoc
  Document Handle

int index
  must be in the range between 0 and the value returned by VpeDevEnumPaperBins() - 1

LPSTR bin_name
  receive string for the paper bin name

int size
  maximum size of the receive string "bin_name"

Returns:
  the Paper Bin name in parameter "bin_name"

Remarks:
  In case of an error, LastError is set to VERR_COMMON. This method only works correctly if you called VpeDevEnumPaperBins() before.

Example:

```
char s[256]
count = VpeDevEnumPaperBins(hDoc) - 1   // initialize enumeration
for i = 0 to count
   VpeGetDevPaperBinName(hDoc, i, s, sizeof(s))
   MessageBox(s)   // show each available bin in a separate message box
next i
```

Do not call VpeDevEnumPaperBins() repeatedly, because each time VPE will query all available paper bins of the currently selected output device from the system.

WRONG EXAMPLE:

```
char s[256]
for i = 0 to VpeDevEnumPaperBins(hDoc) - 1
   VpeGetDevPaperBinName(hDoc, i, s, sizeof(s))
   MessageBox(s)   // show each available bin in a separate message box
next i
```

5.28 VpeGetDevPaperBinID

[Windows platform only; not supported by the Community Edition]

With this method you can retrieve all available paper bins of the currently selected output device. It returns the Bin-ID (see: VpeSetDevPaperBin) specified by "index" of the currently selected output device. The Bin-ID's are numeric Windows system constants to identify a bin. This method only works correctly if you called VpeDevEnumPaperBins() before.

int VpeGetDevPaperBinID(
VpeHandle *hDoc*,
int *index*
)

VpeHandle hDoc
Document Handle

int index
must be in the range between 0 and the value returned by VpeDevEnumPaperBins() - 1

Returns:
the Bin-ID, possible values are:

Constant Name	Value	Comment
VBIN_UNTOUCHED	0	
VBIN_UPPER	1	
VBIN_ONLYONE	1	
VBIN_LOWER	2	
VBIN_MIDDLE	3	
VBIN_MANUAL	4	
VBIN_ENVELOPE	5	
VBIN_ENVMANUAL	6	
VBIN_AUTO	7	
VBIN_TRACTOR	8	
VBIN_SMALLFMT	9	
VBIN_LARGEFMT	10	
VBIN_LARGECAPACITY	11	
VBIN_CASSETTE	14	

Additionally, each printer driver may define its own constants for other bins. Not all of the bin options are available on every printer. Check the printer's documentation for more specific descriptions of these options.

Remarks:
In case of an error, LastError is set to VERR_COMMON. This method works only correctly, if you called DevEnumPaperBins() before.

Example:
see VpeSetDevPaperBin()

5.29 VpeSetDevPaperBin

[Windows platform only; not supported by the Community Edition]

Selects a paper bin for the currently selected output device.

int VpeSetDevPaperBin(
    VpeHandle *hDoc*,
    int *bin_id*
)

VpeHandle hDoc
    Document Handle

int bin_id
    possible values are:

Constant Name	Value	Comment
VBIN_UNTOUCHED	0	
VBIN_UPPER	1	
VBIN_ONLYONE	1	
VBIN_LOWER	2	
VBIN_MIDDLE	3	
VBIN_MANUAL	4	
VBIN_ENVELOPE	5	
VBIN_ENVMANUAL	6	
VBIN_AUTO	7	
VBIN_TRACTOR	8	
VBIN_SMALLFMT	9	
VBIN_LARGEFMT	10	
VBIN_LARGECAPACITY	11	
VBIN_CASSETTE	14	

Additionally, each printer driver may define its own constants for other bins (see Remarks).

Not all of the bin options are available on every printer. Check the printer's documentation for more specific descriptions of these options.

Returns:

Value	Description
True	success
False	failure

Remarks:
In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support multiple paper bins.

The Bin-ID's are numeric Windows system constants to identify a bin. Additionally each printer driver may define its own constants for other bins. To retrieve a Bin-ID defined by a printer driver, enumerate all bins with VpeDevEnumPaperBins() and retrieve each Bin-ID associated with a Bin-Entry with VpeGetDevPaperBinID().

The DevPaperBin property is available for total control. It is only useful if called while processing the VPE_PRINT_NEWPAGE event. Otherwise the DevPaperBin is always overridden by the PaperBin property. **You should always use VpeSetPaperBin() to change the bin for the printer.**

It has been reported that several printer drivers - even from respectable manufacturers - do not behave correctly. Our own tests revealed for example that the HP 2200 D driver can only switch once the bin, but thereafter it will not switch the bin again. The HP 5P driver behaves correctly unless multiple copies and collation are selected. If collation is activated, the driver will print all pages except the first multiple times as if non-collation was selected.

Example:

```
char s[256]
count = VpeDevEnumPaperBins(hDoc) - 1   // initialize enumeration
for i = 0 to count
  VpeGetDevPaperBinName(hDoc, i, s, sizeof(s))
  // show each available device in a separate message box
  MessageBox(s)

  // select the bin
  VpeSetDevPaperBin(hDoc, VpeGetDevPaperBinID(hDoc, i))
next i
```

or, to select a bin directly:

```
VpeSetDevPaperBin(hDoc, VBIN_LOWER)
```

5.30 VpeGetDevPaperBin

[Windows platform only; not supported by the Community Edition]

Retrieves the current selected paper bin of the currently selected output device.

int VpeGetDevPaperBin(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Returns:
the Bin-ID, possible values are:

Constant Name	Value	Comment
VBIN_UNTOUCHED	0	
VBIN_UPPER	1	
VBIN_ONLYONE	1	
VBIN_LOWER	2	
VBIN_MIDDLE	3	
VBIN_MANUAL	4	
VBIN_ENVELOPE	5	
VBIN_ENVMANUAL	6	
VBIN_AUTO	7	
VBIN_TRACTOR	8	
VBIN_SMALLFMT	9	
VBIN_LARGEFMT	10	
VBIN_LARGECAPACITY	11	
VBIN_CASSETTE	14	
False	0	failure

Additionally, each printer driver may define its own constants for other bins (see Remarks).

Remarks:
In case of an error, LastError is set to VERR_COMMON.

The Bin-ID's are numeric Windows system constants to identify a bin. Additionally each printer driver may define its own constants for other bins. To retrieve a Bin-ID defined by a printer driver, enumerate all bins with VpeDevEnumPaperBins() and retrieve each Bin-ID associated with a Bin-Entry with VpeGetDevPaperBinID().

5.31 VpeGetDevPrinterOffsetX

[Windows platform only; not supported by the Community Edition]

Returns the leftmost x-position where the printer can print on for the currently selected output device. Printers have an unprintable area on the paper. This is the area around the paper margins where a printer can not print on due to technical limitations of the printing mechanism.

```
VpeCoord VpeGetDevPrinterOffsetX(
   VpeHandle hDoc
)
```

VpeHandle hDoc
   Document Handle

Returns:
   the leftmost x-position where the printer starts printing

Remarks:
   In case of an error, LastError is set to VERR_COMMON.

5.32 VpeGetDevPrinterOffsetY

[Windows platform only; not supported by the Community Edition]

Returns the topmost y-position where the printer can print on for the currently selected output device. Printers have an unprintable area on the paper. This is the area around the paper margins where a printer can not print on due to technical limitations of the printing mechanism.

VpeCoord VpeGetDevPrinterOffsetY(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  the topmost y-position where the printer starts printing

Remarks:
  In case of an error, LastError is set to VERR_COMMON.

5.33 VpeGetDevPrintableWidth

[Windows platform only; not supported by the Community Edition]

Returns the printable width for the currently selected output device. Printers have an unprintable area on the paper. This is the area around the paper margins where a printer can not print on due to technical limitations of the printing mechanism. The value returned by this function considers the unprintable area of the left and right margin and returns the resulting printable width.

VpeCoord VpeGetDevPrintableWidth(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  the printable width

Remarks:
  In case of an error, LastError is set to VERR_COMMON.

5.34 VpeGetDevPrintableHeight

[Windows platform only; not supported by the Community Edition]

Returns the printable height for the currently selected output device. Printers have an unprintable area on the paper. This is the area around the paper margins where a printer can not print on due to technical limitations of the printing mechanism.
The value returned by this function considers the unprintable area of the top and bottom margin and returns the resulting printable height.

VpeCoord VpeGetDevPrintableHeight(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the printable height

Remarks:
    In case of an error, LastError is set to VERR_COMMON.

5.35 VpeGetDevPhysPageWidth

[Windows platform only; not supported by the Community Edition]

Retrieves the total page width of the currently selected output device.

VpeCoord VpeGetDevPhysPageWidth(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the total page width of the currently selected output device

Remarks:
    In case of an error, LastError is set to VERR_COMMON.

5.36 VpeGetDevPhysPageHeight

[Windows platform only; not supported by the Community Edition]

Retrieves the total page height of the currently selected output device.

VpeCoord VpeGetDevPhysPageHeight(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the total page height of the currently selected output device

Remarks:
    In case of an error, LastError is set to VERR_COMMON.

5.37 VpeSetDevCopies

[Windows platform only; not supported by the Community Edition]

Sets the number of copies for the currently selected output device.

void VpeSetDevCopies(
VpeHandle *hDoc*,
int *copies*
)

VpeHandle hDoc
Document Handle

int copies
the number of copies to print

Remarks:
It is possible to set the default number of copies for most printers in the printer control panel. This value is automatically reflected by *DevCopies* at the moment a device is selected (i.e. when calling VpeOpenDoc() or VpeSetDevice()).

5.38 VpeGetDevCopies

[Windows platform only; not supported by the Community Edition]

Retrieves the selected number of copies for the currently selected output device.

int VpeGetDevCopies(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the selected number of copies for the currently selected output device

Remarks:
    It is possible to set the default number of copies for most printers in the printer control panel. This value is automatically reflected by *DevCopies* at the moment a device is selected (i.e. when calling VpeOpenDoc() or VpeSetDevice()).

5.39 VpeSetDevCollate

[Windows platform only; not supported by the Community Edition]

Specifies, if printed copies shall be collated on the currently selected output device.

void VpeSetDevCollate(
  VpeHandle *hDoc*,
  int *collate*
)

VpeHandle hDoc
  Document Handle

int collate

Value	Description
True	collate
False	do not collate

5.40 VpeGetDevCollate

[Windows platform only; not supported by the Community Edition]

Retrieves the setting for collation on the currently selected output device.

int VpeGetDevCollate(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Returns:

Value	Description
True	collate
False	do not collate

5.41 VpeSetDevFromPage

[Windows platform only; not supported by the Community Edition]

Sets the starting page that shall be printed onto the currently selected output device.

void VpeSetDevFromPage(
  VpeHandle *hDoc*,
  int *from_page*
)

VpeHandle hDoc
  Document Handle

int from_page
  the starting page

Remarks:
  This property is not written to a VPE setup file.

5.42 VpeGetDevFromPage

[Windows platform only; not supported by the Community Edition]

Retrieves the starting page that shall be printed onto the currently selected output device.

int VpeGetDevFromPage(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the starting page

Remarks:
    This property is not written to a VPE setup file.

5.43 VpeSetDevToPage

[Windows platform only; not supported by the Community Edition]

Sets the ending page that shall be printed onto the currently selected output device.

void VpeSetDevToPage(
    VpeHandle *hDoc*,
    int *to_page*
)

VpeHandle hDoc
    Document Handle

int to_page
    the ending page

Remarks:
    This property is not written to a VPE setup file.

5.44 VpeGetDevToPage

[Windows platform only; not supported by the Community Edition]

Retrieves the ending page that shall be printed onto the currently selected output device.

int VpeGetDevToPage(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  the ending page

Remarks:
  This property is not written to a VPE setup file.

5.45 VpeSetDevToFile

[Windows platform only; not supported by the Community Edition]

Specifies for the currently selected output device if the document shall be printed into a file.

void VpeSetDevToFile(
VpeHandle *hDoc*,
int *to_file*
)

VpeHandle hDoc
Document Handle

int to_file

Value	Description
True	print to file
False	do not print to file

Remarks:
This property is not written to a VPE setup file.

5.46 VpeGetDevToFile

[Windows platform only; not supported by the Community Edition]

Retrieves the setting for the currently selected output device whether the document shall be printed into a file.

int VpeGetDevToFile(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Value	**Description**
True	print to file
False	do not print to file

Remarks:
    This property is not written to a VPE setup file.

5.47 VpeSetDevFileName

[Windows platform only; not supported by the Community Edition]

Sets for the currently selected output device the file name that is used, if the document is printed into a file (see: VpeSetDevToFile). If this property is not set or reset to NULL or empty (""), a dialog box will pop-up when the print is started, asking the user for the file name.

With some printer drivers, this dialog box will not pop up.

void VpeSetDevFileName(
    VpeHandle *hDoc*,
    LPCSTR *file_name*
)

VpeHandle hDoc
    Document Handle

LPCSTR file_name
    the (path- and) filename the document is printed to

Remarks:
    This property is not written to a VPE setup file.

5.48 VpeGetDevFileName

[Windows platform only; not supported by the Community Edition]

Retrieves for the currently selected output device the file name that is used, if the document is printed into a file.

void VpeGetDevFileName(
VpeHandle *hDoc*,
LPSTR *file_name*,
int *size*
)

VpeHandle hDoc
Document Handle

LPSTR file_name
receive string for the file name the document is printed to

int size
the maximum size of the receive string

Remarks:
This property is not written to a VPE setup file.

Example:

```
char s[256]
VpeGetDevFileName(hDoc, s, sizeof(s))
```

5.49 VpeSetDevJobName

[Windows platform only; not supported by the Community Edition]

Sets the job name for the currently selected output device, which will be shown in the printer spooler of the system. If no job name is set ("" or NULL), VPE will compose a job name from the name of the application calling VPE and the title of the document (or of the filename if VpeOpenDocFile() is used).

void VpeSetDevJobName(
    VpeHandle *hDoc*,
    LPCSTR *job_name*
)

VpeHandle hDoc
    Document Handle

LPCSTR job_name
    job name

Remarks:
    This property is not written to a VPE setup file.

5.50 VpeGetDevJobName

[Windows platform only; not supported by the Community Edition]

Retrieves the job name for the currently selected output device, which will be shown in the printer spooler of the system. If no job name is set ("" or NULL), VPE will compose a job name from the name of the application calling VPE and the title of the document.

void VpeGetDevJobName(
VpeHandle *hDoc*,
LPSTR *job_name*,
int *size*
)

VpeHandle hDoc
Document Handle

LPSTR job_name
receive string for the job name

int size
the maximum size of the receive string

Remarks:
This property is not written to a VPE setup file.

5.51 VpeDevSendData

[Windows platform only; not supported by the Community Edition]

This method enables your application to send printer dependent Escape-Sequences (control sequences and binary data) directly to the printer. Because the data you send with this method is strictly printer dependent, you must know to what printer model you are printing.

With this method it is possible, to select for example an **output paper bin** by code (or whatever other functionality is provided by the connected printer).

int VpeDevSendData(
VpeHandle *hDoc*,
LPCSTR *data*,
long *size*
)

VpeHandle hDoc
Document Handle

LPCSTR data
the string with the data

long size
the number of bytes (characters) in the parameter string "data"

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set to VERR_COMMON. An error may occur if the device does not support sending binary data.

You may call this function only while processing the event VPE_PRINT_DEVDATA.

If you call VpeDevSendData() while not processing this event, the method does nothing.

Example:

In C / C++ (while processing the VPE_PRINT_DEVDATA event)

```
strcpy(s, "\033&l3O"); // PCL command to change Paper orientation to Reverse
Landscape
VpeDevSendData(hdoc, s, 5);
return PRINT_ACTION_CHANGE;
```

In our example, we send the "Reverse Landscape" PCL command to the printer, which is 5 bytes in size ("\033&130", where "\033" is one character octal coded = 27 decimal).
The characters in the string are: "&" + the letter "l" + "3" + the letter "O"

5.52 VpeWritePrinterSetup

[Windows platform only; not supported by the Community Edition]

Writes the current printer setup to file for persistent storage. This file is 100% identical to the file that is generated by VpeSetupPrinter().

int VpeWritePrinterSetup(
    VpeHandle *hDoc*,
    LPCSTR *file_name*
)

VpeHandle hDoc
    Document Handle

LPCSTR file_name
    the name of the file, the setup is written to

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set to VERR_COMMON.

For standardization, Printer Setup Files created with VPE should have the suffix ".PRS"

Win16 and Win32 printer setup files generated by VPE are NOT identically and therefore can NOT be created on one platform and be used on another platform. VPE detects such a situation and will set LastError = VERR_COMMON.

The setup files contain private device driver data. This has the big advantage, that special settings, options and features of a specific printer can be set and stored to file (for example some printers offer to select an output paper bin, which is not known by the Win API). Therefore, the use of the method VpeSetupPrinter() has some advantages, as the settings a user may make in the printer specific setup dialogs are stored with the file.

But - because of the private driver data - if the user replaces the printer by another model (or manufacturer) or possibly if just the driver version is changed, it might be necessary to delete the setup file(s) and to newly create them.

5.53 VpeReadPrinterSetup

[Windows platform only; not supported by the Community Edition]

Reads a printer setup file that was previously written by VpeSetupPrinter() or VpeWritePrinterSetup() and uses the settings contained in the file.

int VpeReadPrinterSetup(
    VpeHandle *hDoc*,
    LPCSTR *file_name*
)

VpeHandle hDoc
    Document Handle

LPCSTR file_name
    the name of the file, the setup is read from

Returns:

Value	Description
True	success
False	failure

Remarks:
    In case of an error, LastError is set to VERR_COMMON.

6 Layout Functions

6.1 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: "Dynamic Positioning" in the Programmer's Manual.

6.2 VpeSetUnitTransformation

Specifies the coordinate system in which the API of VPE handles coordinates. The coordinate system of VPE can either be in centimeter or inch units, als well as the old (prior to v4.0) 0.1mm units.

void VpeSetUnitTransformation(
    VpeHandle *hDoc,*
    double *factor*
)

VpeHandle hDoc
    Document Handle

double factor

Constant	Value	Description
VUNIT_FACTOR_CM	100000	coordinates are handled in cm units by the VPE API
VUNIT_FACTOR_INCH	254000	coordinates are handled in inch units by the VPE API
VUNIT_FACTOR_MM10	1000	coordinates are handled in 1/10 of a mm by the VPE API (compatible to VPE < v4.0)

Remarks:
    Setting this property does not affect the ruler’s unit measurement of the preview. To change the units of the rulers, use VpeSetRulersMeasure().

6.3 VpeGetUnitTransformation

Returns the coordinate system in which the API of VPE handles coordinates. The coordinate system of VPE can either be in centimeter or inch units, als well as the old (prior to v4.0) 0.1mm units.

double VpeGetUnitTransformation(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    The unit transformation factor:

Constant	Value	Description
VUNIT_FACTOR_CM	100000	coordinates are handled in cm units by the VPE API
VUNIT_FACTOR_INCH	254000	coordinates are handled in inch units by the VPE API
VUNIT_FACTOR_MM10	1000	coordinates are handled in 1/10 of a mm by the VPE API (compatible to VPE < v4.0)

6.4 VpeInternalToExternalUnit

Internally, VPE computes with a precision of 1/10.000 mm. VPE has a few API functions which return values in the internal coordinate system. You can use this function to convert such a value into the current unit transformation.

VpeCoord VpeInternalToExternalUnit(
  VpeHandle *hDoc,*
  int *value*
)

VpeHandle hDoc
  Document Handle

int value
  the internal value you wish to convert

Returns:
  the converted value from external to internal coordinates

6.5 VpeExternalToInternalUnit

Internally, VPE computes with a precision of 1/10.000 mm. You can use this function to convert a value of the current unit transformation into an internal value.

int VpeExternalToInternalUnit(
  VpeHandle *hDoc,*
  VpeCoord *value*
)

VpeHandle hDoc
  Document Handle

VpeCoord value
  the external value you wish to convert

Returns:
  the converted value from external to internal coordinates

6.6 VpeSetEngineRenderMode

[Windows platform only]

VPE v4.0 introduced a new platform independent rendering engine, which works identical on Windows as well as on Mac OS X, Linux, Solaris and all other platforms. Due to the new rendering engine, text strings are computed a bit wider and a bit less in height than in v3.x. So in very rare cases it can happen that word breaks occur on different positions than in v3.x, which means that text objects might require more width than before, but also less height. For this reason we implemented a property, so you can switch back to the original rendering engine. This can be achieved by setting EngineRenderMode = VENGINE_RENDER_MODE_VER3. This is useful to adapt your existing code to this new version of VPE. But the old rendering engine is only available on Windows. **Therefore, for platform independence, it is strongly recommended to use the new rendering engine.** It is also likely that the old rendering engine will be removed in a future version of VPE (in several years).

void VpeSetEngineRenderMode(
VpeHandle *hDoc,*
int *mode*
)

VpeHandle hDoc
Document Handle

int mode
the renderer mode

Constant	Value	Description
VENGINE_RENDER_MODE_VER3	0	VPE Renders Text compatible to Version 3.xx
VENGINE_RENDER_MODE_VER4	1	VPE Renders Text compatible to Version 4.00 and higher

Remarks:
Reading an old v3.xx VPE document will turn VPE into VER3 Mode for the whole document!

6.7 VpeGetEngineRenderMode

[Windows platform only]

Returns the current engine renderer mode.

int VpeGetEngineRenderMode(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the renderer mode

Constant	**Value**	**Description**
VENGINE_RENDER_MODE_VER3	0	VPE Renders Text compatible to Version 3.xx
VENGINE_RENDER_MODE_VER4	1	VPE Renders Text compatible to Version 4.00 and higher

6.8 VpePageBreak

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.

int VpePageBreak(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Returns:

Value	Description
True	success
False	failure

Remarks:
In case of an error, LastError is set. An error can occur, if the document is SwapFile based - i.e. you have used VpeOpenDocFile() - and there was an error writing to the SwapFile, because for example the harddisk is full.

6.9 VpeSetAutoBreak

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

For details see "Page Margins" and "Automatic Text Break" in the Programmer's Manual

void VpeSetAutoBreak(
VpeHandle *hDoc*,
int *mode*
)

VpeHandle hDoc
Document Handle

int mode

Constant Name	Comment
AUTO_BREAK_ON	Auto Break is activated. An Auto Break will happen if y2 = VFREE or y > VBOTTOMMARGIN. Remaining text is broken onto the next page(s) with the following coordinates: x = the original x coordinate of the inserted object x2 = the original x2 coordinate of the inserted object y = top of the Output Rectangle of the successive page - if a new page is generated, it will be the top of the Default Output Rectangle y2 = VFREE
AUTO_BREAK_OFF	Same behavior 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 instead.
AUTO_BREAK_NO_LIMITS	Remaining text is NOT broken onto the next page(s), it can be placed anywhere on the paper with no limits.
AUTO_BREAK_FULL	Auto Break is activated. An Auto Break will happen if y2 = VFREE or y > VBOTTOMMARGIN. Remaining text is broken onto the next page(s) with the following coordinates: x = left margin of the Output Rectangle x2= right margin of the Output Rectangle y = top margin of the Output Rectangle y2= VFREE Note: if a new blank page is added by VPE, the Default Output Rectangle will be used to set x, x2 and y. Otherwise if the next page is already existing, the Output Rectangle of the existing page is used. You can modify the Output Rectangle of an existing page at any time.

Default:
AUTO_BREAK_ON

6.10 VpeGetAutoBreak

Returns the current AutoBreakMode.

int VpeGetAutoBreak(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  the current AutoBreakMode

6.11 VpeGetPageCount

Retrieve the numbers of pages of the document

int VpeGetPageCount(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    The numbers of pages in the document

6.12 VpeGetCurrentPage

Retrieve the number of the current active page. This is the page where you can insert objects. It is independently from the Visual Page shown in the preview (see "Multipage Documents" in the Programmer's Manual and VpeGotoVisualPage().

int VpeGetCurrentPage(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    The number of the current active page.

Example:

```
VpeGotoPage(hDoc, 5)        // moves to page 5
n = VpeGetCurrentPage(hDoc) // returns the value 5
```

6.13 VpeGotoPage

Moves to the specified page; all further output-calls will insert objects onto this page. This page is independently from the Visual Page shown in the preview (see "Multipage Documents" in the Programmer's Manual and VpeGotoVisualPage()).

You can move at any time to any page to insert objects.

int VpeGotoPage(
  VpeHandle *hDoc*,
  int *page*
)

VpeHandle hDoc
  Document Handle

int page
  page number

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set.

If you want to show the preview and let the user work with it (e.g. scroll and print), while your application is still generating the document, see "Generating a Document while the Preview is open" in the Programmer's Manual for details.

Example:

```
VpeGotoPage(hDoc, 5)     // moves to page 5
VpePrint(hDoc, 1, 1, "This is page 5")
VpeGotoPage(hDoc, 2)     // moves to page 2
VpePrint(hDoc, 1, 1, "This is page 2")
```

6.14 VpeSetPageFormat

Sets the page format for the **current page**. You can set the page format for each page separately. All newly with PageBreak() added pages will have the page format which was specified at last.

void VpeSetPageFormat(
VpeHandle *hDoc*,
int *page_format*
)

VpeHandle hDoc
Document Handle

int page_format
one of the constants below

Default:
VPAPER_A4

Example:
VPE instructs the printer during printing, to switch the paper dimensions automatically according to the page dimensions of the VPE Document's currently printed page. Many printers can react on this and chose automatically the right paper from a different paper input bin. If you want to stop VPE from this automatism, you can specify the flag PRINT_NO_AUTO_PAGE_DIMS for the property *PrintOptions*.

Since the PageFormat and the Orientation of the document are independent from the settings of the printer, you can copy the settings of the printer to the document by using one of the following code samples:

Copying the printer's page format to the document from the default printer:

```
hDoc = VpeOpenDoc(hwnd, "Test", 0);
// the default printer is chosen automatically, so just copy the dimensions
VpeSetPageWidth(hDoc, VpeGetDevPaperWidth(hDoc));
VpeSetPageHeight(hDoc, VpeGetDevPaperHeight(hDoc));
VpeSetPageOrientation(hDoc, VpeGetDevOrientation(hDoc));
```

Copying the printer's page format to the document from a specific printer:

```
hDoc = VpeOpenDoc(hwnd, "Test", 0);
// choose "Office Printer 2"
VpeSetDevice(hDoc, "Office Printer 2");
VpeSetPageWidth(hDoc, VpeGetDevPaperWidth(hDoc));
VpeSetPageHeight(hDoc, VpeGetDevPaperHeight(hDoc));
VpeSetPageOrientation(hDoc, VpeGetDevOrientation(hDoc));
```

Copying the printer's page format to the document from a PRS file:

```
hDoc = VpeOpenDoc(hwnd, "Test", 0);
// use PRINT_NO_AUTO_PAGE_DIMS, in order to be able to retrieve the values
// for the page dimensions from the PRS file:
VpeSetPrintOptions(hDoc, PRINT_ALL + PRINT_NO_AUTO_PAGE_DIMS);
VpeSetupPrinter(hDoc, "personal settings.prs", PRINTDLG_ONFAIL);
VpeSetPageWidth(hDoc, VpeGetDevPaperWidth(hDoc));
VpeSetPageHeight(hDoc, VpeGetDevPaperHeight(hDoc));
VpeSetPageOrientation(hDoc, VpeGetDevOrientation(hDoc));

// Switch PrintOptions back and let VPE control the printer's page
// dimensions, i.e. clear the flag PRINT_NO_AUTO_PAGE_DIMS
VpeSetPrintOptions(hDoc, PRINT_ALL);
```

You will notice that we are using *PageWidth* and *PageHeight* in the examples above. The reason is, that this circumvents a bug in many printer drivers, which return wrong values for DevPaperFormat in case a user defined format has been assigned to the printer. Because of this, the construction "VpeSetPageFormat(hDoc, VpeGetDevPaperFormat(hDoc))" **can not be used reliably**. Instead, use the code from the examples above.

page_format **can be one of the following values:**

Constant	Value	Comment
VPAPER_USER_DEFINED	0	User-Defined
VPAPER_A4	-1	A4 Sheet, 210- by 297-millimeters
VPAPER_LETTER	-2	US Letter, 8 1/2- by 11-inches
VPAPER_LEGAL	-3	Legal, 8 1/2- by 14-inches
VPAPER_CSHEET	-4	C Sheet, 17- by 22-inches
VPAPER_DSHEET	-5	D Sheet, 22- by 34-inches
VPAPER_ESHEET	-6	E Sheet, 34- by 44-inches
VPAPER_LETTERSMALL	-7	Letter Small, 8 1/2- by 11-inches
VPAPER_TABLOID	-8	Tabloid, 11- by 17-inches
VPAPER_LEDGER	-9	Ledger, 17- by 11-inches
VPAPER_STATEMENT	-10	Statement, 5 1/2- by 8 1/2-inches
VPAPER_EXECUTIVE	-11	Executive, 7 1/4- by 10 1/2-inches
VPAPER_A3	-12	A3 sheet, 297- by 420-millimeters
VPAPER_A4SMALL	-13	A4 small sheet, 210- by 297-millimeters
VPAPER_A5	-14	A5 sheet, 148- by 210-millimeters
VPAPER_B4	-15	B4 sheet, 250- by 354-millimeters
VPAPER_B5	-16	B5 sheet, 182- by 257-millimeter paper
VPAPER_FOLIO	-17	Folio, 8 1/2- by 13-inch paper
VPAPER_QUARTO	-18	Quarto, 215- by 275-millimeter paper
VPAPER_10X14	-19	10- by 14-inch sheet
VPAPER_11X17	-20	11- by 17-inch sheet
VPAPER_NOTE	-21	Note, 8 1/2- by 11-inches
VPAPER_ENV_9	-22	#9 Envelope, 3 7/8- by 8 7/8-inches
VPAPER_ENV_10	-23	#10 Envelope, 4 1/8- by 9 1/2-inches
VPAPER_ENV_11	-24	#11 Envelope, 4 1/2- by 10 3/8-inches
VPAPER_ENV_12	-25	#12 Envelope, 4 3/4- by 11-inches
VPAPER_ENV_14	-26	#14 Envelope, 5- by 11 1/2-inches
VPAPER_ENV_DL	-27	DL Envelope, 110- by 220-millimeters
VPAPER_ENV_C5	-28	C5 Envelope, 162- by 229-millimeters

VPAPER_ENV_C3	-29	C3 Envelope, 324- by 458-millimeters
VPAPER_ENV_C4	-30	C4 Envelope, 229- by 324-millimeters
VPAPER_ENV_C6	-31	C6 Envelope, 114- by 162-millimeters
VPAPER_ENV_C65	-32	C65 Envelope, 114- by 229-millimeters
VPAPER_ENV_B4	-33	B4 Envelope, 250- by 353-millimeters
VPAPER_ENV_B5	-34	B5 Envelope, 176- by 250-millimeters
VPAPER_ENV_B6	-35	B6 Envelope, 176- by 125-millimeters
VPAPER_ENV_ITALY	-36	Italy Envelope, 110- by 230-millimeters
VPAPER_ENV_MONARCH	-37	Monarch Envelope, 3 7/8- by 7 ½-inches
VPAPER_ENV_PERSONAL	-38	6 3/4 Envelope, 3 5/8- by 6 1/2-inches
VPAPER_FANFOLD_US	-39	US Std Fanfold, 14 7/8- by 11-inches
VPAPER_FANFOLD_STD_GERMAN	-40	German Std Fanfold, 8 1/2- by 12-inches
VPAPER_FANFOLD_LGL_GERMAN	-41	German Legal Fanfold, 8 1/2- by 13-inches
VPAPER_ISO_B4	-42	B4 (ISO) 250 x 353 mm
VPAPER_JAPANESE_POSTCARD	-43	Japanese Postcard 100 x 148 mm
VPAPER_9X11	-44	9 x 11 in
VPAPER_10X11	-45	10 x 11 in
VPAPER_15X11	-46	15 x 11 in
VPAPER_ENV_INVITE	-47	Envelope Invite 220 x 220 mm
VPAPER_RESERVED_48	-48	RESERVED--DO NOT USE
VPAPER_RESERVED_49	-49	RESERVED--DO NOT USE
VPAPER_LETTER_EXTRA	-50	Letter Extra 9 \275 x 12 in
VPAPER_LEGAL_EXTRA	-51	Legal Extra 9 \275 x 15 in
VPAPER_TABLOID_EXTRA	-52	Tabloid Extra 11.69 x 18 in
VPAPER_A4_EXTRA	-53	A4 Extra 9.27 x 12.69 in
VPAPER_LETTER_TRANSVERSE	-54	Letter Transverse 8 \275 x 11 in
VPAPER_A4_TRANSVERSE	-55	A4 Transverse 210 x 297 mm
VPAPER_LETTER_EXTRA_TRANSVERSE	-56	Letter Extra Transverse 9\275 x 12 in
VPAPER_A_PLUS	-57	SuperA/SuperA/A4 227 x 356 mm
VPAPER_B_PLUS	-58	SuperB/SuperB/A3 305 x 487 mm
VPAPER_LETTER_PLUS	-59	Letter Plus 8.5 x 12.69 in
VPAPER_A4_PLUS	-60	A4 Plus 210 x 330 mm
VPAPER_A5_TRANSVERSE	-61	A5 Transverse 148 x 210 mm
VPAPER_B5_TRANSVERSE	-62	B5 (JIS) Transverse 182 x 257 mm
VPAPER_A3_EXTRA	-63	A3 Extra 322 x 445 mm
VPAPER_A5_EXTRA	-64	A5 Extra 174 x 235 mm
VPAPER_B5_EXTRA	-65	B5 (ISO) Extra 201 x 276 mm
VPAPER_A2	-66	A2 420 x 594 mm
VPAPER_A3_TRANSVERSE	-67	A3 Transverse 297 x 420 mm
VPAPER_A3_EXTRA_TRANSVERSE	-68	A3 Extra Transverse 322 x 445 mm
Windows 2000 or Higher:		
VPAPER_DBL_JAPANESE_POSTCARD	-69	Japanese Double Postcard 200 x 148 mm
VPAPER_A6	-70	A6 105 x 148 mm
VPAPER_JENV_KAKU2	-71	Japanese Envelope Kaku #2
VPAPER_JENV_KAKU3	-72	Japanese Envelope Kaku #3
VPAPER_JENV_CHOU3	-73	Japanese Envelope Chou #3
VPAPER_JENV_CHOU4	-74	Japanese Envelope Chou #4
VPAPER_LETTER_ROTATED	-75	Letter Rotated 11 x 8 1/2 in

VPAPER_A3_ROTATED	-76	A3 Rotated 420 x 297 mm
VPAPER_A4_ROTATED	-77	A4 Rotated 297 x 210 mm
VPAPER_A5_ROTATED	-78	A5 Rotated 210 x 148 mm
VPAPER_B4_JIS_ROTATED	-79	B4 (JIS) Rotated 364 x 257 mm
VPAPER_B5_JIS_ROTATED	-80	B5 (JIS) Rotated 257 x 182 mm
VPAPER_JAPANESE_POSTCARD_ROTATED	-81	Japanese Postcard Rotated 148 x 100 mm
VPAPER_DBL_JAPANESE_POSTCARD_ROTATED	-82	Double Japanese Postcard Rotated 148 x 200mm
VPAPER_A6_ROTATED	-83	A6 Rotated 148 x 105 mm
VPAPER_JENV_KAKU2_ROTATED	-84	Japanese Envelope Kaku #2 Rotated
VPAPER_JENV_KAKU3_ROTATED -85	-85	Japanese Envelope Kaku #3 Rotated
VPAPER_JENV_CHOU3_ROTATED	-86	Japanese Envelope Chou #3 Rotated
VPAPER_JENV_CHOU4_ROTATED	-87	Japanese Envelope Chou #4 Rotated
VPAPER_B6_JIS	-88	B6 (JIS) 128 x 182 mm
VPAPER_B6_JIS_ROTATED	-89	B6 (JIS) Rotated 182 x 128 mm
VPAPER_12X11	-90	12 x 11 in
VPAPER_JENV_YOU4	-91	Japanese Envelope You #4
VPAPER_JENV_YOU4_ROTATED	-92	Japanese Envelope You #4 Rotated
VPAPER_P16K	-93	PRC 16K 146 x 215 mm
VPAPER_P32K	-94	PRC 32K 97 x 151 mm
VPAPER_P32KBIG	-95	PRC 32K(Big) 97 x 151 mm
VPAPER_PENV_1	-96	PRC Envelope #1 102 x 165 mm
VPAPER_PENV_2	-97	PRC Envelope #2 102 x 176 mm
VPAPER_PENV_3	-98	PRC Envelope #3 125 x 176 mm
VPAPER_PENV_4	-99	PRC Envelope #4 110 x 208 mm
VPAPER_PENV_5	-100	PRC Envelope #5 110 x 220 mm
VPAPER_PENV_6	-101	PRC Envelope #6 120 x 230 mm
VPAPER_PENV_7	-102	PRC Envelope #7 160 x 230 mm
VPAPER_PENV_8	-103	PRC Envelope #8 120 x 309 mm
VPAPER_PENV_9	-104	PRC Envelope #9 229 x 324 mm
VPAPER_PENV_10	-105	PRC Envelope #10 324 x 458 mm
VPAPER_P16K_ROTATED	-106	PRC 16K Rotated
VPAPER_P32K_ROTATED	-107	PRC 32K Rotated
VPAPER_P32KBIG_ROTATED	-108	PRC 32K(Big) Rotated
VPAPER_PENV_1_ROTATED	-109	PRC Envelope #1 Rotated 165 x 102 mm
VPAPER_PENV_2_ROTATED	-110	PRC Envelope #2 Rotated 176 x 102 mm
VPAPER_PENV_3_ROTATED	-111	PRC Envelope #3 Rotated 176 x 125 mm
VPAPER_PENV_4_ROTATED	-112	PRC Envelope #4 Rotated 208 x 110 mm
VPAPER_PENV_5_ROTATED	-113	PRC Envelope #5 Rotated 220 x 110 mm
VPAPER_PENV_6_ROTATED	-114	PRC Envelope #6 Rotated 230 x 120 mm
VPAPER_PENV_7_ROTATED	-115	PRC Envelope #7 Rotated 230 x 160 mm
VPAPER_PENV_8_ROTATED	-116	PRC Envelope #8 Rotated 309 x 120 mm
VPAPER_PENV_9_ROTATED	-117	PRC Envelope #9 Rotated 324 x 229 mm
VPAPER_PENV_10_ROTATED	-118	PRC Envelope #10 Rotated 458 x 324 mm

6.15 VpeSetPageWidth

Sets the page width for the **current page** to a user defined value. Also all newly with PageBreak() added pages will have this width.

void VpeSetPageWidth(
VpeHandle *hDoc*,
VpeCoord *page_width*
)

VpeHandle hDoc
Document Handle

VpeCoord page_width
the width of the current page

Default:
21 (21cm = width of VPAPER_A4)

Remarks:
see VpeSetPageFormat()

VPE instructs the printer during printing, to switch the paper dimensions automatically according to the page dimensions of the VPE Document's currently printed page. Many printers can react on this and chose automatically the right paper from a different paper input bin. If you want to stop VPE from this automatism, you can specify the flag PRINT_NO_AUTO_PAGE_DIMS for the property *PrintOptions*.

In order to let the printer accept the automatic selection of user defined page formats (i.e. you are **not** specifying PRINT_NO_AUTO_PAGE_DIMS), it might be necessary to define a form of the desired page format.

Example: with "Start Menu | Settings | Printers" the window with the installed printers will appear. Right-click on a blank area of the window and choose "Server Properties" from the pop-up menu. In the upcoming dialog, define a custom form of the desired dimensions. For example name it "Test" and set the width to 760 and height to 1280.

In your source code, select the printer and the desired page format like this:

```
VpeSetDevice(hDoc, "Epson LQ-550");
VpeSetPageWidth(hDoc, 7.60);
VpeSetPageHeight(hDoc, 12.80);
```

That's it.

We experienced that this property doesn't work with some printer drivers. We tested this for example on an Epson LQ-550 dot-matrix printer and it didn't work on WfW 3.11 and NT 3.51, but it worked on Win95. But we heared about, that a LQ-510 printer driver will solve the problem for the LQ-550. So if your printer does not respond to this setting, it is a printer driver problem. In that case you can try to use another compatible printer driver.

The page size is freely definable up to 999cm x 999cm.

Win 9x / Me: on Win 9x/Me the GDI has 16-bit coordinates, so the dimensions of a page may not exceed 32.7 cm (12.9 inch) when using the version 4 renderer, and 138.7 cm (54.6 inch) when using the version 3 renderer. Those values apply to the preview. For other output devices, e.g. printers, the dimensions of a page may not exceed (32767 / DPI) * 2.54 cm, where DPI = the resolution of the output device.

Example: for a printer with 1200 DPI resolution, the maximum page dimensions may be (32767 / 1200) * 2.54 cm = 69.35 cm.

6.16 VpeGetPageWidth

Retrieves the page width of the **current page**.

VpeCoord VpeGetPageWidth(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the width of the current page

6.17 VpeSetPageHeight

Sets the page height for the **current page** to a user defined value. Also all newly with PageBreak() added pages will have this height.

void VpeSetPageHeight(
VpeHandle *hDoc*,
VpeCoord *page_height*
)

VpeHandle hDoc
Document Handle

VpeCoord page_height
the height of the current page

Default:
29.70 (29.7cm = height of VPAPER_A4)

Remarks:
see VpeSetPageFormat()
and VpeSetPageWidth

6.18 VpeGetPageHeight

Retrieves the page height of the **current page**.

VpeCoord VpeGetPageHeight(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the height of the current page

6.19 VpeSetPageOrientation

Sets the page orientation for the **current page**. Also all newly with PageBreak() added pages will have this orientation. The orientation is reflected in the preview if PaperView = True. Also the printer will print the page in the specified orientation.

void VpeSetPageOrientation(
VpeHandle *hDoc*,
int *orientation*
)

VpeHandle hDoc
Document Handle

int orientation
can be one of the following values:

Constant Name	Value	Comment
VORIENT_PORTRAIT	1	
VORIENT_LANDSCAPE	2	

Remarks:

Changing the orientation also modifies the OutRect of the current page and the DefOutRect (see also: "Page Margins" in the Programmer's Manual). The rectangles are rotated in a way that the margin spaces are kept the same. For example, if the OutRect and DefOutRect are set in a way, that all margins are 2cm from the paper borders, changing the orientation will keep those borders by rotating the rectangles.

This property is independent from DevOrientation. You should always use this function to specify the orientation.

Example:

```
VpeHandle hDoc = VpeOpenDoc(hwnd, "title", 0);
VpeSetupPrinter(hDoc, "personal settings.prs", PRINTDLG_ONFAIL);
VpeSetPageFormat(hDoc, VpeGetDevPaperFormat(hDoc));
VpeSetPageOrientation(hDoc, VpeGetDevOrientation(hDoc));
```

Changing the orientation during the print-job doesn’t work with some (buggy) printer drivers, for example some fax drivers and the HP4 M Plus driver (the HP4 PS driver should work with the HP4 M Plus printer!). Changing the orientation with such drivers on other pages than the very first page might not work. Most printer drivers will work.

Printer drivers are manufactured by vendors independent of IDEAL Software; we make no warranty, implied or otherwise, regarding these product’s performance or reliability.

Example:

```
VpeSetPageOrientation(hDoc, VORIENT_LANDSCAPE)
```

Sets the orientation for the current page to Landscape.

```
VpePageBreak(hDoc)
VpeSetPageOrientation(hDoc, VORIENT_PORTRAIT)
```

Adds a new page and sets the Orientation of the new added page to Portrait.

```
VpePageBreak(hDoc)
VpeSetPageOrientation(hDoc, VORIENT_LANDSCAPE)
```

Adds a third page and sets the Orientation of the new added page back to Landscape.

6.20 VpeGetPageOrientation

Retrieves the page orientation for the **current page**.

int VpeGetPageOrientation(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  one of the following values:

Constant Name	Value	Comment
VORIENT_PORTRAIT	1	
VORIENT_LANDSCAPE	2	

6.21 VpeSetPaperBin

[Not supported by the Community Edition]

Sets the paperbin for the **current page**. Also all newly with PageBreak() added pages will print to this bin.

void VpeSetPaperBin(
 VpeHandle *hDoc*,
 int *bin*
)

VpeHandle hDoc
 Document Handle

int bin
 can be one of the following values:

Constant Name	Value	Comment
VBIN_UNTOUCHED	-1	default
VBIN_UPPER	1	
VBIN_ONLYONE	1	
VBIN_LOWER	2	
VBIN_MIDDLE	3	
VBIN_MANUAL	4	
VBIN_ENVELOPE	5	
VBIN_ENVMANUAL	6	
VBIN_AUTO	7	
VBIN_TRACTOR	8	
VBIN_SMALLFMT	9	
VBIN_LARGEFMT	10	
VBIN_LARGECAPACITY	11	
VBIN_CASSETTE	14	

Not all of the bin options are available on every printer. Check the printer's documentation for more specific descriptions of these options. In addition the PaperBin-ID constants above can not be used reliably. Instead enumerate the available paper bins for the selected printer (please see below for details).

Default:
 VBIN_UNTOUCHED

Remarks:
 Setting this property on non-Windows platforms has no effect.

This property is independent from DevPaperBin. You should not use DevPaperBin, instead always use this property to specify the paper bin.

The value VBIN_UNTOUCHED is a VPE internal constant. It instructs VPE not to change the bin from the setting the current selected device has.

This property conflicts with the default behavior of VPE, to provide the page dimensions of the currently printed page to the printer driver. Many printer drivers choose automatically the right paper from a different paper input bin. To override the default behavior of VPE and to make the PaperBin property work, you have to specify the flag PRINT_NO_AUTO_PAGE_DIMS for the property *PrintOptions*.

Do not rely on the bin names of the constants, like VBIN_UPPER, etc. - their names might not match correctly a tray: for example under WinNT for an HP 5P you can select the lower tray with VBIN_LOWER and the upper tray with VBIN_UPPER, but under Win95 for the same printer you can only use VBIN_MANUAL for the upper tray and VBIN_UPPER for the lower tray (yes, it is not a mistake: the LOWER tray is selected with VBIN_UPPER!).
Solution: operate with BIN-ID's and the corresponding bin names only, see VpeDevEnumPaperBins() and ff.

Changing the bin during the print-job doesn't work with some (buggy) printer drivers. Changing the bin with such drivers for other pages than the very first page might not work. Our own tests revealed for example that the HP 2200 D driver can only switch once the bin, but thereafter it will not switch the bin again. The HP 5P driver behaves correctly unless multiple copies and collation are selected. If collation is activated, the driver will print all pages except the first multiple times as if non-collation was selected.

Printer drivers are manufactured by vendors independent of IDEAL Software; we make no warranty, implied or otherwise, regarding these product's performance or reliability.

Example:

```
VpeSetPaperBin(hDoc, VBIN_UPPER)
```

Will instruct the printer during printing, to print this page on the upper paper bin (if available).

```
VpePageBreak(hDoc)
VpeSetPaperBin(hDoc, VBIN_LOWER)
```

Adds a new page and will instruct the printer during printing, to print this page on the lower paper bin (if available).

```
VpePageBreak(hDoc)
VpeSetPaperBin(hDoc, VBIN_UPPER)
```

Adds a third page and will instruct the printer during printing, to print this page on the upper paper bin again (if available).

6.22 VpeGetPaperBin

Retrieves the Paperbin for the **current page**.

int VpeGetPaperBin(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    one of the following values:

Constant Name	Value	Comment
VBIN_UNTOUCHED	-1	default
VBIN_UPPER	1	
VBIN_ONLYONE	1	
VBIN_LOWER	2	
VBIN_MIDDLE	3	
VBIN_MANUAL	4	
VBIN_ENVELOPE	5	
VBIN_ENVMANUAL	6	
VBIN_AUTO	7	
VBIN_TRACTOR	8	
VBIN_SMALLFMT	9	
VBIN_LARGEFMT	10	
VBIN_LARGECAPACITY	11	
VBIN_CASSETTE	14	

6.23 VpeStoreSet

All current property settings (pen-size, alignment, colors, font, etc.) are stored in a buffer under the specified id. 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 want to switch back to the current settings later again.

void VpeStoreSet(
VpeHandle *hDoc*,
int *id*
)

VpeHandle hDoc
Document Handle

int id
the id under which you store the properties

Remarks:

The following properties are stored:

FontName
FontSize
PenSize
PenStyle
PenColor
Bold
Italic
Underline
StrikeOut
TextAlignment
TextColor
BkgColor
BkgMode
GradientStartColor
GradientEndColor
BkgGradientTriColor
BkgGradientMiddleColorPosition
BkgGradientMiddleColor
GradientRotation
HatchStyle
HatchColor
CornerRadius
AutoBreakMode
Rotation

PicturePage
PictureType
PictureKeepAspect
PictureCache
PictureScale2Gray
PictureScale2GrayFloat
PictureX2YResolution
PictureBestFit
PictureEmbedInDoc
PictureDrawExact

PictureDefaultDPIX
PictureDefaultDPIY

JpegExportOptions
TiffExportOptions
BmpExportOptions
PnmExportOptions
GifExportOptions
PictureExportColorDepth
PictureExportDither

Charset
CharPlacement
RTFParagraph - The complete Paragraph Settings
ChartProperties

BarcodeMainText
BarcodeAddText
BarcodeAlignment
BarcodeAutoChecksum
BarcodeThinBar
BarcodeThickBar

Viewable
Printable
Streamable
Shadowed

CharacterCount
SubdividerPenSize
SubdividerPenColor
AltSubdividerNPosition
AltSubdividerPenSize
AltSubdividerPenColor
BottomLinePenSize
BottomLinePenColor
SubdividerStyle
AltSubdividerStyle
EditFlags

BookmarkDestination
BookmarkStyle
BookmarkColor

Bar2DAlignment
DataMatrixEncodingFormat
DataMatrixEccType
DataMatrixRows
DataMatrixColumns
DataMatrixMirror
DataMatrixBorder
PDF417ErrorLevel
PDF417Columns
PDF417Rows
AztecFlags
AztecControl
AztecMenu

AztecMultipleSymbols
AztecID

Example:

```
VpeStoreSet(hDoc, 1)     // store the current settings
VpeSetFontSize(hDoc, 12) // modify the current properties,
VpeSetPenSize(hDoc, 6)   // and output some text
VpeWriteBox(hDoc, 1, 1, "Hello World!")
VpeUseSet(hDoc, 1)       // return to the original settings
VpeRemoveSet(hDoc, 1)    // delete the stored settings
```

6.24 VpeUseSet

This resets all properties to the stored values.

void VpeUseSet(
    VpeHandle *hDoc*,
    int *id*
)

VpeHandle hDoc
    Document Handle

int id
    the id under which you stored the properties

6.25 VpeRemoveSet

Removes the setting specified under ID from memory.

void VpeRemoveSet(
    VpeHandle *hDoc*,
    int *id*
)

VpeHandle hDoc
    Document Handle

int id
    the id under which you stored the properties

6.26 VpeGet

Returns the coordinate / value specified by one of the V-Flags provided in parameter "what". See "Dynamic Positioning" in the Programmer's Manual for details.

VpeCoord VpeGet(
VpeHandle *hDoc*,
VpeCoord *what*
)

VpeHandle hDoc
Document Handle

VpeCoord what
one of the following V-Flags:

Constant Name	**Value**
VLEFT	-2147483550
VRIGHT	-2147483551
VLEFTMARGIN	-2147483552
VRIGHTMARGIN	-2147483553
VTOP	-2147483554
VBOTTOM	-2147483555
VTOPMARGIN	-2147483556
VBOTTOMMARGIN	-2147483557
VWIDTH	-100
VHEIGHT	-101
VRENDERWIDTH	-102
VRENDERHEIGHT	-103
VUDO_LEFT	-104
VUDO_RIGHT	-105
VUDO_TOP	-106
VUDO_BOTTOM	-107
VUDO_WIDTH	-108
VUDO_HEIGHT	-109

Returns:
The coordinate / value specified by parameter "what"

Remarks:
In contrast to the other V-Flags, the VUDO_xyz Flags return the bounding rectangle of the UDO in **DEVICE COORDINATES** (!) (not in metric or inch units).

Note: the values for margins are specified in coordinates relative to the top / left paper border, e.g. if the right margin shall be 2cm away from the right paper border, set it to ‘page_width - 2’. If you would set the right margin = 2, it would be 2cm away from the left border.

For use with C/C++ there are macros defined in VPECOMON.H, which are easier to use:

```
#define nLeft(hDoc)         VpeGet(hDoc, VLEFT)
#define nTop(hDoc)          VpeGet(hDoc, VTOP)
#define nRight(hDoc)        VpeGet(hDoc, VRIGHT)
#define nBottom(hDoc)       VpeGet(hDoc, VBOTTOM)
#define nLeftMargin(hDoc)   VpeGet(hDoc, VLEFTMARGIN)
#define nTopMargin(hDoc)    VpeGet(hDoc, VTOPMARGIN)
#define nRightMargin(hDoc)  VpeGet(hDoc, VRIGHTMARGIN)
#define nBottomMargin(hDoc) VpeGet(hDoc, VBOTTOMMARGIN)
#define nWidth(hDoc)        VpeGet(hDoc, VWIDTH)
#define nHeight(hDoc)       VpeGet(hDoc, VHEIGHT)
#define nRenderWidth(hDoc)  VpeGet(hDoc, VRENDERWIDTH)
#define nRenderHeight(hDoc) VpeGet(hDoc, VRENDERHEIGHT)
#define nUDOLeft(hDoc)      VpeGet(hDoc, VUDO_LEFT)
#define nUDORight(hDoc)     VpeGet(hDoc, VUDO_RIGHT)
#define nUDOTop(hDoc)       VpeGet(hDoc, VUDO_TOP)
#define nUDOBottom(hDoc)    VpeGet(hDoc, VUDO_BOTTOM)
#define nUDOWidth(hDoc)     VpeGet(hDoc, VUDO_WIDTH)
#define nUDOHeight(hDoc)    VpeGet(hDoc, VUDO_HEIGHT)
```

Example:

```
VpeWrite(hDoc, 1, 2, 4, 3, "Hello")
```

After executing this method the properties contain the following values:
VLEFT = 1, VTOP = 2, VRIGHT = 4, VBOTTOM = 3

You can insert an object at the right border of the last inserted object with:

```
VpePrint(hDoc, VpeGet(hDoc, VRIGHT), VpeGet(hDoc, VTOP), " World!")
```

The equal result (but with faster processing) is created with the following statement:

```
VpePrint(hDoc, VRIGHT, VTOP, " World!")
```

If you want to insert the second object with an offset, you can only use:

```
VpePrint(hDoc, VpeGet(hDoc, VRIGHT) + 0.5, VpeGet(hDoc, VTOP), " World!")
```

which will insert the object with an offset of 0.5 cm to the right border of the previously inserted object.

The following is not possible:

```
VpePrint(hDoc, VRIGHT + 0.5, VTOP, " World!")
```

because VRIGHT is a constant (its value is -2147483551), which instructs VPE to retrieve the value of the property RIGHT internally. If you add 0.5, the result is -2147483550.5, which is interpreted as a standard coordinate.

6.27 VpeSet

Sets the coordinate / value specified by one of the V-Flags provided in parameter "what". See "Dynamic Positioning" in the Programmer's Manual for details.

void VpeSet(
    VpeHandle *hDoc*,
    VpeCoord *what*,
    VpeCoord *value*
)

VpeHandle hDoc
    Document Handle

VpeCoord what
    one of the following V-Flags:

Constant Name	Value
VLEFT	-2147483550
VRIGHT	-2147483551
VLEFTMARGIN	-2147483552
VRIGHTMARGIN	-2147483553
VTOP	-2147483554
VBOTTOM	-2147483555
VTOPMARGIN	-2147483556
VBOTTOMMARGIN	-2147483557
VRENDERWIDTH	-102
VRENDERHEIGHT	-103

VpeCoord value
    new coordinate / value for the V-Flag

See Also:
    VpeGet

6.28 VpeSetDefOutRect

Sets the **default** output rectangle which will be used for each newly created page - NOT for the current page.

The output rectangle is very important for use with the AutoPageBreak option of VPE for rendering text.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

"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.

void VpeSetDefOutRect(
 VpeHandle *hDoc*,
 VpeCoord *x*,
 VpeCoord *y*,
 VpeCoord *x2*,
 VpeCoord *y2*
)

VpeHandle hDoc
 Document Handle

VpeCoord x, y, x2, y2
 the output rectangle

Remarks:
The method VpeSetOutRect() calls internally SetDefOutRect() and sets the DefOutRect to the same rectangle. Therefore if you need to call VpeSetOutRect() and VpeSetDefOutRect() with different values at the same time, always call VpeSetOutRect() first.

Example:

```
VpeSetDefOutRectSP(hDoc, 3, 3, 19, 25)
```

6.29 VpeSetOutRect

Sets the **current** output rectangle for the currently active page. (see "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details)

This method has the same effect like setting each nMargin property separately (see VpeSet).

void VpeSetOutRect(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
the output rectangle

Remarks:
This function sets the DefOutRect to the same rectangle automatically.

6.30 VpeStorePos

Stores the coordinates nLeft, nTop, nRight, nBottom (x, y, x2, y2) of the last inserted object on a dynamic stack (limited in size only by available memory). The stack is a LIFO stack (Last In - First Out), which means that the last stored position is retrieved first (see VpeRestorePos).

void VpeStorePos(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

6.31 VpeRestorePos

Restores the last stored coordinates nLeft, nTop, nRight, nBottom (x, y, x2, y2) from the stack.

void VpeRestorePos(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

6.32 VpeSetRotation

[Not supported by the Community Edition]

Sets the new rotation angle for text, images and barcodes. Rotation is done clockwise. Angles are given in 1/10 degrees.

The x, y position will always be the same. It is not modified by rotation. 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 LEON3_32.DLL to rotate images.

For a detailed explanation see "Rotation of Text, Images and Barcodes" in the Programmer's Manual.

void VpeSetRotation(
  VpeHandle *hDoc*
  int *angle*
)

VpeHandle hDoc
  Document Handle or
  VPE Object Handle (works only for TVPEObjects which reside in a template!)

int angle
  the rotation angle in 0.1 degrees, clockwise

Default:
  0 degrees

Remarks:
  Rotation of images is only possible, if you are using the Enhanced Edition or above.

Example:

```
VpeSetRotation(hdoc, 900)
```

The next inserted objects will be rotated by 90 degrees clockwise.

6.33 VpeGetRotation

[Not supported by the Community Edition]

Returns the current rotation angle for text, images and barcodes.
Rotation is done clockwise in 1/10 degrees.

int VpeGetRotation(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    The current rotation angle.

6.34 VpeSetViewable

[Professional Edition and above]

Controls, whether the next inserted object(s) are visible in the preview.

void VpeSetViewable(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int yes_no

Value	Description
True	viewable
False	not viewable

Default:
True

6.35 VpeSetPrintable

[Professional Edition and above]

Controls, whether the next inserted object(s) are printed and exported to external files, i.e. to PDF or image files.

void VpeSetPrintable(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int yes_no

Value	Description
True	yes
False	no

Default:
True

Remarks:
By default, objects which are marked as non-printable are not exported to external file formats (for example to images or PDF). See the property ExportNonPrintableObjects to override this behavior.

6.36 VpeSetExportNonPrintableObjects

[Professional Edition and above]

By default, objects which are marked as non-printable are not exported to external file formats (for example to images or PDF). If this property is set to *True*, all objects of a document which are marked as non-printable are exported, too.

This property is document-wide in effect, it does not affect single objects.

void VpeSetExportNonPrintableObjects(
    VpeHandle *hDoc*,
    int *yes_no*
)

VpeHandle hDoc
    Document Handle

int yes_no
    possible values are:

Value	Description
True	1: yes, export objects which are marked as non-printable
False	0: no, do not export objects which are marked as non-printable

Default:
    False, do not export objects which are marked as non-printable

Remarks:
    This property does not affect VPE Document files (.VPE files). Objects which are marked as non-printable are always written to VPE Document files. The property is only relevant for export operations to external file formats, like PDF or image files.

    But the setting of this property is also written to VPE Document files, i.e. if you read a VPE Document file, the setting of this property is read from the file.
    Example: you set the property = true and write a document to the file "test.vpe". Later you open a new document, so by default this property is false, but after you call VpeReadDoc("test.vpe"), the property will be true, because it is read from "test.vpe".

    The setting of this property also affects how VPE will create PDF file attachments when sending e-mails. Furthermore it affects in the same way the document viewer *VPEView*.

    You can set the property temporarily = *True* before exporting a document, and immediately afterwards back to *False*. This assures that it is not written to a VPE Document file.

6.37 VpeGetExportNonPrintableObjects

[Professional Edition and above]

Returns the current setting of the property.

int VpeGetExportNonPrintableObjects(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

6.38 VpeSetStreamable

[Professional Edition and above]

Controls, whether the next inserted object(s) are streamed to the VPE Document File.

void VpeSetStreamable(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int yes_no

Value	Description
True	yes
False	no

Default:
True

6.39 VpeSetShadowed

[Professional Edition and above]

Draws a shadow automatically, valid for all rectangular objects.

void VpeSetShadowed(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int yes_no

Value	**Description**
True	yes, draw a shadow
False	no

Default:
True

6.40 VpeClearPage

[Enterprise Edition and above]

Deletes all VPE Objects of the current page. The current page will be blank and contains no objects after calling this function.

void VpeClearPage(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Remarks:
    The preview will refresh automatically after calling this function.

6.41 VpeInsertPage

[Professional Edition and above]

Inserts a new blank page at the position of the current page in the document.

The previously current page becomes the successor of the newly inserted page.

The new blank page becomes the current page.

method void VpeInsertPage(
)

Remarks:

Pages numbered with the @PAGE macro or the $(Page) field are **not renumbered** after calling this method. We recommend to use the "*<page> of <total pages>*" *technique* explained in the Programmer's Manual.

The preview will refresh automatically after calling this function.

Enterprise Edition and above:
As you insert by code the page numbers as text objects into the document, obtain the VPE Object handles of those text objects by using the property LastInsertedObject and store these object references in a list. If there is a requirement for renumbering the document, first delete all objects in this list and then start the numbering process again.

6.42 VpeRemovePage

[Professional Edition and above]

Removes the current page from the document.
If the current page has a succeeding page, the succeeding page will become the new current page.
If there is no succeeding page (i.e. you removed the last page of the document), the preceding page will become the new current page.
If the document has only one page, it can not be removed. Instead VPE will call internally VpeClearPage().

method void VpeRemovePage(
)

Remarks:

Pages numbered with the @PAGE macro or the $(Page) field are **not renumbered** after calling this method. We recommend to use the "*<page> of <total pages>*" *technique* explained in the Programmer's Manual.

The preview will refresh automatically after calling this function.

Enterprise Edition and above:
As you insert by code the page numbers as text objects into the document, obtain the VPE Object handles of those text objects by using the property LastInsertedObject and store these object references in a list. If there is a requirement for renumbering the document, first delete all objects in this list and then start the numbering process again.

6.43 VpeDeleteObject

[Enterprise Edition and above]

Deletes the given VPE Object from the document. References to VPE Objects can be obtained with the properties LastInsertedObject and FirstObject.

void VpeDeleteObject(
  VpeHandle *hDoc*,
  VpeHandle *hObject*
)

VpeHandle hDoc
  Document Handle

VpeHandle hObject
  VPE Object Handle

Remarks:

Use this method with caution: if the deleted object was inserted from a dumped template, and you try to access it with VpeGetInsertedVpeObject(), an Access Violation will occur.
As an alternative, you could make the object invisible instead of deleting it, by setting its properties *Streamable*, *Viewable* and *Printable* to false.

This function works only, if the VPE Document is not stream based, i.e. you did not open / create it using VpeOpenDocFile().
The preview will not refresh automatically after calling this function. Call VpeRefreshDoc() to do so.

6.44 VpeGetLastInsertedObject

[Enterprise Edition and above]

Returns the VPE Object handle of the last object that has been inserted into the VPE Document.

VpeHandle VpeGetLastInsertedObject(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Remarks:
If there is no object available, NULL is returned.

6.45 VpeGetFirstObject

[Enterprise Edition and above]

Returns the handle of the first VPE Object of the current page. Each VPE Object itself offers the property *NextObject* in order to iterate through all VPE Objects of a page.

VpeHandle VpeGetFirstObject(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Remarks:
  If there is no object available, NULL is returned.

7 Rendering

7.1 Rendering

These methods help to find out the size of text and images **without** inserting them into a document. The methods compute the size of text and pictures. The text or image is not inserted into the document.

Italic fonts are a bit higher than non-italic fonts. This is caused by the Windows System GDI. The consequence is, that italic text needs more height, which might result in clipped (not drawn) text, in case the height returned by a text-render method for a non-italic font is used for an italic font.

Remarks:

For framed objects (also FormFields with Bottom-Line / (Alt-)Subdivider) the center of the pen is used a sbasis for computations. For example if RenderWriteBox() returned as computed position / dimensions (1, 1, 2, 2) and the box of the text object has a PenSize (frame thickness) of 1cm, then the true surrounding rectangle of the box has the coordinates (0.5, 0.5, 2.5, 2.5).
In order to compute the true outer borders you need to inflate the rectangle by ½ of the PenSize.

See also: "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

Pictures:

Because parameters width and height can be of any value, you can for example compute the width of an undistorted image (see VpeSetPictureKeepAspect) at a given height. In most cases you will set width = VFREE and height = VFREE.

See also:

"Rendering Objects" in the Programmer's Manual

7.2 VpeRenderPrint

Computes the dimensions of a given text, based on the method VpePrint().

int VpeRenderPrint(
    VpeHandle *hDoc*,
    VpeCoord *x*,
    VpeCoord *y*,
    LPCSTR *s*
)

VpeHandle hDoc
    Document Handle

VpeCoord x, y
    position

LPCSTR s
    the text to be rendered

Returns:
    The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:
    The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

Example:

```
VpeRenderPrint(hDoc, 0, 0, "X")
font_height = VpeGet(hDoc, VRENDERHEIGHT)
```

retrieves the font height for a single line

See also:
    "Rendering Objects" in the Programmer's Manual

7.3 VpeRenderPrintBox

Computes the dimensions of a given text, based on the method VpePrintBox(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

int VpeRenderPrintBox(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
LPCSTR *s*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y
position

LPCSTR s
the text to be rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:
The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:
"Rendering Objects" in the Programmer's Manual

7.4 VpeRenderWrite

Computes the dimensions of a given text, based on the method VpeWrite().

int VpeRenderWrite(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *s*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions, x2 and / or y2 may be set to VFREE

LPCSTR s
the text to be rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:
The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:
"Rendering Objects" in the Programmer's Manual

7.5 VpeRenderWriteBox

Computes the dimensions of a given text, based on the method VpeWriteBox(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

int VpeRenderWriteBox(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *s*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions, x2 and / or y2 may be set to VFREE

LPCSTR s
the text to be rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:
The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:
"Rendering Objects" in the Programmer's Manual

7.6 VpeRenderPicture

Computes the dimensions of a given image, based on the method VpePicture(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

If the property PictureCache is true, the rendered image will automatically be stored in the image cache. We recommend to set PictureCache = true always.

void VpeRenderPicture(
  VpeHandle *hDoc*,
  VpeCoord *width*,
  VpeCoord *height*,
  LPCTSTR *file_name*
)

VpeHandle hDoc
  Document Handle

VpeCoord width, height
  Can be used to compute either the width or the height by setting one value to a numeric value whilst using VFREE for the other value. Usually you will set both values to VFREE.

LPCSTR file_name
  the image file to be rendered

Remarks:

In case of an error, LastError is set.

The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

Rendering a picture will move the picture to the image cache (if VpeSetPictureCache() has been set to true before).

Example:

```
VpeSetPictureCache(hDoc, TRUE);
VpeRenderPicture(hDoc, VFREE, VFREE, "image.bmp")
xsize = VpeGet(hDoc, VRENDERWIDTH)
ysize = VpeGet(hDoc, VRENDERHEIGHT)
```

Computes the width and height of the image stored in "image.bmp". By setting *PictureCache* to true, the image will be loaded into the cache, so if you insert the image later into a document, it isn’t loaded from file a second time (assuming, that it isn’t flushed from the cache, because you rendered / inserted many other - or huge - images that needed the cache). *PictureCache* is by default true when you open a document.

```
VpeSetPictureKeepAspect(hDoc, TRUE)
VpeRenderPicture(hDoc, VFREE, 5, "image.bmp")
xsize = VpeGet(hDoc, VRENDERWIDTH)
```

Computes the undistorted width of the image stored in "image.bmp" relative to the given height of 5cm.

See also:

"Rendering Objects" in the Programmer's Manual

7.7 VpeRenderPictureStream

[Professional Edition and above]

Identical to VpeRenderPicture(), but renders a picture from a memory stream.

void VpeRenderPictureStream(
  VpeHandle *hDoc*,
  VpeHandle *hStream*,
  VpeCoord *width*,
  VpeCoord *height*,
  LPCTSTR *identifier*
)

VpeHandle hDoc
  Document Handle

VpeHandle hStream
  The handle of the stream where the picture is read from. The stream must have been created before by calling VpeCreateMemoryStream(), and of course it must have been initialized with valid image data.

VpeCoord width, height
  Can be used to compute either the width or the height by setting one value to a numeric value whilst using VFREE for the other value. Usually you will set both values to VFREE.

LPCTSTR identifier
  A name for the picture. The internal image cache uses this name to distinguish images. But the image cache also computes a CRC (checksum) of the image data, so you can also leave the identifier blank (NOT NULL!). However, we do recommend to use a name if possible, so the CRC is not the only factor.

Remarks:
  If you wish to use one and the same image multiple times, always provide the same stream handle (and therefore of course the same stream) as well as the same identifier, when calling this method.

See also:
  "Rendering Objects" in the Programmer's Manual

7.8 VpeRenderPictureResID

[Windows platform only]

Computes the dimensions of a given image, based on the method VpePictureResID(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

void VpeRenderPictureResID(
VpeHandle *hDoc*,
VpeCoord *width*,
VpeCoord *height*,
int *hInstance*,
unsigned int *res_id*
)

VpeHandle hDoc
Document Handle

VpeCoord width, height
Can be used to compute either the width or the height by setting one value to a numeric value whilst using VFREE for the other value. Usually you will set both values to VFREE.

int hInstance
Instance-Handle

unsigned int res_id
the resource id

Remarks:

In case of an error, LastError is set.

The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

Example:

see VpeRenderPicture

See also:

"Rendering Objects" in the Programmer's Manual

7.9 VpeRenderPictureResName

[Windows platform only]

Computes the dimensions of a given image, based on the method VpePictureResName(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

void VpeRenderPictureResName(
VpeHandle *hDoc*,
VpeCoord *width*,
VpeCoord *height*,
int *hInstance*,
LPCSTR *res_name*
)

VpeHandle hDoc
Document Handle

VpeCoord width, height
Can be used to compute either the width or the height by setting one value to a numeric value whilst using VFREE for the other value. Usually you will set both values to VFREE.

int hInstance
Instance-Handle

LPCSTR res_name
the resource name

Remarks:
In case of an error, LastError is set.
The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

Example:
see VpeRenderPicture

See also:
"Rendering Objects" in the Programmer's Manual

7.10 VpeRenderPictureDIB

[Windows platform only]

Computes the dimensions of a given image, based on the method VpePictureDIB(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

void VpeRenderPictureDIB(
VpeHandle *hDoc*,
VpeCoord *width*,
VpeCoord *height*,
HGLOBAL *hDIB*
)

VpeHandle hDoc
Document Handle

VpeCoord width, height
Can be used to compute either the width or the height by setting one value to a numeric value whilst using VFREE for the other value. Usually you will set both values to VFREE.

HGLOBAL hDIB
handle of the DIB to be rendered

Remarks:

In case of an error, LastError is set.

The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

Example:

see VpeRenderPicture

See also:

"Rendering Objects" in the Programmer's Manual

7.11 VpeRenderRTF

[Professional Edition and above]

Computes the dimensions of a given RTF text, based on the method VpeWriteRTF().

int VpeRenderRTF(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*,
  LPCSTR *s*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y, x2, y2
  position and dimensions, y2 may be set to VFREE

LPCSTR s
  the RTF text to be rendered

Returns:

The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:

VFREE: only the y2 coordinate may be set to VFREE, not x2.
In case of an error, LastError is set.
The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:

"Rendering Objects" in the Programmer's Manual

7.12 VpeRenderBoxRTF

[Professional Edition and above]

Computes the dimensions of a given RTF text, based on the method VpeWriteBoxRTF(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

int VpeRenderBoxRTF(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *s*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions, y2 may be set to VFREE

LPCSTR s
the RTF text to be rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:
VFREE: only the y2 coordinate may be set to VFREE, not x2.
In case of an error, LastError is set.
The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:
"Rendering Objects" in the Programmer's Manual

7.13 VpeRenderRTFFile

[Professional Edition and above]

Computes the dimensions of a given RTF text file, based on the method VpeWriteRTFFile().

int VpeRenderRTFFile(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*,
  LPCSTR *file_name*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y, x2, y2
  position and dimensions, y2 may be set to VFREE

LPCSTR file_name
  the file with RTF text that is rendered

Returns:
  The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:
  VFREE: only the y2 coordinate may be set to VFREE, not x2.
  In case of an error, LastError is set.
  The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:
  "Rendering Objects" in the Programmer's Manual

7.14 VpeRenderBoxRTFFile

[Professional Edition and above]

Computes the dimensions of a given RTF text file, based on the method VpeWriteBoxRTFFile(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

int VpeRenderBoxRTFFile(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *file_name*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions, y2 may be set to VFREE

LPCSTR file_name
the file with RTF text that is rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:
VFREE: only the y2 coordinate may be set to VFREE, not x2.
In case of an error, LastError is set.
The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:
"Rendering Objects" in the Programmer's Manual

7.15 VpeRenderRTFStream

[Professional Edition and above]

Computes the dimensions of a given RTF stream, based on the method VpeWriteRTFFile().

int VpeRenderRTFStream(
    VpeHandle *hDoc*,
    VpeHandle *hStream*,
    VpeCoord *x*,
    VpeCoord *y*,
    VpeCoord *x2*,
    VpeCoord *y2*
)

VpeHandle hDoc
    Document Handle

VpeHandle hStream
    Stream Handle

VpeCoord x, y, x2, y2
    position and dimensions, y2 may be set to VFREE

Returns:

The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:

VFREE: only the y2 coordinate may be set to VFREE, not x2.
In case of an error, LastError is set.
The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:

"Rendering Objects" in the Programmer's Manual

7.16 VpeRenderBoxRTFStream

[Professional Edition and above]

Computes the dimensions of a given RTF stream, based on the method VpeWriteBoxRTFFile(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

int VpeRenderBoxRTFStream(
VpeHandle *hDoc*,
VpeHandle *hStream*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*
)

VpeHandle hDoc
Document Handle

VpeHandle hStream
Stream Handle

VpeCoord x, y, x2, y2
position and dimensions, y2 may be set to VFREE

Returns:
The method returns one of the following values, indicating the AutoBreak status:

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur
RENDER_BREAK	1	Auto Break will occur
RENDER_SKIP_BREAK	2	Auto Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (VRENDERWIDTH and VRENDERHEIGHT are not set)

Remarks:
VFREE: only the y2 coordinate may be set to VFREE, not x2.
In case of an error, LastError is set.
The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:
"Rendering Objects" in the Programmer's Manual

7.17 VpeRenderFormField

[Enterprise Edition and above]

Computes the dimensions of a given FormField text, based on the method FormField(). The dimensions are computed including the surrounding frame, if PenSize is <> 0.

int VpeRenderFormField(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *s*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions, y2 may be set to VFREE

LPCSTR s
the text to be rendered

Returns:
The method always returns

Constant Name	Value	Comment
RENDER_NO_BREAK	0	NO Auto Break will occur

Remarks:
If x2 = VFREE, the FormField will behave the same as if VpeWrite(Box) was used.

The computed dimensions can be retrieved with

```
VpeGet(hDoc, VRENDERWIDTH)
VpeGet(hDoc, VRENDERHEIGHT)
```

See also:
"Rendering Objects" in the Programmer's Manual

8 Drawing Functions

8.1 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 Text Functions in the next section.

8.2 VpeSetPen

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 by the GDI to pens with a size of 1 pixel under Win 3.x, 9x and ME. So you should always use PS_SOLID until the GDI changes.

void VpeSetPen(
    VpeHandle *hDoc*,
    VpeCoord *pen_size*,
    int *pen_style*,
    COLORREF *pen_color*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

VpeCoord pen_size
    the thickness of the pen

int pen_style
    one of the windows pen styles; possible values are:

Constant Name	Value	Comment
PS_SOLID	0	
PS_DASH	1	--------
PS_DOT	2	....
PS_DASHDOT	3	_·_·_·_
PS_DASHDOTDOT	4	_··_··_

COLORREF pen_color
    one of the the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
    0.03 (which is 0.3 mm), PS_SOLID, COLOR_BLACK

8.3 VpeNoPen

Hides the pen (the property PenSize is internally set to zero). All drawing objects that are inherited from the pen-object will use no pen (see "The Object-Oriented Style" in the Programmer's Manual for details). The pen can be made visible by setting the PenSize to a different value from 0.

```
void VpeNoPen(
    VpeHandle hDoc
)
```

VpeHandle hDoc
    Document Handle or VPE Object Handle

8.4 VpeSetPenSize

Sets the pen size.

void VpeSetPenSize(
VpeHandle *hDoc*,
VpeCoord *pen_size*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

VpeCoord pen_size
the thickness of the pen

Default:
0.03 (which is 0.3 mm)

Example:

```
VpeSetPenSize(hDoc, 0.06) // 0.6 mm
```

8.5 VpePenSize

The same as VpeSetPenSize. Sets the pen size

void VpePenSize(
    VpeHandle *hDoc*,
    VpeCoord *pen_size*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

VpeCoord pen_size
    the thickness of the pen

Default:
    0.03 (which is 0.3 mm)

8.6 VpeGetPenSize

Returns the current pen size.

VpeCoord VpeGetPenSize(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the current pen size

8.7 VpeSetPenStyle

Sets the style for the pen. You can use the PS_xyz pen styles from Windows GDI, but pen styles other than PS_SOLID are limited by the GDI to pens with a size of 1 pixel under Win 3.x, 9x, and ME. So you should always use PS_SOLID until the GDI changes.

void VpeSetPenStyle(
VpeHandle *hDoc*,
int *pen_style*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int pen_style
one of the windows pen styles; possible values are:

Constant Name	Value	Comment
PS_SOLID	0	
PS_DASH	1	-------
PS_DOT	2	....
PS_DASHDOT	3	_._._._
PS_DASHDOTDOT	4	_.._.._

Default:
PS_SOLID

8.8 VpeGetPenStyle

Returns the style of the pen.

int VpeGetPenStyle(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    one of the windows pen styles; possible values are:

Constant Name	Value	Comment
PS_SOLID	0	
PS_DASH	1	-------
PS_DOT	2	....
PS_DASHDOT	3	_._._._
PS_DASHDOTDOT	4	_.._.._

8.9 VpePenStyle

The same as VpeSetPenStyle. Use VpeSetPenStyle instead. 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 by the GDI to pens with a size of 1 pixel under Win 3.x, 9x and ME. So you should always use PS_SOLID, until the GDI changes.

void VpePenStyle(
    VpeHandle *hDoc*,
    int *pen_style*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

int pen_style
    one of the windows pen styles; possible values are:

Constant Name	Value	Comment
PS_SOLID	0	
PS_DASH	1	-------
PS_DOT	2	....
PS_DASHDOT	3	_._._._
PS_DASHDOTDOT	4	_.._.._

Default:
    PS_SOLID

8.10 VpeSetPenColor

Sets the color of the pen.

void VpeSetPenColor(
    VpeHandle *hDoc*,
    COLORREF *color*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

COLORREF color
    one of the predefined the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
    Black

8.11 VpeGetPenColor

Returns the color of the pen.

COLORREF VpeGetPenColor(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

COLORREF color
one of the predefined the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
Black

8.12 VpePenColor

The same as VpeSetPenColor. Sets the color of the pen.

void VpePenColor(
  VpeHandle *hDoc*,
  COLORREF *color*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

COLORREF color
  one of the predefined the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
  COLOR_BLACK

8.13 VpeLine

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

void VpeLine(
    VpeHandle *hDoc*,
    VpeCoord *x*,
    VpeCoord *y*,
    VpeCoord *x2*,
    VpeCoord *y2*
)

VpeHandle hDoc
    Document Handle

VpeCoord x, y
    starting coordinates

VpeCoord x2, y2
    ending coordinates

Remarks:
    VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
    For details please see "Dynamic Positioning" in the Programmer's Manual.

    There is also an "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

See Also:
    VpePolyLine

8.14 VpePolyLine

Creates a PolyLine Object using the current pen. After a call to this method, the created PolyLine Object is ready for use with the method VpeAddPolyPoint().

VpeHandle VpePolyLine(
  VpeHandle *hDoc*,
  unsigned int *size*
)

VpeHandle hDoc
  Document Handle

unsigned int size
  the size of the array (count of elements, NOT bytes)

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

Remarks:
  If you have to draw a huge number of lines at once, this method is much faster and saves memory compared to the method VpeLine().

  There is also an "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

8.15 VpeAddPolyPoint

Adds a new point to the Polyline Object specified in <p>.

void VpeAddPolyPoint(
    VpeHandle *hDoc*,
    VpeHandle *hPolyLine*,
    VpeCoord *x*,
    VpeCoord *y*
)

VpeHandle hDoc
    Document Handle

VpeHandle hPolyLine
    polyline object handle

VpeCoord x, y
    coordinate of new point

- The first point you add contains the starting coordinate
- Each added point contains the next coordinate where to draw to
- If a point is -1, -1, the next coordinate is interpreted as a NEW starting coordinate

8.16 VpeSetBkgMode

Sets the background mode. The background is the area inside of an object.

void VpeSetBkgMode(
  VpeHandle *hDoc*,
  int *mode*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

int mode
  possible values are:

Constant Name	**Value**	**Comment**
VBKG_SOLID	0	solid background color
VBKG_TRANSPARENT	1	transparent background
VBKG_GRD_LINE	2	line gradient background
VBKG_GRD_RECT	3	rectangular gradient background
VBKG_GRD_ELLIPSE	4	elliptic gradient background

Default:
  VBKG_TRANSPARENT

8.17 VpeGetBkgMode

Returns the background mode. The background is the area inside of an object.

int VpeGetBkgMode(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    possible values are:

Constant Name	Value	Comment
VBKG_SOLID	0	solid background color
VBKG_TRANSPARENT	1	transparent background
VBKG_GRD_LINE	2	line gradient background
VBKG_GRD_RECT	3	rectangular gradient background
VBKG_GRD_ELLIPSE	4	elliptic gradient background

Community Edition:

The Community Edition only supports VBKG_SOLID and VBKG_TRANSPARENT.

8.18 VpeSetBkgColor

Sets the background color. The background color is the color inside of an object. To make the background color painted, the Background Mode must be VBKG_SOLID.

void VpeSetBkgColor(
VpeHandle *hDoc*,
COLORREF *color*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

COLORREF color
one of the predefined "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
COLOR_WHITE (but the Transparent Background Mode is activated!)

8.19 VpeGetBkgColor

Returns the current background color. The background color is the color inside of an object.

COLORREF VpeGetBkgColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    The current background color.

8.20 VpeSetBkgGradientStartColor

[Not supported by the Community Edition]

Specifies the gradient start color. To make the gradient painted, the Background Mode must be set to a gradient mode.

void VpeSetBkgGradientStartColor(
  VpeHandle *hDoc*,
  COLORREF *color_start*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

COLORREF color_start
  one of the predefined "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
  COLOR_WHITE (but the Transparent Background Mode is activated!)

Remarks:
  Gradients do only work for rectangular objects like Boxes, Text, Ellipses, etc. Gradients are not drawn in Pies and Polygons. If a gradient mode is selected, hatching for the same object is not possible.

Example:

```
VpeSetBkgMode(hDoc, VBKG_GRD_LINE)
VpeSetBkgGradientStartColor(hDoc, COLOR_BLUE)
VpeSetBkgGradientEndColor(hDoc, COLOR_GREEN)
VpePrintBox(hDoc, 1, 1, "Hello World!")
```

  Will draw the text "Hello World!" with a gradient running from blue to green in the background.

8.21 VpeGetBkgGradientStartColor

[Not supported by the Community Edition]

Returns the gradient start color.

COLORREF VpeGetBkgGradientStartColor(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:
the RGB color of the gradient start color

8.22 VpeSetBkgGradientEndColor

[Not supported by the Community Edition]

Specifies the gradient end color. To make the gradient painted, the Background Mode must be set to a gradient mode.

void VpeSetBkgGradientEndColor(
  VpeHandle *hDoc*,
  COLORREF *color_end*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

COLORREF color_end
  one of the predefined "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
  COLOR_BLACK (but the Transparent Background Mode is activated!)

Remarks:
  Gradients do only work for rectangular objects like Boxes, Text, Ellipses, etc. Gradients are not drawn in Pies and Polygons. If a gradient mode is selected, hatching for the same object is not possible.

Example:

```
VpeSetBkgMode(hDoc, VBKG_GRD_LINE)
VpeSetBkgGradientStartColor(hDoc, COLOR_BLUE)
VpeSetBkgGradientEndColor(hDoc, COLOR_GREEN)
VpePrintBox(hDoc, 1, 1, "Hello World!")
```

Will draw the text "Hello World!" with a gradient running from blue to green in the background.

8.23 VpeGetBkgGradientEndColor

[Not supported by the Community Edition]

Returns the gradient end color.

COLORREF VpeGetBkgGradientEndColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the RGB color of the gradient end color

8.24 VpeSetBkgGradientTriColor

[Not supported by the Community Edition]

Activates / deactivates the tri-color gradient mode. If activated, you can specify a third middle color, so the gradient is drawn from the start color to the middle color and then to the end color. This allows to draw nice three-dimensional gradients.

void VpeSetBkgGradientTriColor(
VpeHandle *hDoc*,
int *on_off*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int on_off

Value	Description
True	On
False	Off

Default:
False

8.25 VpeGetBkgGradientTriColor

[Not supported by the Community Edition]

Returns the tri-color gradient mode.

int VpeGetBkgGradientTriColor(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:

Value	Description
True	On
False	Off

8.26 VpeSetBkgGradientMiddleColorPosition

[Not supported by the Community Edition]

Specifies the position of the middle color for tri-color gradients.

void VpeSetBkgGradientMiddleColorPosition(
VpeHandle *hDoc*,
int *position*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int position
the position in percent of the object’s height (or width, if the gradient is rotated)

Default:
50 %

Example:

```
VpeSetBkgGradientTriColor(hDoc, TRUE);
VpeSetBkgGradientMiddleColorPosition(hDoc, 35);
VpeSetBkgGradientStartColor(hDoc, COLOR_LTGRAY);
VpeSetBkgGradientMiddleColor(hDoc, COLOR_WHITE);
VpeSetBkgGradientEndColor(hDoc, COLOR_DKGRAY);
VpePrint(hDoc, 1, 1, "Hello World!")
```

Will draw the text "Hello World!" with a gradient running from light grey over white (at a position which is 35% of the object’s height) to dark grey, which will give it a silver cylindrical appearance.

8.27 VpeGetBkgGradientMiddleColorPosition

[Not supported by the Community Edition]

Returns the gradient middle color.

int VpeGetBkgGradientMiddleColorPosition(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the position in percent

8.28 VpeSetBkgGradientMiddleColor

[Not supported by the Community Edition]

Specifies the gradient middle color for tri-color gradients. To make the gradient painted, the Background Mode must be set to a gradient mode and the Tri Color Mode must be activated.

void VpeSetBkgGradientMiddleColor(
VpeHandle *hDoc*,
COLORREF *color_middle*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

COLORREF color_middle
one of the predefined "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:

COLOR_WHITE (but the Transparent Background Mode is activated and Tri Color Mode is deactivated!)

Remarks:

Gradients do only work for rectangular objects like Boxes, Text, Ellipses, etc. Gradients are not drawn in Pies and Polygons. If a gradient mode is selected, hatching for the same object is not possible.

Example:

```
VpeSetBkgGradientTriColor(hDoc, TRUE);
VpeSetBkgGradientMiddleColorPosition(hDoc, 35);
VpeSetBkgGradientStartColor(hDoc, COLOR_LTGRAY);
VpeSetBkgGradientMiddleColor(hDoc, COLOR_WHITE);
VpeSetBkgGradientEndColor(hDoc, COLOR_DKGRAY);
VpePrint(hDoc, 1, 1, "Hello World!")
```

Will draw the text "Hello World!" with a gradient running from light grey over white (at a position which is 35% of the object’s height) to dark grey, which will give it a silver cylindrical appearance.

8.29 VpeGetBkgGradientMiddleColor

[Not supported by the Community Edition]

Returns the gradient middle color.

COLORREF VpeGetBkgGradientMiddleColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the RGB color of the gradient middle color

8.30 VpeSetBkgGradientRotation

[Not supported by the Community Edition]

Specifies the rotation for the line gradient.

void VpeSetBkgGradientRotation(
VpeHandle *hDoc*,
int *angle*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int angle
rotation angle clockwise in 0.1 degrees
possible values are: 0, 900, 1800, 2700

Default:
0 degrees

Example:

```
VpeSetBkgGradientRotation(hDoc, 900)
VpeSetBkgMode(hDoc, VBKG_GRD_LINE)
VpeSetBkgGradientStartColor(hDoc, COLOR_BLUE)
VpeSetBkgGradientEndColor(hDoc, COLOR_GREEN)
VpePrintBox(hDoc, 1, 1, "Hello World!")
```

Will draw the text "Hello World!" with a gradient rotated by 90 degrees to the right, running from blue to green in the background.

8.31 VpeGetBkgGradientRotation

[Not supported by the Community Edition]

Returns the rotation angle of the line gradient.

int VpeSetBkgGradientRotation(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:
rotation angle clockwise in 0.1 degrees
possible values are: 0, 900, 1800, 2700

8.32 VpeSetBkgGradientPrint

[Not supported by the Community Edition]

Because gradients drawn on b/w printers waste toner (or ink) and might make text and other things in the foreground unreadable, you can specify how gradients are printed. This property does not affect how gradients are drawn in the preview.

void VpeSetBkgGradientPrint(
VpeHandle *hDoc*,
int *mode*
)

VpeHandle hDoc
Document Handle

int mode

Constant Name	Value	Comment
VGRD_PRINT_AUTO	0	if printer is a color printer, the gradient is printed, otherwise the alternate solid color is used
VGRD_PRINT_GRADIENT	1	the gradient is always printed
VGRD_PRINT_SOLID	2	the alternate solid color is always printed

Default:
VGRD_PRINT_AUTO

Remarks:
In contrast to all other graphical object properties, this property is valid for ALL objects in the document at once.

We experienced, that even some color printer drivers identify themselves as b/w printers (for example the Epson Stylus PHOTO 700 on Win-NT (driver v2.05) identifies itself as b/w printer, but the Epson Stylus Color ESC/P2 driver (shipped with Win-NT 4.0 SP 3) identifies itself as color printer).

Printer drivers are manufactured by vendors independent of IDEAL Software; we make no warranty, implied or otherwise, regarding these product’s performance or reliability.

If VPE should print on a color printer in VGRD_PRINT_AUTO mode gradients in b/w (so the printer driver identified itself wrongly as b/w printer), use VGRD_PRINT_GRADIENT instead.

Example:

```
VpeSetBkgGradientPrint(hDoc, VGRD_PRINT_SOLID)
VpeSetBkgGradientPrintSolidColor(hDoc, COLOR_LTGRAY)
```

Will force VPE to draw a very light grey color in the background of gradient objects when they are printed.

```
VpeSetBkgGradientPrint(hDoc, VGRD_PRINT_AUTO)
VpeSetBkgGradientPrintSolidColor(hDoc, COLOR_LTGRAY)
```

VPE will check the printer during the print job, if it is a color printer. If so, the gradients are drawn. Otherwise VPE will draw a very light grey color in the background of gradient objects when they are printed.

8.33 VpeSetBkgGradientPrintSolidColor

[Not supported by the Community Edition]

Specifies the alternative solid color, which shall be drawn depending on the setting for BkgGradientPrint.

void VpeSetBkgGradientPrintSolidColor(
VpeHandle *hDoc*,
COLORREF *color*
)

VpeHandle hDoc
Document Handle

COLORREF color
one of the predefined "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
COLOR_LTGRAY

Remarks:
In contrast to all other graphical object properties, this property is valid for ALL objects in the document at once.

8.34 VpeSetTransparentMode

Sets the background mode to transparent / not transparent.
The background is the area inside of an object. You can use VpeSetBkgMode also.

void VpeSetTransparentMode(
    VpeHandle *hDoc*,
    int *on_off*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

int on_off

Value	Description
True	on (transparent)
False	off (not transparent)

Default:
    True

8.35 VpeGetTransparentMode

Returns the current background mode. The background is the area inside of an object.

You can use VpeGetBkgMode also.

int VpeGetTransparentMode(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:

Value	**Description**
True	on (transparent)
False	off (not transparent)

8.36 VpeSetHatchStyle

[Not supported by the Community Edition]

Sets the hatch style. All rectangular objects - except charts, barcodes and images - can be hatched with the predefined windows hatch styles.

void VpeSetHatchStyle(
VpeHandle *hDoc*,
int *style*
)

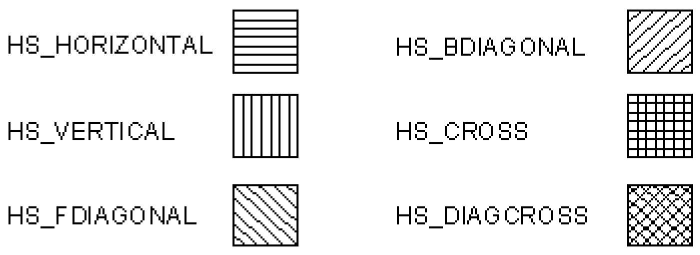
VpeHandle hDoc
Document Handle or VPE Object Handle

int style
hatch style; possible values are:

Constant Name	Value	Comment
HS_NONE	-1	see Possible styles below
HS_HORIZONTAL	0	see Possible styles below
HS_VERTICAL	1	see Possible styles below
HS_FDIAGONAL	2	see Possible styles below
HS_BDIAGONAL	3	see Possible styles below
HS_CROSS	4	see Possible styles below
HS_DIAGCROSS	5	see Possible styles below

Default:
HS_NONE, which means NO hatching

Possible styles are:



8.37 VpeGetHatchStyle

[Not supported by the Community Edition]

Returns the current hatch style. All rectangular objects - except charts, barcodes and images - can be hatched with the predefined windows hatch styles.

int VpeGetHatchStyle(
 VpeHandle *hDoc*
)

VpeHandle hDoc
 Document Handle or VPE Object Handle

Returns:
 hatch style; possible values are:

Constant Name	**Value**	**Comment**
HS_NONE	-1	see Possible styles in VpeSetHatchStyle
HS_HORIZONTAL	0	see Possible styles in VpeSetHatchStyle
HS_VERTICAL	1	see Possible styles in VpeSetHatchStyle
HS_FDIAGONAL	2	see Possible styles in VpeSetHatchStyle
HS_BDIAGONAL	3	see Possible styles in VpeSetHatchStyle
HS_CROSS	4	see Possible styles in VpeSetHatchStyle
HS_DIAGCROSS	5	see Possible styles in VpeSetHatchStyle

8.38 VpeSetHatchColor

[Not supported by the Community Edition]

Sets the RGB color of the hatching.

void VpeSetHatchColor(
VpeHandle *hDoc*,
COLORREF *color*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

COLORREF color
one of the predefined "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
COLOR_BLACK

8.39 VpeGetHatchColor

[Not supported by the Community Edition]

Returns the RGB color of the hatching.

COLORREF VpeGetHatchColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the RGB color of the hatching

8.40 VpeSetCornerRadius

[Not supported by the Community Edition]

The radius of rounded corners. Setting this property different from zero causes Boxes, Text and Rich Text to be drawn with rounded corners that have the given radius.

void VpeSetCornerRadius(
    VpeHandle *hDoc*,
    VpeCoord *radius*
)

VpeHandle hDoc
    Document Handle

VpeCoord radius
    the corner radius

Default:
    0

Remarks:
    On Win 9x/Me objects with rounded corners may not have a gradient. Gradients are drawn outside the rounded corners on Win 9x/Me.

8.41 VpeGetCornerRadius

[Not supported by the Community Edition]

Returns the radius of rounded corners.

VpeCoord VpeGetCornerRadius(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the radius of rounded corners

8.42 VpeBox

Draws a box object at position x, y with the right border at x2 and the bottom border at y2. The current penstyle, background mode and background color are used.

void VpeBox(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions of the box

Remarks:

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

There is also an "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

8.43 VpePolygon

Creates a Polygon object with the current pen, background and hatchstyle. After a call to this method, the created Polygon Object is ready for use with the method VpeAddPolygonPoint().

VpeHandle VpePolygon(
VpeHandle *hDoc*,
unsigned int *size*
)

VpeHandle hDoc
Document Handle

unsigned int size
the size of the array (count of elements, NOT bytes).

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

Remarks:
There is also an "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

8.44 VpeAddPolygonPoint

Adds a new point to the polygon object specified by the handle *hPolygon*.

void VpeAddPolygonPoint(
    VpeHandle *hDoc*,
    VpeHandle *hPolygon*,
    VpeCoord *x*,
    VpeCoord *y*
)

VpeHandle hDoc
    Document Handle

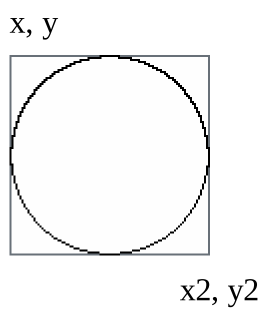
VpeHandle hPolygon
    polygon object handle

VpeCoord x, y
    coordinate of new point

- 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

8.45 VpeEllipse

Draws an ellipse or circle within the given rectangle.



void VpeEllipse(
    VpeHandle *hDoc*,
    VpeCoord *x*,
    VpeCoord *y*,
    VpeCoord *x2*,
    VpeCoord *y2*
)

VpeHandle hDoc
    Document Handle

VpeCoord x, y, x2, y2
    surrounding rectangle of the ellipse

Remarks:
    VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
    For details please see "Dynamic Positioning" in the Programmer's Manual.

    There is also an "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

8.46 VpePie

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

void VpePie(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
int *begin_angle*,
int *end_angle*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
surrounding rectangle of the pie

int begin_angle
angle in 0.1 degrees, clockwise

int end_angle
angle in 0.1 degrees, clockwise

Remarks:

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

There is also an "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

9 Text Functions

9.1 Text Functions

These methods and properties deal with text formatting and output.

For important details about fonts and font handling, please see the Programmer's Manual, chapter "Programming Techniques", subchapter "Fonts and Font Handling".

9.2 VpeSetFont

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.

void VpeSetFont(
    VpeHandle *hDoc*,
    LPCSTR *name*,
    int *size*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

LPCSTR name
    font name (e.g. "Arial")

int size
    font size in points (not in metric or inch units!)

Default:

Platform	Value
Windows and Mac OS X	Arial, 10pt
Any other	Helvetica, 10pt

Remarks:

VPE can only handle True-Type fonts and the 14 built-in PostScript fonts. It can not use bitmap fonts, like for example “MS Sans Serif”.

For important details about fonts and font handling, please see the Programmer's Manual, chapter "Programming Techniques", subchapter "Fonts and Font Handling".

9.3 VpeSelectFont

The same as VpeSetFont().

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.

void VpeSelectFont(
VpeHandle *hDoc*,
LPCSTR *name*,
int *size*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

LPCSTR name
font name (e.g. "Arial")

int size
font size in points (not in metric or inch units!)

Default:

Platform	Value
Windows and Mac OS X	Arial, 10pt
Any other	Helvetica, 10pt

Remarks:

VPE can only handle True-Type fonts and the 14 built-in PostScript fonts. It can not use bitmap fonts, like for example “MS Sans Serif”.

For important details about fonts and font handling, please see the Programmer's Manual, chapter "Programming Techniques", subchapter "Fonts and Font Handling".

9.4 VpeSetFontName

Sets a font. You can only select True-Type fonts installed on the machine on which VPE is running. Other fonts than True-Type fonts are not supported.

void VpeSetFontName(
  VpeHandle *hDoc*,
  LPCSTR *name*,
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

LPCSTR name
  font name (e.g. "Arial")

Default:

Platform	**Value**
Windows and Mac OS X	Arial
Any other	Helvetica

Remarks:

VPE can only handle True-Type fonts and the 14 built-in PostScript fonts. It can not use bitmap fonts, like for example “MS Sans Serif”.

For important details about fonts and font handling, please see the Programmer's Manual, chapter "Programming Techniques", subchapter "Fonts and Font Handling".

9.5 VpeGetFontName

Returns the current font name.

void VpeGetFontName(
VpeHandle *hDoc*,
LPSTR *value*,
UINT **size*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

LPSTR value
Pointer to a buffer that receives the string font name (e.g. "Arial"). This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character. If value is NULL, and size is non-NULL, the function returns and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:
the current font name

9.6 VpeSetFontSize

Sets a font size.

void VpeSetFontSize(
    VpeHandle *hDoc*,
    int *size*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

int size
    font size in points (not in metric or inch units!)

Default:
    10 pt

9.7 VpeGetFontSize

Returns the current font size.

int VpeSetFontSize(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the current font size in points (not in metric or inch units!)

9.8 VpeSetFontSubstitution

Substitutes a given font with another font. By using this method, VPE will substitute fonts immediately when creating a document. This is especially useful on Non-Windows platforms when importing RTF (Rich Text) documents. Often RTF documents are created on Windows platforms and use Windows specific True-Type fonts. With this method you can instruct VPE to substitute for example the True-Type font "Arial" with the Base 14 font "Helvetica", so Helvetica is used in place of Arial.

void VpeSetFontSubstitution(
VpeHandle *hDoc,*
LPCTSTR *original_font,*
LPCTSTR *subst_font*
)

VpeHandle hDoc
Document Handle

LPCTSTR original_font
the font which shall be subtituted by the subst_font

LPCTSTR subst_font
the font which shall be used in place of the original_font

Remarks:

In contrast to nearly all other methods, the font substitution is not related to the current document! It is active during the lifetime of your application for ALL documents which are currently open, as well as for ALL documents you are going to create after calling this method! Use this method with care.

You can reset all font substitution settings at once by calling the method VpePurgeFontSubstitution().

On Windows, this method also affects the preview and printing. This method does not affect the document export to other formats than PDF. There is a second method VpeSetFontControl() to substitute fonts, which is only active during the export to PDF documents.

For important details about fonts and font handling, please see the Programmer's Manual, chapter "Programming Techniques", subchapter "Fonts and Font Handling".

Example:

```
VpeSetFontSubstitution(hDoc, "Arial", "Helvetica")
```

Substitutes the True-Type font "Arial" with the Base 14 font "Helvetica", so Helvetica is used in place of Arial.

9.9 VpePurgeFontSubstitution

Resets all font substitution settings at once. This clears all font substitution rules, so no substitution takes place.

void VpePurgeFontSubstitution(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

9.10 VpeSetCharset

Sets the charset.

void VpeSetCharset(
VpeHandle *hDoc*,
long *charset*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

long charset
character set, possible values are:

Constant Name	**Value**	**Comment**
VCHARSET_DEFAULT	1	Windows: Choses the VCHARSET_WIN_ character set that fits to the localization settings of the current user. All other platforms: sets the charset to ISO_LATIN_1
VCHARSET_SYMBOL	2	Required for using Symbol Fonts, like WingDings
VCHARSET_MAC_ROMAN	77	Macintosh Roman
Character sets compatible to the character sets of the Windows operating system:		
VCHARSET_WIN_ANSI	0	Western character set: Afrikaans, Basque, Catalan, Danish, Dutch, English, Esperanto, Faroese, Finnish, French, Frisian, Galician, German, Icelandic, Indonesian, Interlingua, Irish, Italian, Latin, Malay, Maltese, Norwegian, Pilipino, Portuguese, Spanish, Swahili, Swedish, Welsh
VCHARSET_WIN_HEBREW	177	Hebrew
VCHARSET_WIN_ARABIC	178	Arabic
VCHARSET_WIN_GREEK	161	Greek
VCHARSET_WIN_TURKISH	162	Turkish
VCHARSET_WIN_VIETNAMESE	163	Vietnamese
VCHARSET_WIN_THAI	222	Thai
VCHARSET_WIN_EAST_EUROPE	238	Albanian, Belarusian, Croatian, Czech, Hungarian, Polish, Romanian, Serbian, Slovak, Slovenian
VCHARSET_WIN_CYRILLIC	204	Bulgarian, Russian, Serbian and Ukrainian (Cyrillic)
VCHARSET_WIN_BALTIC	186	Estonian, Latvian and Lithuanian
Character sets compatible to the ISO character sets (taken from unicode.org):		
VCHARSET_ISO_LATIN_1	50	Latin-1, Western character set (ISO-8859-1)
VCHARSET_ISO_LATIN_2	51	Latin-2, East European (ISO-8859-2)
VCHARSET_ISO_LATIN_3	52	Latin-3, South European (ISO-8859-3)
VCHARSET_ISO_LATIN_4	53	Latin-4, Baltic (ISO-8859-4)
VCHARSET_ISO_CYRILLIC	54	Cyrillic (ISO-8859-5)

VCHARSET_ISO_ARABIC	55	Arabic (ISO-8859-6)
VCHARSET_ISO_GREEK	56	Greek (ISO-8859-7)
VCHARSET_ISO_HEBREW	57	Hebrew (ISO-8859-8)
VCHARSET_ISO_LATIN_5	58	Latin-5, Turkish (ISO-8859-9)
VCHARSET_ISO_LATIN_6	59	Latin-6, Nordic (ISO-8859-10)
VCHARSET_ISO_THAI	60	Thai (ISO-8859-11)
VCHARSET_ISO_LATIN_7	62	Latin-7, Baltic (ISO-8859-13)
VCHARSET_ISO_LATIN_8	63	Latin-8, Celtic (ISO-8859-14)
VCHARSET_ISO_LATIN_9	64	Latin-9, Western (ISO-8859-15)

Default:

DEFAULT_CHARSET

Remarks:

The Mac and Iso charsets are intended for Non-Windows platforms. They should not be used with .NET, they will not function under .NET. They can be used on Windows with the ActiveX or DLL, but in this case only for PDF file creation, the preview will display wrong characters.

The supported character sets for the **Base 14 fonts are limited** to WinAnsi, WinEastEurope, WinTurkish, WinBaltic, IsoLatin1, IsoLatin2, IsoLatin5, IsoLatin7, IsoLatin9. MacRoman is nearly fully supported, but the following characters are missing:
unicode 221e (ansi code 176) INFINITY
unicode 220f (ansi code 184) GREEK CAPITAL LETTER PI
unicode 03c0 (ansi code 185) GREEK SMALL LETTER PI
unicode 222b (ansi code 186) INTEGRAL
unicode 03a9 (ansi code 189) GREEK CAPITAL LETTER OMEGA
unicode 2248 (ansi code 197) ALMOST EQUAL TO (asymptotic to)
unicode f8ff (ansi code 240) Apple Logo.
For other character sets, True-Type fonts are required.

When exporting to PDF, some characters are missing in the WIN_ARABIC and ISO_GREEK codepages. This is due to the technique, VPE uses to define character sets within PDF documents – and that Adobe did not define those characters in their Glyphlist.
A future version of VPE will overcome this problem.

The following characters of the charset WIN_ARABIC are missing:
Unicode U+06A9, Code 152, Arabic Letter "Keheh"
Unicode U+06BE, Code 170, Arabic Letter " Heh Doachashmee"
Both characters are for Persian and Urdu scripts only. So outputting arabic text is not affected.
NOTE: the ISO_ARABIC codepage is fully supported.

The following character of the charset ISO_GREEK is missing:
Unicode U+20AF, Code 165, Drachma Sign
NOTE: the WIN_GREEK codepage is fully supported.

To display characters from a **symbol font**, it is required to set the Charset to SYMBOL_CHARSET.

To use a specific charset, it might be required to use a font which supports this charset. For example, if you use the font Arial with the THAI charset, this font is not available on western versions of Windows by default.

VPE does not support double-byte character sets, nor does it support UNICODE.

RTL reading (right-to-left) is not supported.
In a future release a UNICODE version of VPE will be created.

9.11 VpeGetCharset

Returns the current charset.

long VpeGetCharset(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

returns:
    the current character set

9.12 VpeSetFontAttr

Sets all font-attributes at once.

void VpeSetFontAttr(
VpeHandle *hDoc*,
int *alignment*,
int *bold*,
int *underline*,
int *italic*,
int *strikeout*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int alignment
possible values are:

Constant Name	**Value**	**Comment**
ALIGN_LEFT	0	
ALIGN_RIGHT	1	
ALIGN_CENTER	2	
ALIGN_JUSTIFIED	3	
ALIGN_JUSTIFIED_AB	5	

int bold

Value	**Description**
True	bold
False	not bold

int underline

Value	**Description**
True	underlined
False	not underlined

int italic

Value	**Description**
True	italic
False	not italic

int strikeout

Value	**Description**
True	strikeout
False	not strikeout

Default:

ALIGN_LEFT, False, False, False, False

Remarks:

Italic fonts are a bit higher than non-italic fonts. This is caused by the Windows System GDI. The consequence is, that italic text needs more height, which might result in clipped (not drawn) text, in case the height returned by a text-render method for a non-italic font is used for an italic font.

ALIGN_JUSTIFIED_AB: Text will be aligned justified. In contrast to ALIGN_JUSTIFIED, which aligns the last line left aligned, the last line of the text will also be aligned justified - if it is not ending with a CR / LF character. This flag works only for plain text, not for RTF.

9.13 VpeSetTextAlignment

Sets the text alignment

void VpeSetTextAlignment(
VpeHandle *hDoc*,
int *alignment*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int alignment
possible values are:

Constant Name	Value	Comment
ALIGN_LEFT	0	
ALIGN_RIGHT	1	
ALIGN_CENTER	2	
ALIGN_JUSTIFIED	3	
ALIGN_JUSTIFIED_AB	5	

Default:
ALIGN_LEFT

Remarks:
ALIGN_JUSTIFIED_AB: Text will be aligned justified. In contrast to ALIGN_JUSTIFIED, which aligns the last line left aligned, the last line of the text will also be aligned justified - if it is not ending with a CR / LF character. This flag works only for plain text, not for RTF.

9.14 VpeGetTextAlignment

Returns the text alignment attribute.

void VpeGetTextAlignment(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:
possible values are:

Constant Name	Value	Comment
ALIGN_LEFT	0	
ALIGN_RIGHT	1	
ALIGN_CENTER	2	
ALIGN_JUSTIFIED	3	
ALIGN_JUSTIFIED_AB	5	

9.15 VpeSetAlign

The same as SetTextAlignment. Sets the text alignment. Use SetTextAlignment instead.

void VpeSetAlign(
VpeHandle *hDoc*,
int *alignment*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int alignment
possible values are:

Constant Name	Value	Comment
ALIGN_LEFT	0	
ALIGN_RIGHT	1	
ALIGN_CENTER	2	
ALIGN_JUSTIFIED	3	
ALIGN_JUSTIFIED_AB	5	

Default:
ALIGN_LEFT

Remarks:
ALIGN_JUSTIFIED_AB: Text will be aligned justified. In contrast to ALIGN_JUSTIFIED, which aligns the last line left aligned, the last line of the text will also be aligned justified - if it is not ending with a CR / LF character. This flag works only for plain text, not for RTF.

9.16 VpeSetBold

Sets the text bold attribute on / off.

void VpeSetBold(
VpeHandle *hDoc*,
int *bold*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int bold

Value	Description
True	bold
False	not bold

Default:
False

9.17 VpeGetBold

Returns the text bold attribute.

void VpeGetBold(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:

Value	Description
True	bold
False	not bold

9.18 VpeSetUnderlined

Sets the text underline on / off

void VpeSetUnderlined(
VpeHandle *hDoc*,
int *underlined*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int underlined

Value	Description
True	underlined
False	not underlined

Default:
False

9.19 VpeGetUnderlined

Returns the text underline attribute.

void VpeGetUnderlined(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:

Value	Description
True	underlined
False	not underlined

9.20 VpeSetUnderline

Sets the text underline on / off

void VpeSetUnderline(
VpeHandle *hDoc*,
int *underline*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int underline

Value	Description
True	underlined
False	not underlined

Default:
False

9.21 VpeGetUnderline

Returns the text underline attribute.

void VpeGetUnderline(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:

Value	Description
True	underlined
False	not underlined

9.22 VpeSetItalic

Sets text italic on / off

void VpeSetItalic(
VpeHandle *hDoc*,
int *italic*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int underlined

Value	Description
True	italic
False	not italic

Default:
False

Remarks:
Italic fonts are a bit higher than non-italic fonts. This is caused by the Windows System GDI. The consequence is, that italic text needs more height, which might result in clipped (not drawn) text, in case the height returned by a text-render method for a non-italic font is used for an italic font.

9.23 VpeGetItalic

Returns text italic attribute.

void VpeGetItalic(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:

Value	Description
True	italic
False	not italic

9.24 VpeSetStrikeOut

Sets text strikeout on / off

void VpeSetStrikeOut(
  VpeHandle *hDoc*,
  int *strikeout*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

int strikeout

Value	Description
True	strikeout
False	not strikeout

Default:
  False

9.25 VpeGetStrikeOut

Returns the text strikeout attribute.

void VpeGetStrikeOut(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:

Value	Description
True	strikeout
False	not strikeout

9.26 VpeSetTextColor

Sets the text color. This is the foreground color which also applies to Barcodes, RTF and Charts.

```
void VpeSetTextColor(
    VpeHandle hDoc,
    COLORREF color
)
```

VpeHandle hDoc
Document Handle or VPE Object Handle

COLORREF color
one of the predefined "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
COLOR_BLACK

9.27 VpeGetTextColor

Returns the text color.

This is the foreground color which also applies to Barcodes, RTF and Charts.

void VpeGetTextColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    The RGB color of the text.

9.28 VpeWrite

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.

```
VpeCoord VpeWrite(
    VpeHandle hDoc,
    VpeCoord x,
    VpeCoord y,
    VpeCoord x2,
    VpeCoord y2,
    LPCSTR text
)
```

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions

LPCSTR text
the text to output

Returns:
the bottom y-coordinate generated by the output

Remarks:
Do not use leading blanks for text output. You will realize, that this might result under special circumstances in a x- direction misalignment at the beginning of the lines, even if you are using a non-proportional font like "Courier New". This is normal and happens due to the technique, how VPE renders text. Instead of using leading blanks, eliminate those blanks and position the first character as needed by setting a correct x-start coordinate.

You should never use static values for Y2 when doing text output, e.g. Write(1, 1, 5, 3, "Hello"). It may happen that you see text on the screen and on some printers, but not on other printers. This is, because some printer-drivers clip complete lines, in case only one pixel is outside their clipping rectangle. Also some printer-drivers add internal offsets to the bottom of text, so that it is even clipped if the text itself fits into the rectangle.
Use VFREE - or for performance reasons: Render the height of a line and use this value.

It is not possible to change font styles *within* one string. For such need, you should use the Professional Edition, which can interpret and display Rich Text Format (RTF).

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

If the very first character of the output string is a '**[**' (square bracket), VPE expects a sequence of Embedded Flags. In order to print a square bracket as the very first character, output two consecutive square brackets, i.e. '**[[**'. For details, see "Embedded Flag-Setting" in the Programmer's Manual.

9.29 VpeWriteBox

The 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.

VpeCoord VpeWriteBox(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*,
  LPCSTR *text*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y, x2, y2
  position and dimensions

LPCSTR text
  the text to output

Returns:
  the bottom y-coordinate generated by the output

Remarks:
  It is not possible to change font styles *within* one string. For such need, you should use the Professional Edition, which can interpret and display Rich Text Format (RTF).

  VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
  For details please see "Dynamic Positioning" in the Programmer's Manual.

  If the very first character of the output string is a '**[**' (square bracket), VPE expects a sequence of Embedded Flags. In order to print a square bracket as the very first character, output two consecutive square brackets, i.e. '**[[**'. For details, see "Embedded Flag-Setting" in the Programmer's Manual.

9.30 VpePrint

The same as VpeWrite() with the parameters (x, y, VFREE, VFREE).

The pen is invisible.

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() not using VFREE.

VpeCoord VpePrint(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  LPCSTR *text*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y
  position

LPCSTR text
  the text to output

Returns:
  the bottom y-coordinate generated by the output

Remarks:
  It is not possible to change font styles *within* one string. For such need, you should use the Professional Edition, which can interpret and display Rich Text Format (RTF).

  VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
  For details please see "Dynamic Positioning" in the Programmer's Manual.

  If the very first character of the output string is a '**[**' (square bracket), VPE expects a sequence of Embedded Flags. In order to print a square bracket as the very first character, output two consecutive square brackets, i.e. '**[[**'. For details, see "Embedded Flag-Setting" in the Programmer's Manual.

9.31 VpePrintBox

The same as VpeWriteBox() with the parameters (x, y, VFREE, VFREE).

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() not using VFREE.

VpeCoord VpePrintBox(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  LPCSTR *text*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y
  position

LPCSTR text
  the text to output

Returns:
  the bottom y-coordinate generated by the output

Remarks:
  It is not possible to change font styles *within* one string. For such need, you should use the Professional Edition, which can interpret and display Rich Text Format (RTF).

  VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
  For details please see "Dynamic Positioning" in the Programmer's Manual.

  If the very first character of the output string is a '**[**' (square bracket), VPE expects a sequence of Embedded Flags. In order to print a square bracket as the very first character, output two consecutive square brackets, i.e. '**[[**'. For details, see "Embedded Flag-Setting" in the Programmer's Manual.

9.32 VpeDefineHeader

Exactly the same as VpeWriteBox, but the object is inserted automatically on each new page. The string may contain the sequence "@PAGE” which will be replaced by the current page number.

The header is inserted into the document by VPE on the current page and on all successively created pages. This function is intended to be used for very simple headers. For complex headers / footers see "Headers and Footers" in the Programmer's Manual.

void VpeDefineHeader(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*,
  LPCSTR *text*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y, x2, y2
  position and dimensions

LPCSTR text
  the header text to output

Remarks:
  After the header has been defined once, it can not be redefined nor deleted.

9.33 VpeDefineFooter

Exactly the same as VpeWriteBox, but the object is inserted automatically on each new page. The string may contain the sequence "@PAGE” which will be replaced by the current page number.

The footer is inserted into the document by VPE on the current page and on all successively created pages. This function is intended to be used for very simple footers. For complex headers / footers see "Headers and Footers" in the Programmer's Manual.

void VpeDefineFooter(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*,
  LPCSTR *text*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y, x2, y2
  position and dimensions

LPCSTR text
  the footer text to output

Remarks:
  After the footer has been defined once, it can not be redefined nor deleted.

9.34 VpeSetCharPlacement

[Professional Edition and above]

Allows to specify a constant offset from one character cell to another for text objects (**not RTF**). This is very useful for filling in forms that have pre-printed columns (cells) for each letter. The provided offset is understood as a "character cell width", VPE will print each character centered within this "cell".

void VpeSetCharPlacement(
  VpeHandle *hDoc*,
  VpeCoord *distance*
)

VpeHandle hDoc
  Document Handle

VpeCoord distance
  the distance from one character cell to the next

Default:
  0 (no character placement)

Remarks:
  When using CharPlacement, the TextAlignment must be either ALIGN_LEFT or ALIGN_RIGHT.

Example:

```
VpeSetCharPlacement(hDoc, 0.5)
VpePrint(hDoc, 1, 1, "123")
```

VPE will generate for each character in the given string internally a cell of 5 mm width. Each number of the string "123" will be printed centered within a cell. The first cell will be placed with the "1" at x-position 1.0 (1cm from the left paper margin), "2" at x-position 1.5 and "3" at x-position 2.0.

10 Picture Functions

10.1 Picture Functions

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

See also "Pictures" in the Programmer's Manual for a detailed description on how to use image files.

10.2 VpeSetPictureCacheSize

[Not supported by the Community Edition]

To increase performance, VPE has a build-in dynamic image cache. With this method you can set the maximum amount of memory, VPE will use to cache image files.

See "Image Cache" in the Programmer's Manual for details.

void VpeSetPictureCacheSize(
  long *size*
)

long size
  size of image cache in kilobytes (NOT megabytes)

Default:
  16384 (= 16 MB)

Remarks:
  The setting of this property has great impact on performance. If you are using a lot of different images or huge images, and you know in advance that the machine has enough memory, we recommend to raise the cache size. The factory default size has been chosen very conservatively small.

  Note, that the image cache does not use any memory unless an image is loaded into memory, and it uses only as much memory as required.

Community Edition:
  The Community Edition does not provide caching of images.

See Also:
  "Pictures" in the Programmer's Manual

10.3 VpeGetPictureCacheSize

Returns the current size of the image cache in kilobytes.

See also "Image Cache" in the Programmer's Manual.

long VpeGetPictureCacheSize(
)

Returns:
 the current size of the image cache in kilobytes

Community Edition:
 The Community Edition does not provide caching of images.

See Also:
 "Pictures" in the Programmer's Manual

10.4 VpeGetPictureCacheUsed

Returns the used space of the image cache in kilobytes.

See also "Image Cache" in the Programmer's Manual.

long VpeGetPictureCacheUsed(
)

Returns:

the used space of the image cache in kilobytes

See Also:

"Pictures" in the Programmer's Manual

10.5 VpeGetPictureTypes

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

void VpeGetPictureTypes(
  VpeHandle *hDoc*,
  LPSTR *s*,
  int *size*
)

VpeHandle hDoc
  Document Handle

LPSTR s
  the receive string with the type-list

int size
  the maximum number of bytes, the string s can receive

Returns:
  A list of picture types in parameter "s" of the following form:
  "*.WMF;*.BMP;*.TIF;*.JPG;*.PCX;*.TIF"

Example:

```
char s[256]
VpeGetPictureTypes(hDoc, s, sizeof(s))
```

See Also:
  "Pictures" in the Programmer's Manual

10.6 VpeSetPictureType

By default, VPE determines the type of an image that shall be imported or exported by its filename extension. If PictureType is other than PIC_TYPE_AUTO (the default = determining the file-type by its extension), the extension will be ignored and VPE is forced to use the specified file type.

If PIC_TYPE_AUTO is used for import, VPE is able to determine the image type not only by looking at the file suffix, but also by examining the file itself. So PIC_TYPE_AUTO will work in most cases even if the file suffix does not describe the image type. Example: you try to import a JPEG image with the name "test.001" and have set PictureType = PIC_TYPE_AUTO, even then VPE will detect that this is a JPEG file!

This property applies to image-file import and export.

void VpeSetPictureType(
  VpeHandle *hDoc*,
  long *type*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle (Template VPE Object only)

long type
  possible values are:

Constant Name	Value	Comment
PIC_TYPE_AUTO	255	Automatic determination by filename-suffix (default)
PIC_TYPE_BMP	0	Windows-Bitmap
PIC_TYPE_WMF	5	WMF
PIC_TYPE_EMF	6	EMF
PIC_TYPE_TIFF	64	TIFF-File
PIC_TYPE_GIF	65	GIF-File
PIC_TYPE_PCX	66	PCX-File
PIC_TYPE_JPEG	68	JPEG-File
PIC_TYPE_PNG	69	PNG (Portable Network Graphic)
PIC_TYPE_ICO	70	ICO (Windows Icon)
PIC_TYPE_JNG	71	JNG (JPEG Network Graphics)
PIC_TYPE_KOALA	72	KOA (C64 Koala Graphics)
PIC_TYPE_IFF	73	IFF/LBM (Interchangeable File Format - Amiga/Deluxe Paint)
PIC_TYPE_MNG	74	MNG (Multiple-Image Network Graphics)
PIC_TYPE_PBM	75	PBM (Portable Bitmap [ASCII])
PIC_TYPE_PBM_RAW	76	PBM (Portable Bitmap [RAW])
PIC_TYPE_PCD	77	PCD (Kodak PhotoCD)
PIC_TYPE_PGM	78	PGM (Portable Greymap [ASCII])
PIC_TYPE_PGM_RAW	79	PGM (Portable Greymap [RAW])

PIC_TYPE_PPM	80	PPM (Portable Pixelmap [ASCII])
PIC_TYPE_PPM_RAW	81	PPM (Portable Pixelmap [RAW])
PIC_TYPE_RAS	82	RAS (Sun Raster Image)
PIC_TYPE_TARGA	83	TGA/TARGA (Truevision Targa)
PIC_TYPE_WBMP	84	WAP/WBMP/WBM (Wireless Bitmap)
PIC_TYPE_PSD	85	PSD (Adobe Photoshop)
PIC_TYPE_CUT	86	CUT (Dr. Halo)
PIC_TYPE_XBM	87	XBM (X11 Bitmap Format)
PIC_TYPE_XPM	88	XPM (X11 Pixmap Format)
PIC_TYPE_DDS	89	DDS (DirectX Surface)
PIC_TYPE_HDR	90	HDR (High Dynamic Range Image)
PIC_TYPE_FAX_G3	91	G3 (Raw fax format CCITT G.3)
PIC_TYPE_SGI	92	SGI (SGI Image Format)

Default:

PIC_TYPE_AUTO

Remarks:

Setting this property for a VPE Object that resides in a document can have unpredictable results. In contrast you may set this property for a VPE Object that resides in a template without problems, if the file name contains at least one field.

Example:

```
VpeSetPictureType(hDoc, PIC_TYPE_TIFF)
VpePicture(hDoc, 1, 1, VFREE, VFREE, "image.001")
```

Will import "image.001" as TIFF file.

See Also:

"Pictures" in the Programmer's Manual

10.7 VpeGetPictureType

Returns the current picture type. See VpeSetPictureType for details.

long VpeGetPictureType(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:
the picture type

See Also:
"Pictures" in the Programmer's Manual

10.8 VpeSetPictureCache

[Not supported by the Community Edition]

Instructs VPE to move all images, which are subsequently added to the document with the methods VpePicture() and VpeRenderPicture(), into the dynamic image cache.

See "Image Cache" in the Programmer's Manual.

We recommend to set this property always to true.

void VpeSetPictureCache(
VpeHandle *hDoc*,
int *on_off*
)

VpeHandle hDoc
Document Handle

int on_off

Value	Description
True	On
False	Off

Default:
True

Community Edition:
The Community Edition does not provide caching of images.

See Also:
"Pictures" in the Programmer's Manual

10.9 VpeGetPictureCache

Returns the current setting for the picture cache.

int VpeGetPictureCache(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current setting for the picture cache

See Also:
    "Pictures" in the Programmer's Manual

10.10 VpeGetPicturePageCount

[Enhanced Edition and above]

Retrieves the number of available pages stored in the specified image file. At present only multipage TIFF and GIF files are supported.

long VpeGetPicturePageCount(
  VpeHandle *hDoc*,
  LPCSTR *file_name*
)

VpeHandle hDoc
  Document Handle

LPCSTR file_name
  image file to check for the number of available pages

Returns:
  the number of available pages stored in the specified image file

Remarks:
  In case of an error, LastError is set.

Example:

```
n = VpeGetPicturePageCount(hDoc, "test.tif")
```

See Also:
  "Pictures" in the Programmer's Manual

10.11 VpeSetPicturePage

[Enhanced Edition and above]

Sets the number of the page which will be loaded from a multipage image file by the next call to VpePicture(). The first page starts with 1.

void VpeSetPicturePage(
    VpeHandle *hDoc*,
    long *page_no*
)

VpeHandle hDoc
    Document Handle

long page_no
    the page number which shall be loaded from a multipage image file

Default:
    1

Example:

```
n = VpeGetPicturePageCount(hDoc, "test.tif")
for i = 1 to n
  VpeSetPicturePage(hDoc, i)
  Picture(hDoc, 0, 0, VFREE, VFREE, "test.tif")
  if i < n then
       PageBreak(hDoc)
  end if
next i
```

See Also:
    "Pictures" in the Programmer's Manual

10.12 VpeGetPicturePage

[Enhanced Edition and above]

Retrieves the number of the page which will be loaded by the next call to VpePicture(). The first page starts with 1.

long VpeGetPicturePage(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the page number which will be loaded by the next call to VpePicture()

See Also:
    "Pictures" in the Programmer's Manual

10.13 VpeSetPictureEmbedInDoc

When writing a VPE document to file (either by working with *SwapFileName*, or *WriteDoc*), this property enforces that the image will be stored directly in the VPE document file with all its binary data. If this property is set to false, an image is stored with its pathname link (which will need much less space: just a few bytes) in a VPE document. Setting this property to true is useful, if the location of the referenced image changes, or if you transport a VPE document file to another system, where the image is not available (for example by e-mail).

VPE embeds images highly optimized within document files:

- Images are only stored once within a document file, regardless how often they are used (for example on each page).
- Compressed images (TIF, GIF, JPG, PNG) are stored in their original form, i.e. the images are copied directly and unchanged from the hard disk into a VPE document.
- Uncompressed images will be flate compressed (using the ZLIB), if compression for the document file is activated (which is the default).

For details about creating and using VPE document files, please see the "Programmer's Manual", chapter "Programming Techniques", subchapter "VPE Document Files".

void VpeSetPictureEmbedInDoc(
VpeHandle *hDoc*,
int *on_off*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int on_off

Value	Description
True	on
False	off

Default:
True

Remarks:
ATTENTION: When you read one or more VPE document files into the current document using *ReadDoc()*, and those files have embedded images, you may NOT delete or modify the source document files, while the current document is open! This is, because VPE loads the embedded images directly out of the source document files each time they are not found in the image cache.

See Also:

"Pictures" in the Programmer's Manual

10.14 VpeGetPictureEmbedInDoc

Returns the current setting for the embedding of pictures.

int VpeGetPictureEmbedInDoc(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the current setting for the embedding of pictures

See Also:
    "Pictures" in the Programmer's Manual

10.15 VpeSetPictureKeepAspect

Setting this mode on/off determines whether a scaled picture shall be distorted or not, when the x OR y dimension is calculated automatically. This makes sense, if only ONE parameter of a Picture-Method is VFREE: x2 OR y2. If both are VFREE, the image is always undistorted.

void VpeSetPictureKeepAspect(
    VpeHandle *hDoc*,
    int *on_off*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

int on_off

Value	Description
True	on (pictures will be scaled undistorted)
False	Off

Default:
    True

Example:

```
VpeSetPictureKeepAspect(hDoc, TRUE)
VpePicture(hDoc, 1, 1, -5, VFREE, "image1.tif")
```

In the example, the width of the image will be 5cm. If the image had an original size of 15 x 21 cm, the width is now 1/3 of the original size. Because PictureKeepAspect is True, VPE will compute the height to the same aspect ratio = 1/3 of the original height, which would then be 7 cm.
Note: the above example assumes that the resolution of the image is the same for the width and the height. Otherwise the computations performed by VPE are slightly more difficult.

See Also:
    "Pictures" in the Programmer's Manual

10.16 VpeGetPictureKeepAspect

Returns the current setting of this property.

int VpeGetPictureKeepAspect(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the current setting of this property

See Also:
    "Pictures" in the Programmer's Manual

10.17 VpeSetPictureBestFit

Keeps a picture's aspect and allows for the largest zoom within a given rectangle without distortion. In contrast to *PictureKeepAspect* this flag is useful, if you specify all four coordinates.

See "Scaling" in the Programmer's Manual for details.

void VpeSetPictureBestFit(
VpeHandle *hDoc*,
int *on_off*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int on_off

Value	Description
True	on
False	off

Default:
False

See Also:
"Pictures" in the Programmer's Manual

10.18 VpeGetPictureBestFit

Returns the current setting of this property.

int VpeGetPictureBestFit(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:
the current setting of this property

See Also:
"Pictures" in the Programmer's Manual

10.19 VpeSetPictureDefaultDPI

Some bitmap-files do not contain the DPI resolution information in which they were originally created. For example, GIF images have no such information (you should use 96 by 96 DPI). Since VPE is a WYSIWYG-system, it needs this information for correct representation of the bitmap in metric coordinates. If the resolution information cannot be found in the image, VPE will use the default values you specify with this method.

Additionally, for the Professional Edition and above, *exported* bitmaps will have the resolution specified with this method.

void VpeSetPictureDefaultDPI(
VpeHandle *hDoc*,
int *dpix*,
int *dpiy*
)

VpeHandle hDoc
Document Handle

int dpix, dpiy
default x- and y-resolution in DPI

Default:
96 by 96 DPI which is the resolution of the screen

See Also:
"Pictures" in the Programmer's Manual

10.20 VpeSetPictureX2YResolution

Force the y-resolution of the image to the value of the x-resolution. Sometimes bitmaps have stored wrong resolution (DPI) information. With this flag you can instruct VPE to extract the x-resolution out of the image and to assume the y-resolution to be the same.

void VpeSetPictureX2Yresolution(
VpeHandle *hDoc*,
int *on_off*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int on_off

Value	Description
True	on
False	off

Default:
False

See Also:
"Pictures" in the Programmer's Manual

10.21 VpeGetPictureX2YResolution

Returns the current setting of this property.

int VpeGetPictureX2YResolution(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the current setting of this property

See Also:
    "Pictures" in the Programmer's Manual

10.22 VpeSetPictureDrawExact

Draws bitmaps exactly (and slow) to avoid seldom arising pixel-misalignment problems due to WYSIWYG coordinate-rounding problems, or if the graphics driver has bugs. For large bitmaps (for example full-page forms) it might happen that during scrolling a pixel line gets lost. By specifying this flag this can not happen. This flag is only useful for the preview, not for printing.

void VpeSetPictureDrawExact(
VpeHandle *hDoc*,
int *on_off*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int on_off

Value	Description
True	on
False	off

Default:
False

See Also:
"Pictures" in the Programmer's Manual

10.23 VpeGetPictureDrawExact

Returns the current setting of this property.

```
int VpeGetPictureDrawExact(
    VpeHandle hDoc
)
```

VpeHandle hDoc
  Document Handle or VPE Object Handle

Returns:
  the current setting of this property

See Also:
  "Pictures" in the Programmer's Manual

10.24 VpeSetPictureScale2Gray

[Professional Edition and above only]

Will perform a Scale-to-Gray of the provided bitmap for best readability at a 1:1 preview scale factor of the preview. The grayscale image is generated and than cached.

This property does not work for Metafiles and Enhanced Metafiles.

See "Scale-to-Gray Technology" in the Programmer's Manual for details.

void VpeSetPictureScale2Gray(
  VpeHandle *hDoc*,
  int *on_off*
)

VpeHandle hDoc
  Document Handle

int on_off

Value	Description
True	on
False	off

Default:
  False

See Also:
  "Pictures" in the Programmer's Manual

10.25 VpeGetPictureScale2Gray

Returns the current setting of this property.

int VpeGetPictureScale2Gray(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current setting of this property

See Also:
    "Pictures" in the Programmer's Manual

10.26 VpeSetPictureScale2GrayFloat

[Professional Edition and above only]

Will perform a Scale-To-Gray for every preview scaling, not only for a preview scaling of 1:1 as PictureScale2Gray does.

This property does not work for Metafiles, Enhanced Metafiles and DXF files.

See "Scale-to-Gray Technology" in the Programmer's Manual for details.

void VpeSetPictureScale2GrayFloat(
VpeHandle *hDoc*,
int *on_off*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int on_off

Value	Description
True	on
False	off

Default:
False

Remarks:
In contrast to *PictureScale2Gray* the ***unscaled*** image is cached and the grayscale image is always newly generated when displayed in a different scaling or on a new page, which needs more processor time.

See Also:
"Pictures" in the Programmer's Manual

10.27 VpeGetPictureScale2GrayFloat

Returns the current setting of this property.

```
int VpeGetPictureScale2GrayFloat(
    VpeHandle hDoc
)
```

VpeHandle hDoc

Document Handle or VPE Object Handle

Returns:

the current setting of this property

See Also:

"Pictures" in the Programmer's Manual

10.28 VpePicture

Inserts a picture object into the document. For supported file formats, please see the chapter "Pictures" in the Programmer's Manual.

VpeCoord VpePicture(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *file_name*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions
Dimensions: if x2 is VFREE, it is computed automatically; if y2 is VFREE, it is computed automatically. Computations are performed based on the size of the image in pixels and its resolution information in DPI (see also the properties PictureKeepAspect and PictureBestFit).

LPCSTR file_name
The name of the file that shall be imported. VPE determines the file-type by the suffix (i.e. ".TIF" = TIFF, etc.) or by analyzing the file header - a full path is optional. (see also: VpeSetPictureType)

Returns:
the bottom coordinate (y2) of the inserted image

Remarks:
In case of an error, LastError is set.

The frame drawn around images can be made invisible by setting the PenSize to zero before importing the image.

We recommend to use the methods VpePicture() and VpePictureStream() only.
This guarantees platform independence.

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

Example:

```
VpePicture(hDoc, 1, 1, VFREE, VFREE, "test.tif")
```

See Also:
"Pictures" in the Programmer's Manual

10.29 VpePictureStream

[Professional Edition and above]

Identical to VpePicture(), but imports a picture from a memory stream.

VpeCoord VpePictureStream(
VpeHandle *hDoc*,
VpeHandle *hStream*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCTSTR *identifier*
)

VpeHandle hDoc
Document Handle

VpeHandle hStream
The handle of the stream where the picture is imported from. The stream must have been created before by calling VpeCreateMemoryStream(), and of course it must have been initialized with valid image data.

VpeCoord x, y, x2, y2
position and dimensions

LPCTSTR identifier
A name for the picture. The internal image cache uses this name to distinguish images. But the image cache also computes a CRC (checksum) of the image data, so you can also leave the identifier blank (NOT NULL!). However, we do recommend to use a name if possible, so the CRC is not the only factor.

Returns:
the bottom coordinate (y2) of the inserted image

Remarks:
If you wish to use one and the same image multiple times, always provide the same stream handle (and therefore of course the same stream) as well as the same identifier, when calling this method.

See Also:
"Pictures" in the Programmer's Manual

10.30 VpePictureResID

[Windows platform only]

Loads the image from a resource by id. The image is taken from the resource of

- the calling application, if hInstance = 0; or
- a DLL, with hInstance = handle of the instance of the loaded DLL

VpeCoord VpePictureResID(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
int *hInstance*,
unsigned int *res_id*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions

int hInstance
Instance-Handle

unsigned int res_id
resource-id

Returns:
the bottom coordinate (y2) of the inserted image

Remarks:
From resources, VPE can only load BMP files, not JPEG, WMF, TIFF or any other file types.
If you create a VPE document file with images loaded from a resource, and the PIC_IN_FILE flag is NOT used, VPE stores the resource-link in document files. If you open such a 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 set automatically in the document files.

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

See Also:
"Pictures" in the Programmer's Manual

10.31 VpePictureResName

[Windows platform only]

The same as VpePictureResID(), but instead of a numeric ID the resource is identified by name. The image is taken from the resource of

- the calling application, if hInstance = 0; or
- a DLL, with hInstance = handle of the instance of the loaded DLL

VpeCoord VpePictureResName(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*,
  int *hInstance*,
  LPCSTR *res_name*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y, x2, y2
  position and dimensions

int hInstance
  Instance-Handle

LPCSTR res_name
  resource-name

Returns:
  the bottom coordinate (y2) of the inserted image

Remarks:
  From resources, VPE can only load BMP files, not JPEG, WMF, TIFF or any other file types.
  If you create a VPE document file with images loaded from a resource, and the PIC_IN_FILE flag is NOT used, VPE stores the resource-link in document files. If you open such a 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 set automatically in the document files.

  VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
  For details please see "Dynamic Positioning" in the Programmer's Manual.

See Also:
  "Pictures" in the Programmer's Manual

10.32 VpePictureDIB

[Windows platform only]

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 of 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, as they are managed much better by VPE with the PictureRes-functions (locking, unlocking, conversion, etc.).

VpeCoord VpePictureDIB(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
HGLOBAL *hDIB*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions

HGLOBAL hDIB
handle of DIB

Returns:
the bottom coordinate (y2) of the inserted image

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

VpePictureDIB() expects a handle for a memory block that was allocated with GlobalAlloc(). VpePictureDIB() creates a copy of the supplied memory block, so the calling application is responsible for freeing the DIB handle. BUT, if you supply an identical handle multiple times, VPE will not create additional copies, instead VPE will reference the initial copied memory. In other words: VPE creates only ONE copy of the supplied memory, for one and the same handle.

See Also:
"Pictures" in the Programmer's Manual

11 Barcode Functions (1D)

11.1 Barcode Functions

[Enhanced Edition and above only]

These methods and properties deal with the generation and presentation of the various one-dimensional barcodes offered by VPE.

See Also:

"Barcodes (1D)" in the Programmer's Manual

11.2 VpeSetBarcodeParms

[Enhanced Edition and above]

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

void VpeSetBarcodeParms(
VpeHandle *hDoc*,
int *top_bottom*,
int *add_top_bottom*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int top_bottom
assigned to the main barcode text; one of the BCP_xyz constants below

int add_top_bottom
assigned to the add-on barcode text; one of the BCP_xyz constants below

Default:
BCP_BOTTOM, BCP_BOTTOM

Possible values for top_bottom and add_top_bottom are:

Constant Name	Value	Comment
BCP_BOTTOM	0	Barcode Text at Bottom
BCP_TOP	1	Barcode Text at Top
BCP_HIDE	2	Barcode Text Hidden
BCP_DEFAULT	3	Barcode Text at default position appropriate to barcode type

See Also:
"Barcodes (1D)" in the Programmer's Manual

11.3 VpeSetBarcodeMainTextParms

[Enhanced Edition and above]

Sets the position of the main barcode label text.

int VpeSetBarcodeMainTextParms(
    VpeHandle *hDoc*,
    int *top_bottom*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

int top_bottom
    assigned to the main barcode text; one of the BCP_xyz constants below

Default:
    BCP_BOTTOM

Possible values for top_bottom are:

Constant Name	**Value**	**Comment**
BCP_BOTTOM	0	Barcode Text at Bottom
BCP_TOP	1	Barcode Text at Top
BCP_HIDE	2	Barcode Text Hidden

See Also:
    "Barcodes (1D)" in the Programmer's Manual

11.4 VpeGetBarcodeMainTextParms

[Enhanced Edition and above]

Returns the position of the main barcode label text.

int VpeGetBarcodeMainTextParms(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    Possible values are:

Constant Name	Value	Comment
BCP_BOTTOM	0	Barcode Text at Bottom
BCP_TOP	1	Barcode Text at Top
BCP_HIDE	2	Barcode Text Hidden

See Also:
    "Barcodes (1D)" in the Programmer's Manual

11.5 VpeSetBarcodeAddTextParms

[Enhanced Edition and above]

Sets the position of the add-on barcode label text.

int VpeSetBarcodeMainTextParms(
    VpeHandle *hDoc*,
    int *top_bottom*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

int top_bottom
    assigned to the add-on barcode text; one of the BCP_xyz constants below

Default:
    BCP_BOTTOM

Possible values for top_bottom are:

Constant Name	Value	Comment
BCP_BOTTOM	0	Barcode Text at Bottom
BCP_TOP	1	Barcode Text at Top
BCP_HIDE	2	Barcode Text Hidden

See Also:
    "Barcodes (1D)" in the Programmer's Manual

11.6 VpeGetBarcodeAddTextParms

[Enhanced Edition and above]

Returns the position of the add-on barcode label text.

int VpeGetBarcodeAddTextParms(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    Possible values are:

Constant Name	Value	Comment
BCP_BOTTOM	0	Barcode Text at Bottom
BCP_TOP	1	Barcode Text at Top
BCP_HIDE	2	Barcode Text Hidden

See Also:
    "Barcodes (1D)" in the Programmer's Manual

11.7 VpeSetBarcodeAlignment

[Enhanced Edition and above]

Sets the alignment of barcodes within the rectangle they are drawn.

void VpeSetBarcodeAlignment(
  VpeHandle *hDoc*,
  int *alignment*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

int alignment
  possible values are:

Constant Name	Value	Comment
ALIGN_LEFT	0	
ALIGN_RIGHT	1	
ALIGN_CENTER	2	

Default:
  ALIGN_LEFT

See Also:
  "Barcodes (1D)" in the Programmer's Manual

11.8 VpeGetBarcodeAlignment

[Enhanced Edition and above]

Returns the alignment of barcodes within the rectangle they are drawn.

int VpeGetBarcodeAlignment(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:
possible values are:

Constant Name	Value	Comment
ALIGN_LEFT	0	
ALIGN_RIGHT	1	
ALIGN_CENTER	2	

See Also:
"Barcodes (1D)" in the Programmer's Manual

11.9 VpeSetBarcodeAutoChecksum

[Enhanced Edition and above]

Specifies, whether an auto checksum for a barcode will be generated automatically or not.

void VpeSetBarcodeAutoChecksum(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int yes_no

Value	Description
True	yes, the auto checksum will be generated
False	no, the auto checksum is not generated

Default:
True

See Also:
"Barcodes (1D)" in the Programmer's Manual

11.10 VpeGetBarcodeAutoChecksum

[Enhanced Edition and above]

Returns, whether an auto checksum for a barcode will be generated automatically or not.

int VpeGetBarcodeAutoChecksum(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:

Value	Description
True	yes, the auto checksum will be generated
False	no, the auto checksum is not generated

See Also:
    "Barcodes (1D)" in the Programmer's Manual

11.11 VpeSetBarcodeThinBar

[Enhanced Edition and above]

Specifies the relative width for thin barcode modules (1D barcodes only).

A barcode module is a line or a gap. Barcodes consist of small and thick modules.

Using BarcodeThinBar and BarcodeThickBar, the ratio of thin to thick modules can be adjusted.

void VpeSetBarcodeThinBar(
VpeHandle *hDoc*,
int *relative_width*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int relative_width
the relative width of thin barcode modules, a value of zero will cause VPE to chose an optimal value in accordance to the available space for drawing the barcode

Default:
0

Remarks:
If you set this porperty as well as the ThickBar property to zero, the barcode library will chose a reasonable value.

Example:
To adjust a ratio of 1 : 2,5 between thin and thick modules, set

```
BarcodeThinBar = 2
```

and

```
BarcodeThickBar = 5
```

See Also:
"Barcodes (1D)" in the Programmer's Manual

11.12 VpeGetBarcodeThinBar

[Enhanced Edition and above]

Returns the relative width for thin barcode modules (1D barcodes only).

int VpeGetBarcodeThinBar(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

Returns:
  the relative width of thin barcode modules

See Also:
  "Barcodes (1D)" in the Programmer's Manual

11.13 VpeSetBarcodeThickBar

[Enhanced Edition and above]

Specifies the relative width for thick barcode modules (1D barcodes only).

A barcode module is a line or a gap. Barcodes consist of small and thick modules.

Using BarcodeThinBar and BarcodeThickBar, the ratio of thin to thick modules can be adjusted.

void VpeSetBarcodeThickBar(
VpeHandle *hDoc*,
int *relative_width*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

int relative_width
the relative width of thick barcode modules, a value of zero will cause VPE to chose an optimal value in accordance to the available space for drawing the barcode

Default:
0

Remarks:
If you set this porperty as well as the ThinBar property to zero, the barcode library will chose a reasonable value.

Example:
To adjust a ratio of 1 : 2,5 between thin and thick modules, set

```
BarcodeThinBar = 2
```

and

```
BarcodeThickBar = 5
```

See Also:
"Barcodes (1D)" in the Programmer's Manual

11.14 VpeGetBarcodeThickBar

[Enhanced Edition and above]

Returns the relative width for thick barcode modules (1D barcodes only).

int VpeGetBarcodeThickBar(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the relative width of thick barcode modules

See Also:
    "Barcodes (1D)" in the Programmer's Manual

11.15 VpeBarcode

[Enhanced Edition and above]

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 barcode text is set with VpeSetTextColor. The font used for the barcode labels can be specified with the property FontName, for example UPC-A and UPC-E require an OCR-B font.

void VpeBarcode(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
int *code_type*,
LPCSTR *code*,
LPCSTR *add_code*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions

int code_type
see below

LPCSTR code
string with the code (e.g. "123456")

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

the parameter "code_type" can be one of the following values:

Constant Name	Value	Comment
BCT_EAN13	1	Checkdigit required, OCR-B Font required
BCT_EAN13_2	10	Checkdigit required, OCR-B Font required
BCT_EAN13_5	11	Checkdigit required, OCR-B Font required
BCT_EAN8	2	Checkdigit required, OCR-B Font required
BCT_EAN8_2	12	Checkdigit required, OCR-B Font required

BCT_EAN8_5	13	Checkdigit required, OCR-B Font required
BCT_UPCA	3	Checkdigit required, OCR-B Font required
BCT_UPCA_2	14	Checkdigit required, OCR-B Font required
BCT_UPCA_5	15	Checkdigit required, OCR-B Font required
BCT_UPCE	9	first digit always '0', Checkdigit required,OCR-B Font required
BCT_UPCE_2	16	first digit always '0', Checkdigit required, OCR-B Font required
BCT_UPCE_5	17	first digit always '0', Checkdigit required, OCR-B Font required
BCT_EAN2	28	no Checkdigit
BCT_EAN5	29	no Checkdigit
BCT_CODABAR	5	Checkdigit optional, uThick / uThin
BCT_2OF5	7	2 of 5 Industrial (25ID), Checkdigit optional, uThick / uThin
BCT_INTERLEAVED2OF5	8	2 of 5 Interleaved (25IL), Checkdigit optional, uThick / uThin
BCT_CODE39	6	Code 3 of 9, Checkdigit optional, uThick / uThin
BCT_CODE39EXT	30	Code 3 of 9 Extended, Checkdigit optional, uThick / uThin
BCT_CODE93	21	Code 93, Checkdigit optional
BCT_CODE93EXT	31	Code 93 Extended, Checkdigit optional
BCT_POSTNET	22	uThick / uThin
BCT_CODE128	26	set of CODE 128A, B and C, Checkdigit required
BCT_EAN128	27	set of EAN 128A, B and C, Checkdigit required
BCT_RM4SCC	32	Royal Mail 4 State Customer Code, Checkdigit required
BCT_MSI	33	Checkdigit optional, uThick / uThin
BCT_ISBN	34	International Standard Book Number, Checkdigit required
BCT_ISBN_5	35	ISBN + EAN5 (for pricing), Checkdigit required
BCT_IDENTCODE	36	Identcode of the Deutsche Post AG, Checkdigit required
BCT_LEITCODE	37	Leitcode of the Deutsche Post AG, Checkdigit required
BCT_PZN	38	Pharma Zentral Code, Checkdigit required, uThick / uThin
BCT_CODE11	39	Code 11, Checkdigit optional, uThick / uThin
BCT_2OF5MATRIX	40	2 of 5 Matrix, Checkdigit optional, uThick / uThin
BCT_TELEPENA	41	Telepen-A, Checkdigit optional

Remarks:

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

See Also:

"Barcodes (1D)" in the Programmer's Manual

12 Barcode Functions (2D)

12.1 Barcode Functions (2D)

[Professional Edition and above only]

These methods and properties deal with the generation and presentation of the various two-dimensional barcodes offered by VPE.

See also:

"Barcodes (2D)" in the Programmer's Manual

12.2 VpeSetBar2DAlignment

[Professional Edition and above only]

Sets the horizontal and vertical alignment of the 2D-barcodes within the given rectangle.

void VpeSetBar2DAlignment(
VpeHandle *hDoc*,
int *alignment*
)

VpeHandle hDoc
Document Handle

int alignment
the alignment; possible values are:

Constant Name	Value	Comment
VBAR2D_ALIGN_CENTER	0	horizontally and vertically centered
VBAR2D_ALIGN_CENTER_H	0	horizontally centered
VBAR2D_ALIGN_CENTER_V	0	vertically centered
VBAR2D_ALIGN_LEFT	1	
VBAR2D_ALIGN_RIGHT	2	
VBAR2D_ALIGN_TOP	4	
VBAR2D_ALIGN_BOTTOM	8	

Default:
VBAR2D_ALIGN_CENTER

Remarks:
You combine values for horizontal and vertical alignment by adding two flags.

Example:

```
VpeSetBar2DAlignment(hDoc, VBAR2D_ALIGN_LEFT + VBAR2D_ALIGN_TOP)
```

Will align the barcode to the top-left corner.

See also:
"Barcodes (2D)" in the Programmer's Manual

12.3 VpeGetBar2DAlignment

[Professional Edition and above only]

Returns the current setting for the horizontal and vertical alignment of 2D-barcodes.

int VpeGetBar2DAlignment(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

See also:
  "Barcodes (2D)" in the Programmer's Manual

12.4 VpeSetDataMatrixEncodingFormat

[Professional Edition and above only]

Sets the encoding format ID for DataMatrix symbologies.

void VpeSetDataMatrixEncodingFormat(
    VpeHandle *hDoc*,
    int *format*
)

VpeHandle hDoc
    Document Handle

int format
    the format ID; possible values are:

Constant Name	Value	Comment
VBAR2D_DATA_MATRIX_ENC_BASE11	1	numeric data, 3.5 bits per digit
VBAR2D_DATA_MATRIX_ENC_BASE27	2	upper case alphabetic, 4.8 bits per character
VBAR2D_DATA_MATRIX_ENC_BASE41	3	upper case alphanumeric and punctuation, 5.5 bits per character
VBAR2D_DATA_MATRIX_ENC_BASE37	4	upper case alphanumeric, 5.25 bits per character
VBAR2D_DATA_MATRIX_ENC_ASCII	5	128 ASCII set, 7 bits per character
VBAR2D_DATA_MATRIX_ENC_BINARY	6	8-bit Byte, 8 bits per character

Default:
    VBAR2D_DATA_MATRIX_ENC_BASE11

Remarks:
    For ECC200 this is the start-up code page, while for ECC000 - ECC140 the selected format covers the entire message.

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.5 VpeGetDataMatrixEncodingFormat

[Professional Edition and above only]

Returns the current encoding format ID for DataMatrix symbologies.

int VpeGetDataMatrixEncodingFormat(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.6 VpeSetDataMatrixEccType

[Professional Edition and above only]

Sets the error checking and correction scheme for DataMatrix symbologies.

void VpeSetDataMatrixEccType(
VpeHandle *hDoc*,
int *ecc_type*
)

VpeHandle hDoc
Document Handle

int ecc_type
the error correction scheme; possible values are:

Constant Name	Value	Comment
VBAR2D_DATA_MATRIX_ECC000	0	
VBAR2D_DATA_MATRIX_ECC010	1	
VBAR2D_DATA_MATRIX_ECC040	2	
VBAR2D_DATA_MATRIX_ECC050	3	
VBAR2D_DATA_MATRIX_ECC060	4	
VBAR2D_DATA_MATRIX_ECC070	5	
VBAR2D_DATA_MATRIX_ECC080	6	
VBAR2D_DATA_MATRIX_ECC090	7	
VBAR2D_DATA_MATRIX_ECC100	8	
VBAR2D_DATA_MATRIX_ECC110	9	
VBAR2D_DATA_MATRIX_ECC120	10	
VBAR2D_DATA_MATRIX_ECC130	11	
VBAR2D_DATA_MATRIX_ECC140	12	
VBAR2D_DATA_MATRIX_ECC200	26	

Default:
VBAR2D_DATA_MATRIX_ECC200

See also:
"Barcodes (2D)" in the Programmer's Manual

12.7 VpeGetDataMatrixEccType

[Professional Edition and above only]

Returns the current error checking and correction scheme for DataMatrix symbologies.

int VpeGetDataMatrixEccType(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.8 VpeSetDataMatrixRows

[Professional Edition and above only]

Sets the desired number of matrix rows (0: choose automatically based on message).

void VpeSetDataMatrixRows(
  VpeHandle *hDoc*,
  int *rows*
)

VpeHandle hDoc
  Document Handle

int rows
  Desired number of matrix rows (0: choose automatically based on message).

Default:
  0

Remarks:
  Note: The number of columns and rows heavily depends on the ECC type that has been selected. For example, a valid ECC100 symbol spans at least 13x13 modules and at most 49x49, where only odd combinations are allowed (i.e. 13x13, 15x15, 17x17, .. 47x47, 49x49).

See also:
  "Barcodes (2D)" in the Programmer's Manual

12.9 VpeGetDataMatrixRows

[Professional Edition and above only]

Returns the current setting for the desired number of matrix rows (0: choose automatically based on message).

int VpeGetDataMatrixRows(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.10 VpeSetDataMatrixColumns

[Professional Edition and above only]

Sets the desired number of matrix columns (0: choose automatically based on message).

void VpeSetDataMatrixColumns(
VpeHandle *hDoc*,
int *columns*
)

VpeHandle hDoc
Document Handle

int columns
Desired number of matrix columns (0: choose automatically based on message).

Default:
0

Remarks:
Note: The number of columns and rows heavily depends on the ECC type that has been selected. For example, a valid ECC100 symbol spans at least 13x13 modules and at most 49x49, where only odd combinations are allowed (i.e. 13x13, 15x15, 17x17, .. 47x47, 49x49).

See also:
"Barcodes (2D)" in the Programmer's Manual

12.11 VpeGetDataMatrixColumns

[Professional Edition and above only]

Returns the current setting for the desired number of matrix columns (0: choose automatically based on message).

int VpeGetDataMatrixColumns(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.12 VpeSetDataMatrixMirror

[Professional Edition and above only]

Specifies, whether a mirrored version of the barcode shall be created (applies to ECC000 – ECC140 only).

void VpeSetDataMatrixMirror(
VpeHandle *hDoc*,
int *mirror*
)

VpeHandle hDoc
Document Handle

int mirror
possible values are:

Value	Description
0	do not mirror
1	mirror

Default:
0

See also:
"Barcodes (2D)" in the Programmer's Manual

12.13 VpeGetDataMatrixMirror

[Professional Edition and above only]

Returns the current setting of the mirror property.

int VpeGetDataMatrixMirror(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.14 VpeSetDataMatrixBorder

[Professional Edition and above only]

Sets the number of modules used for the border (usually 1).

void VpeSetDataMatrixBorder(
    VpeHandle *hDoc*,
    int *border*
)

VpeHandle hDoc
    Document Handle

int border
    the number of modules used for the border (usually 1)

Default:
    1

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.15 VpeGetDataMatrixBorder

[Professional Edition and above only]

Returns the current setting of the border property.

int VpeGetDataMatrixBorder(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.16 VpeDataMatrix

[Professional Edition and above only]

Inserts a DataMatrix barcode into the VPE document.

void VpeDataMatrix(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCTSTR *lpszText*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
coordinates

LPCTSTR lpszText
the text of the barcode

Example:

```
VpeDataMatrix(hDoc, 1, 1, -3, -3, "30Q324343430794<OQQ")
```

Remarks:
sets LastError

See also:
"Barcodes (2D)" in the Programmer's Manual

12.17 VpeRenderDataMatrix

[Professional Edition and above only]

Computes the dimensions of a DataMatrix barcode.

void VpeRenderDataMatrix(
VpeHandle *hDoc*,
LPCTSTR *lpszText*,
VpeCoord *nWidth*,
VpeCoord *nHeight*,
VpeCoord *nModuleWidth*
)

VpeHandle hDoc
Document Handle

LPCTSTR lpszText
the text of the barcode

VpeCoord nWidth, nHeight
With these two parameters you can specify a maximum size the barcode can have.

The following rules apply:

- nWidth and nHeight is specified: in this case the maximum rectangle of the barcode will be computed, which will fit into the given rectangle.
- Only nWidth is specified, nHeight is zero: In this case nHeight is computed (nWidth is also computed, i.e. adjusted).
- Only nHeight is specified, nWidth is zero: In this case nWidth is computed (nHeight is also computed, i.e. adjusted).
- nWidth and nHeight are zero: in this case the smallest possible rectangle is computed

VpeCoord nModuleWidth
If this parameter is zero, VPE will chose itself an optimum barcode module width. If it is non-zero, you can specify the width of a barcode module.

Example:

```
VpeRenderDataMatrix(hDoc, "30Q324343430794<OQQ", 1, 1, 0)
xsize = VpeGet(hDoc, VRENDERWIDTH)
ysize = VpeGet(hDoc, VRENDERHEIGHT)
```

Remarks:

sets LastError

See also:

"Barcodes (2D)" in the Programmer's Manual

12.18 VpeMaxiCode

[Professional Edition and above only]

Inserts a MaxiCode barcode into the VPE document.

This version accepts mode 2 and 3 messages beginning with “[)>$^{R}{}_{S}$01$^{G}{}_{S}$“, which conform to particular open system standards (for details refer to AIM specification). It selects the appropriate mode based on the ZIP code included in lpszText.

void VpeMaxiCode(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCTSTR *lpszText*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
coordinates

LPCTSTR lpszText
the text of the barcode

Example:

```
// Sample Message (in C++ syntax):
char *szUPS =    "[)>" "\x1e" "01" "\x1d" "9641460"
   "\x1d" "276" "\x1d" "068" "\x1d" "1Z51146547"
   "\x1d" "UPSN" "\x1d" "32630V" "\x1d" "327" "\x1d"
   "\x1d" "1/1" "\x1d" "2" "\x1d" "N" "\x1d"
   "\x1d" "NEUSS" "\x1d" "\x1e" "\x04"
   "!!!!!!!!!!!!!!!!!!!!!!!!!!"
VpeMaxiCode(hDoc, 1, 6, -3, -3, szUPS)
```

Remarks:
sets LastError

See also:
"Barcodes (2D)" in the Programmer's Manual

12.19 VpeRenderMaxiCode

[Professional Edition and above only]

Computes the dimensions of a MaxiCode barcode.

void VpeRenderMaxiCode(
VpeHandle *hDoc*,
LPCTSTR *lpszText*
)

VpeHandle hDoc
Document Handle

LPCTSTR lpszText
the text of the barcode

Remarks:

The width, height and module width of the MaxiCode barcode are fixed. You can not specify these parameters.

Sets LastError.

Example:

```
// Sample Message (in C++ syntax):
char *szUPS =    "[)>" "\x1e" "01" "\x1d" "9641460"
   "\x1d" "276" "\x1d" "068" "\x1d" "1Z51146547"
   "\x1d" "UPSN" "\x1d" "32630V" "\x1d" "327" "\x1d"
   "\x1d" "1/1" "\x1d" "2" "\x1d" "N" "\x1d"
   "\x1d" "NEUSS" "\x1d" "\x1e" "\x04"
   "!!!!!!!!!!!!!!!!!!!!!!!!!!"
VpeRenderMaxiCode(hDoc, szUPS)
xsize = VpeGet(hDoc, VRENDERWIDTH)
ysize = VpeGet(hDoc, VRENDERHEIGHT)
```

See also:

"Barcodes (2D)" in the Programmer's Manual

12.20 VpeMaxiCodeEx

[Professional Edition and above only]

Inserts a MaxiCode barcode into the VPE document. It is recommended to use VpeMaxiCode() instead. This function has been implemented for completeness.

void VpeMaxiCodeEx(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
int *nMode*,
LPCTSTR *lpszZip*,
int *nCountryCode*,
int *nServiceClass*,
LPCTSTR *lpszMessage*,
int *nSymbolNumber*,
int *nSymbolCount*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
coordinates

int nMode
Modes define the structuring of data and error correction within a symbol.

Value	Description
0	Obsolete
1	Obsolete
2	Structured Carrier Message. Encode the destination address and class of service in the transport industry. Mode 2 supports up to nine numeric digits for the ZIP code.
3	Similar to mode 2. Supports up to six alphanumeric characters for ZIP code.
4	Standard Symbol (for details refer to AIM specification)
5	EEC (for details refer to AIM specification)
6	Reader Programming (for details refer to AIM specification)

LPCTSTR lpszZip
Postal code, for example D40211 or 42119

int nCountryCode
ISO 3166 country code (for example 276 for Germany)

int nServiceClass
Carrier-dependent service class

LPCTSTR lpszMessage
Secondary message

int nSymbolNumber
Total number of symbols (structured append)

int nSymbolCount
Current number of symbol (structured append)

Example:

```
VpeMaxiCodeEx(hDoc, 1, 11, -5, -5, 2, "068107317", 840, 001, "AOE This is
              MaxiCode AOE", 1, 1)
```

Remarks:
sets LastError

See also:
"Barcodes (2D)" in the Programmer's Manual

12.21 VpeRenderMaxiCodeEx

[Professional Edition and above only]

Computes the dimensions of a MaxiCode barcode.

void VpeRenderMaxiCodeEx(
   VpeHandle *hDoc*,
   int *nMode*,
   LPCTSTR *lpszZip*,
   int *nCountryCode*,
   int *nServiceClass*,
   LPCTSTR *lpszMessage*,
   int *nSymbolNumber*,
   int *nSymbolCount*
)

VpeHandle hDoc
   Document Handle

int nMode
   Modes define the structuring of data and error correction within a symbol.

Value	**Description**
0	Obsolete
1	Obsolete
2	Structured Carrier Message. Encode the destination address and class of service in the transport industry. Mode 2 supports up to nine numeric digits for the ZIP code.
3	Similar to mode 2. Supports up to six alphanumeric characters for ZIP code.
4	Standard Symbol (for details refer to AIM specification)
5	EEC (for details refer to AIM specification)
6	Reader Programming (for details refer to AIM specification)

LPCTSTR lpszZip
   Postal code, for example D40211 or 42119

int nCountryCode
   ISO 3166 country code (for example 276 for Germany)

int nServiceClass
   Carrier-dependent service class

LPCTSTR lpszMessage
   Secondary message

int nSymbolNumber
   Total number of symbols (structured append)

int nSymbolCount
   Current number of symbol (structured append)

Remarks:
   The width, height and module width of the MaxiCode barcode are fixed. You can not specify these parameters.
   Sets LastError.

Example:

```
VpeMaxiCodeEx(hDoc, 1, 11, -5, -5, 2, "068107317", 840, 001,"AOE This is
              MaxiCode AOE", 1, 1)
xsize = VpeGet(hDoc, VRENDERWIDTH)
ysize = VpeGet(hDoc, VRENDERHEIGHT)
```

See also:

"Barcodes (2D)" in the Programmer's Manual

12.22 VpeSetPDF417ErrorLevel

[Professional Edition and above only]

Sets the error level for PDF417 barcodes.

void VpeSetPDF417ErrorLevel(
  VpeHandle *hDoc*,
  int *level*
)

VpeHandle hDoc
  Document Handle

int level
  The higher the level, the more redundancy is added to the symbol allowing more errors to be corrected at the expense of less user data capacity. If the level is -1, an error level is chosen automatically based on the message length as recommended by the AIM specification.

Default:
  -1

See also:
  "Barcodes (2D)" in the Programmer's Manual

12.23 VpeGetPDF417ErrorLevel

[Professional Edition and above only]

Returns the current setting of the error-level property.

int VpeGetPDF417ErrorLevel(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.24 VpeSetPDF417Rows

[Professional Edition and above only]

Sets the desired number of matrix rows (0: choose automatically based on message).

void VpeSetPDF417Rows(
    VpeHandle *hDoc*,
    int *rows*
)

VpeHandle hDoc
    Document Handle

int rows
    Desired number of matrix rows (0: choose automatically based on message).

Default:
    0

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.25 VpeGetPDF417Rows

[Professional Edition and above only]

Returns the current setting for the desired number of matrix rows (0: choose automatically based on message).

int VpeGetPDF417Rows(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.26 VpeSetPDF417Columns

[Professional Edition and above only]

Sets the desired number of matrix columns (0: choose automatically based on message).

void VpeSetPDF417Columns(
    VpeHandle *hDoc*,
    int *columns*
)

VpeHandle hDoc
    Document Handle

int columns
    Desired number of matrix columns (0: choose automatically based on message).

Default:
    0

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.27 VpeGetPDF417Columns

[Professional Edition and above only]

Returns the current setting for the desired number of matrix columns (0: choose automatically based on message).

int VpeGetPDF417Columns(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.28 VpePDF417

[Professional Edition and above only]

Inserts a PDF417 barcode into the VPE document.

void VpePDF417(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCTSTR *lpszText*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
coordinates

LPCTSTR lpszText
the text of the barcode

Example:

```
szStr ="01\t05\t{)>\t82\tWSP3.5.1DE\r"
  "02\t23112001\t1Z32630V6851146547\t08\tM\t9838571153\t3\t1.5\tKgs\t1.5\t\t
  P/P\t0.00\t\t\t\t\tDE\t\t\t\r"
  "04\tSH\tX-LOGISTIK GMBH\tWALLENHORST\t\t49134\tDE\t32630V\tC/O INTERTRADE
  AG\t\t\t\tHERR ERNST LEMKE\t+495407-834343\t\tAM OHLENBERG 14\t\r"
  "04\tST\tIDEAL SOFTWARE GMBH\tNEUSS\t\t41464\tDE\t\tERFTSTR.
  102A\t\t\t\tMR. XXX\t\t\t\t\r"
   "99\r";
VpePDF417(hDoc, 1, 17, -5.5, -3.5, szStr);
```

Remarks:
sets LastError

See also:
"Barcodes (2D)" in the Programmer's Manual

12.29 VpeRenderPDF417

[Professional Edition and above only]

Computes the dimensions of a PDF417 barcode.

void VpeRenderPDF417(
VpeHandle *hDoc*,
LPCTSTR *lpszText*,
VpeCoord *nWidth*,
VpeCoord *nHeight*,
VpeCoord *nModuleWidth*
)

VpeHandle hDoc
Document Handle

LPCTSTR lpszText
the text of the barcode

VpeCoord nWidth, nHeight
With these two parameters you can specify a maximum size the barcode can have.

The following rules apply:

- nWidth and nHeight is specified: in this case the maximum rectangle of the barcode will be computed, which will fit into the given rectangle.
- Only nWidth is specified, nHeight is zero: In this case nHeight is computed (nWidth is also computed, i.e. adjusted).
- Only nHeight is specified, nWidth is zero: In this case nWidth is computed (nHeight is also computed, i.e. adjusted).
- nWidth and nHeight are zero: in this case the smallest possible rectangle is computed

VpeCoord nModuleWidth
If this parameter is zero, VPE will chose itself an optimum barcode module width. If it is non-zero, you can specify the width of a barcode module.

Example:

```
szStr ="01\t05\t{)>\t82\tWSP3.5.1DE\r"
  "02\t23112001\t1Z32630V6851146547\t08\tM\t9838571153\t3\t1.5\tKgs\t1.5\t\t
  P/P\t0.00\t\t\t\t\tDE\t\t\t\r"
  "04\tSH\tX-LOGISTIK GMBH\tWALLENHORST\t\t49134\tDE\t32630V\tC/O INTERTRADE
  AG\t\t\t\tHERR ERNST LEMKE\t+495407-834343\t\tAM OHLENBERG 14\t\r"
  "04\tST\tIDEAL SOFTWARE GMBH\tNEUSS\t\t41464\tDE\t\tERFTSTR.
  102A\t\t\t\tMR. XXX\t\t\t\t\r"
  "99\r";
VpeRenderPDF417(hDoc, szStr, 0, 0, 0.05)
xsize = VpeGet(hDoc, VRENDERWIDTH)
ysize = VpeGet(hDoc, VRENDERHEIGHT)
```

Remarks:

sets LastError

See also:

"Barcodes (2D)" in the Programmer's Manual

12.30 VpeSetAztecFlags

[Professional Edition and above only]

Sets the flag for the interpretation of the lpszString parameter of the VpeAztec()-function.

void VpeSetAztecFlags(
    VpeHandle *hDoc*,
    int *flags*
)

VpeHandle hDoc
    Document Handle

int flags
    If true, the lpszString parameter of the VpeAztec()-function uses "<Esc>n" for FLG(n), and "<Esc><Esc>" for "<Esc>"

Default:
    0 (false)

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.31 VpeGetAztecFlags

[Professional Edition and above only]

Returns the current setting of the flags-property.

int VpeGetAztecFlags(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.32 VpeSetAztecControl

[Professional Edition and above only]

Sets the flag for the interpretation of the lpszString parameter of the VpeAztec()-function.

void VpeSetAztecControl(
    VpeHandle *hDoc*,
    int *control*
)

VpeHandle hDoc
    Document Handle

int control

Value	Description
0	default error correction level
01 to 99	minimum error correction percentage
101 to 104	1 to 4-layer Compact symbol
201 to 232	1 to 32-layer Full-Range symbol
300	a simple Aztec "Rune"

Default:
    0

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.33 VpeGetAztecControl

[Professional Edition and above only]

Returns the current setting of the control-property.

int VpeGetAztecControl(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.34 VpeSetAztecMenu

[Professional Edition and above only]

Specifies, if this is a "Menu" symbol.

void VpeSetAztecMenu(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no
specifies, if this is a "Menu" symbol

Default:
0 (false)

See also:
"Barcodes (2D)" in the Programmer's Manual

12.35 VpeGetAztecMenu

[Professional Edition and above only]

Returns the current setting of the menu-property.

int VpeGetAztecMenu(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.36 VpeSetAztecMultipleSymbols

[Professional Edition and above only]

Specifies # of symbols for message… 1 (normal) to 26.

void VpeSetAztecMultipleSymbols(
    VpeHandle *hDoc*,
    int *count*
)

VpeHandle hDoc
    Document Handle

int count
    # of symbols for message… 1 (normal) to 26

Default:
    1

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.37 VpeGetAztecMultipleSymbols

[Professional Edition and above only]

Returns the current setting of the MultipleSymbols-property.

int VpeGetAztecMenu(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "Barcodes (2D)" in the Programmer's Manual

12.38 VpeSetAztecID

[Professional Edition and above only]

Specifies an optional null-terminated ID field for append.

void VpeSetAztecID(
  VpeHandle *hDoc*,
  LPCTSTR *id*
)

VpeHandle hDoc
  Document Handle

LPCTSTR id
  optional null-terminated ID field for append

Default:
  NULL

See also:
  "Barcodes (2D)" in the Programmer's Manual

12.39 VpeAztec

[Professional Edition and above only]

Inserts an Aztec barcode into the VPE document.

void VpeAztec(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCTSTR *lpszText*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
coordinates

LPCTSTR lpszText
the text of the barcode

Example:

```
VpeAztec(hDoc, 1, 23, -3, -3, "This is a message")
```

Remarks:
sets LastError

See also:
"Barcodes (2D)" in the Programmer's Manual

12.40 VpeRenderAztec

[Professional Edition and above only]

Computes the dimensions of an Aztec barcode.

void VpeRenderAztec(
    VpeHandle *hDoc*,
    LPCTSTR *lpszText*,
    VpeCoord *nWidth*,
    VpeCoord *nHeight*,
    VpeCoord *nModuleWidth*
)

VpeHandle hDoc
    Document Handle

LPCTSTR lpszText
    the text of the barcode

VpeCoord nWidth, nHeight
    With these two parameters you can specify a maximum size the barcode can have.

    The following rules apply:

- nWidth and nHeight is specified: in this case the maximum rectangle of the barcode will be computed, which will fit into the given rectangle.
- Only nWidth is specified, nHeight is zero: In this case nHeight is computed (nWidth is also computed, i.e. adjusted).
- Only nHeight is specified, nWidth is zero: In this case nWidth is computed (nHeight is also computed, i.e. adjusted).
- nWidth and nHeight are zero: in this case the smallest possible rectangle is computed

VpeCoord nModuleWidth
    If this parameter is zero, VPE will chose itself an optimum barcode module width. If it is non-zero, you can specify the width of a barcode module.

Example:

```
VpeRenderAztec(hDoc, "This is a message", 0, 0, 0.05)
xsize = VpeGet(hDoc, VRENDERWIDTH)
ysize = VpeGet(hDoc, VRENDERHEIGHT)
```

Remarks:
    sets LastError

See also:
    "Barcodes (2D)" in the Programmer's Manual

13 E-Mail Functions

13.1 E-Mail Functions

[Windows platform only; not supported by the Community Edition]

VPE allows to send e-mails with VPE Documents via MAPI. Additionally, the e-mail feature allows very easily to fax VPE Documents directly with MS-FAX and MS-MAIL through MAPI.

E-mailing and faxing can be done with user interaction (mail dialog is shown) or without user interaction - only by code. You can specify by code:

- the sender
- a **list** of receivers (also CC and BCC)
- the message subject
- the message text
- a **list** of attachments (including or excluding VPE Documents)

Even if mailing is done with user interaction (mail dialog is shown), the properties you had set by code are valid and will be shown in the mail dialog.

VPE includes VPE View (see "VPE View - the Document Viewer" in the Programmer's Manual) a royalty-free **Document Viewer** for VPE Documents, which is very useful if documents are sent by e-mail, so the recipient can view the documents with a single mouse click. VPE View is also required if VPE Documents are faxed by using VPE's e-mail functions.

13.2 VpeIsMAPIInstalled

[Windows platform only; not supported by the Community Edition]

Checks the machine if MAPI is installed.

int VpeIsMAPIInstalled(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Constant Name	**Value**	**Comment**
VMAPI_NOT_INSTALLED	0	MAPI is not installed
VMAPI_INSTALLED	1	MAPI is installed
VMAPI_UNSURE	2	unsure (obsolete, was for 16-bit version)

Remarks:

In case of an error, LastError is set.

There are some problems with MAPI. Some vendors of MAPI clients (e-mail software) do not set up the registry correctly. The problems are sometimes very heavy and difficult to solve. In the following we list the problems we heared about:

32-bit VPE on Win-32 (9x / ME, NT >= 4.0 / 2000 / XP) platform
MAPI clients are not correctly registered. The guideline is, that the following entry must be present in the registry for all 32-bit platforms listed above:

HKEY_LOCAL_MACHINE \ SOFTWARE \ Microsoft \ Windows Messaging Subsystem \ MAPI = 1

But for NT <= v4.00 there has to be the following entry in the WIN.INI in the section [Mail]:

```
[Mail]

MAPI = 1
```

VPE checks for all conditions above, but some MAPI clients do not setup the registry as they should.

To cure the problem listed above, we implemented the following solution:
VPE checks for the guideline rules first and returns VMAPI_INSTALLED if they are matched. Otherwise VPE will check for a WIN.INI entry (yes!) which is made by some clients in the [Mail] section:

```
[Mail]

MAPIX = 1
```

If this entry is found, VPE will return VMAPI_UNSURE, because it is not 100% sure that a 32-bit MAPI client is really installed. Otherwise VPE will return VMAPI_NOT_INSTALLED. If you are sure there is a working MAPI client on the machine, but the registry key is missing, you can add it manually or let this perform by your setup software.

The e-mail toolbar button and the method MailDoc()
The e-mail button will be enabled for the condition VMAPI_INSTALLED and will be disabled (grayed) for the conditions VMAPI_UNSURE and VMAPI_NOT_INSTALLED.

You have the chance to re-enable the e-mail button after calling OpenDoc() , if you are sure a working MAPI client is installed (see VpeEnableMailButton).

The method MailDoc() will work for all conditons (i.e. it is not blocked), but for VMAPI_NOT_INSTALLED and VMAPI_UNSURE you call it on your own risc.

13.3 VpeSetMailSender

[Windows platform only; not supported by the Community Edition]

Allows you to specify the message sender by code.

void VpeSetMailSender(
VpeHandle *hDoc*,
LPCSTR *sender*
)

VpeHandle hDoc
Document Handle

LPCSTR sender
the name / e-mail address of the sender

Remarks:
Normally it is not necessary to set this property, because the MAPI client will use the default profile, or the user will be able to select his profile.

Example:

```
VpeSetMailSender(hDoc, "Support@IdealSoftware.com")
```

13.4 VpeAddMailReceiver

[Windows platform only; not supported by the Community Edition]

Allows you to specify a list of e-mail receivers by code.

void VpeAddMailReceiver(
VpeHandle *hDoc*,
LPCSTR *receiver*,
long *recip_class*
)

VpeHandle hDoc
Document Handle

LPCSTR receiver
the name / e-mail address of the receiver

long recip_class
posssible values are:

Constant Name	Value	Comment
VMAIL_ORIG	0	Recipient is message originator
VMAIL_TO	1	Recipient is a primary recipient
VMAIL_CC	2	Recipient is a copy recipient
VMAIL_BCC	3	Recipient is blind copy recipient
VMAIL_RESOLVE_NAME	-2147483648	try to resolve the name from the mail client's address book

Remarks:

The list of recipients will be valid until VpeClearMailReceivers() is called.

Due to the security features in Outlook 2002 or higher (which can also be installed for earlier Outlook versions as security fix), Outlook will show a warning message that an external application is trying to access the address book, when sending an e-mail. To avoid this, VPE does not call MAPIResolveName() anymore. If you want VPE to look up a name in the mail client's address book, add VMAIL_RESOLVE_NAME to the *RecipientClass* parameter.

```
VpeAddMailReceiver(hDoc, "IDEAL Software", VMAIL_TO + VMAIL_RESOLVE_NAME)
```

Will resolve the receiver to "Support@IdealSoftware.com" if "IDEAL Software" is stored in the address book of the MAPI client as "Support@IdealSoftware.com".
However, if you use this flag - with Outlook 2002 or higher or the security fix installed - this will cause a warning message to appear that an extearnal application is trying to access the address book.

The warning is also shown by Outlook - and can not be avoided - if you sent e-mails without showing a dialog (i.e. MailWithDialog = False).

The warning can be disabled in Outlook with "Options | Security" by unchecking "Show warning if other applications try to send an e-mail under my name".

If you are not using VMAIL_RESOLVE_NAME and you are using Outlook / Exchange, make sure SMTP is properly configured, otherwise an error message might appear that the message could not be delivered.

Example:

```
VpeAddMailReceiver(hDoc, "max@alpha.com", VMAIL_CC)
```

Will add "max@alpha.com" as carbon copy recipient for the same e-mail.

```
VpeAddMailReceiver(hDoc, "[FAX: +49 1234 12345678]", VMAIL_TO)
```

Will **FAX** the message to the given phone number! If a VPE Document is attached to the mail, the VISIBLE CONTENT of the document will be faxed. Requires MS-MAIL and MS-FAX, or a similar MAPI- Mail / Fax software to be installed on the system. Also the VPE Document viewer VPE View (see "VPE View - the Document Viewer" in the Programmer's Manual) needs to be correctly installed, otherwise VPE Document attachments will not be faxed with their visual content.

13.5 VpeClearMailReceivers

[Windows platform only; not supported by the Community Edition]

Clears the list of mail recipients created with VpeAddMailReceiver().

void VpeClearMailReceivers(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

13.6 VpeAddMailAttachment

[Windows platform only; not supported by the Community Edition]

Allows you to specify a list of e-mail attachments (files) by code.

void VpeAddMailAttachment(
VpeHandle *hDoc*,
LPCSTR *path*,
LPCSTR *file_name*
)

VpeHandle hDoc
Document Handle

LPCSTR path
The fully qualified path and file name of the attachment file. This path should include the disk drive letter and directory name. The attachment file should be closed before this call is made.

When VpeMailDoc() is called, VPE will examine all attachment paths. If the last character of a path is a backslash ("\"), VPE will write the current document as a temporary file to the specified path and add it as attachment to the mail.

If the examined path is NULL or empty (""), VPE will create and attach the temporary document in the temporary directory specified by the environment variable named TMP or TEMP, or - if both are not set - in the current working directory.

Any temporarily created file will be deleted before the VpeMailDoc() function returns.

Otherwise - if a fully qualified path and file name was specified - VPE expects that the attachment file has already been created. This gives you the **possibility to mail any kind of attachment**, not being limited to VPE-Documents.

LPCSTR file_name
The file name seen by the recipient. This name can differ from the filename in *path* if temporary files are being used. If *file_name* is NULL or empty (""), the file name from *path* is used. If the attachment is an OLE object, *file_name* contains the class name of the object, such as "Microsoft Excel Worksheet."

Remarks:

By default, the current document is automatically set as an attachment located in the tmp-dir (the file will only be created if - and in the moment when - MailDoc() is executed). The file name of the attachment is either the Application Name or - if VpeOpenDocFile() is used - the document file name. To clear that default, call VpeClearMailAttachments() before specifying your own attachments.

The property MailAutoAttachDocType controls, whether a VPE Document or a PDF Document is attached to the e-mail.

Examples:

```
VpeClearMailAttachments(hDoc)
```

Deletes the default attachment, which is the VPE Document itself.

```
VpeAddMailAttachment(hDoc, "c:\important_file.zip", NULL)
```

Sets "c:\important_file.zip" as the ONLY attachment. The VPE Document will not be sent, because VpeClearMailAttachments() deleted the default attachment of the VPE Document.

```
VpeAddMailAttachment(hDoc, "c:\important_file.txt", NULL)
```

Adds "c:\important_file.txt" as second attachment.

```
VpeWriteDoc(hDoc, "c:\tmp\doc.vpe")
VpeAddMailAttachment(hDoc, "c:\tmp\doc.vpe", NULL)
```

Adds "c:\tmp\doc.vpe" as third attachment.

13.7 VpeSetMailAutoAttachDocType

[Windows platform only; not supported by the Community Edition]

The value of this property controls, what type of document is automatically attached to e-mails (see also VpeAddMailAttachment).

void VpeSetMailAutoAttachDocType(
VpeHandle *hDoc*,
int *doc_type*
)

VpeHandle hDoc
Document Handle

int doc_type
possible values are:

Constant Name	Value	Comment
VPE_DOC_TYPE_AUTO	0	VpeMailDoc() will attach the document as VPE, PDF or HTML file, depending on the file suffix of the parameter *file_name* which is supplied to AddMailAttachment().
VPE_DOC_TYPE_VPE	1	MailDoc() will attach the document as VPE document file, regardless of the file suffix
VPE_DOC_TYPE_PDF	2	MailDoc() will attach the document as PDF document file, regardless of the file suffix
VPE_DOC_TYPE_HTML	3	MailDoc() will attach the document as HTML document file, regardless of the file suffix

Default:

VPE_DOC_TYPE_PDF

13.8 VpeGetMailAutoAttachDocType

[Windows platform only; not supported by the Community Edition]

Returns the current setting of the property MailAutoAttachDocType.

int VpeGetMailAutoAttachDocType(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

13.9 VpeClearMailAttachments

[Windows platform only; not supported by the Community Edition]

Clears the list of mail attachments created with VpeAddMailAttachment().

void VpeClearMailAttachments(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

13.10 VpeSetMailSubject

[Windows platform only; not supported by the Community Edition]

Allows you to specify the e-mail subject by code.

void VpeSetMailSubject(
  VpeHandle *hDoc*,
  LPCSTR *subject*
)

VpeHandle hDoc
  Document Handle

LPCSTR subject
  the subject of the message

Example:

```
VpeSetMailSubject(hDoc, "New VPE v3.0")
```

13.11 VpeSetMailText

[Windows platform only; not supported by the Community Edition]

Allows you to specify the e-mail text by code.

void VpeSetMailText(
    VpeHandle *hDoc*,
    LPCSTR *text*
)

VpeHandle hDoc
    Document Handle

LPCSTR text
    the message text

Example:

```
VpeSetMailText(hDoc, "Dear customer of VPE," + chr$(13) + chr$(10) + "we
              have a new version released.")
```

13.12 VpeSetMailWithDialog

[Windows platform only; not supported by the Community Edition]

Specifies, whether a dialog is shown to the user with input fields for recipients, subject, text and attachments when an e-mail is sent. An e-mail is sent either by pushing the e-Mail button in the preview or by executing MailDoc by code. It is very useful to set **MailWithDialog** to false, if you do automatic serial e-mailings in the background without user interaction.

void VpeSetMailWithDialog(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	show dialog
False	do not show the dialog

Default:
True

Remarks:
If the user has not logged on to a MAPI client, a logon dialog might be displayed once for the first e-mail that is sent, regardless of the setting of this property.

If you are setting MailWithDialog = False, at least one recipient must be specified by code (see VpeAddMailReceiver, otherwise a dialog is shown.

This option does not work with some MAPI clients (a dialog will always be shown). It also depends sometimes on the MAPI client configuration, whether a dialog is shown or not.

13.13 VpeMailDoc

[Windows platform only; not supported by the Community Edition]

Sends a mail with the current Mail-Properties.

int VpeMailDoc(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set.

The name of the default attachment is either the DevJobName (if it is set), or the title of the VPE preview window.

There are some problems with MAPI clients (e-mail software). This may cause that MailDoc() will return an error and / or that the e-mail button in the toolbar is disabled (grayed).

Recipients, subject, text and attachments specified by code will be shown in the MAPI client dialog (for example Microsoft Exchange or Outlook).

Not all MAPI clients return correct error codes. Some of them return "success", even if the user aborted. Others return "common failure“ if an attached file is not existing, instead of "attachment not found", etc.

Netscape Messenger and the Mozilla mail client seem not to work correctly as MAPI clients.

MAPI clients are manufactured by vendors independent of IDEAL Software; we make no warranty, implied or otherwise, regarding these product’s performance or reliability.

Special care must be taken, when creating VPE Document files (mailing a VPE Document means that a VPE Document file is created automatically) that contain pictures or UDO's.
For details please see VpeWriteDoc.

Example:

```
VpeSetMailSubject(hDoc, "VPE Demo")
VpeAddMailReceiver(hDoc, "MrX@dummyxyz.com", VMAIL_TO)
VpeAddMailReceiver(hDoc, "MrY@dummyxyz.com", VMAIL_CC)
VpeAddMailAttachment(hDoc, "c:\data\report.vpe", 0)
VpeMailDoc(hDoc)
```

```
VpeAddMailReceiver(hDoc, "[FAX: +49 1234 12345678]", VMAIL_TO)
VpeMailDoc(hDoc)
```

Will **FAX** the message to the given phone number! If a VPE Document is attached to the mail, the VISIBLE CONTENT of the document will be faxed. Requires MS-MAIL and MS-FAX, or a similar MAPI- Mail / Fax software to be installed on the system. Also the VPE Document viewer VPE View (see "VPE View - the Document Viewer" in the Programmer's Manual) needs to be correctly installed, otherwise VPE Document attachments will not be faxed with their visual content.

14 RTF Functions

14.1 RTF Functions

[Professional Edition and above]

Features of VPE

- Version independent and error tolerant RTF parser.This means: no termination, if unknown or erroneous formats are processed, instead they are ignored and skipped until the next known keyword is found.
- Any font-types, -sizes and -attributes (bold, italic underlined) can be used even in a single line or word
- Text- and background color can be set for each letter
- unlimited number of paragraphs (only limited by available memory)
- Automatic text break over multiple pages

For an introduction into RTF and a detailed description of VPE's RTF features, see see "RTF - Rich Text Format" in the Programmer's Manual.

NOTE: Often RTF documents are created on Windows platforms and use Windows specific True-Type fonts. Especially for the use on Non-Windows platforms there is the important method VpeSetFontSubstitution() available to substitute fonts, so you can use other fonts in place of the fonts used within an RTF document.

14.2 VpeWriteRTF

Outputs RTF text within a rectangle at position x, y, with the right border at x2 and the bottom border at y2.

The pen is invisible.

VpeCoord VpeWriteRTF(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *rtf_text*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions

LPCSTR rtf_text
the string to output

Returns:
the bottom y-coordinate generated by the output

Remarks:
VFREE: only the y2 coordinate may be set to VFREE, not x2.

In case of an error, LastError is set.

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Pictureobjects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

Example:

```
VpeSetFont(hDoc, "Arial", 12)
VpeWriteRTF(hDoc, 1, 1, -5, VFREE, "Hello \b World!")
VpeSetFont(hDoc, "Times New Roman", 16)
VpeSetUnderlined(hDoc, TRUE)
VpeWriteRTF(VLEFT, VBOTTOM, VRIGHT, VFREE, "Hello \b World!")
```

Produces the following output:

Hello **World!**
Hello **World!**

See also:
"RTF - Rich Text Format" in the Programmer's Manual

14.3 VpeWriteBoxRTF

Same as VpeWriteRTF(), but pen- and box-settings are used. Outputs RTF text within a rectangle at position x, y, with the right border at x2 and the bottom border at y2.

VpeCoord VpeWriteBoxRTF(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*,
  LPCSTR *rtf_text*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y, x2, y2
  position and dimensions

LPCSTR rtf_text
  the string to output

Returns:
  the bottom y-coordinate generated by the output

Remarks:
  VFREE: only the y2 coordinate may be set to VFREE, not x2.

  In case of an error, LastError is set.

  VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
  For details please see "Dynamic Positioning" in the Programmer's Manual.

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

14.4 VpeWriteRTFFile

Outputs RTF text, which is read from the file specified in "file_name", within a rectangle at position x, y, with the right border at x2 and the bottom border at y2. This is very useful for processing text already created by your end user with an RTF editor.

The pen is invisible.

VpeCoord VpeWriteRTFFile(
  VpeHandle *hDoc*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*,
  LPCSTR *file_name*
)

VpeHandle hDoc
  Document Handle

VpeCoord x, y, x2, y2
  position and dimensions

LPCSTR file_name
  the file with RTF text that is imported

Returns:
  the bottom y-coordinate generated by the output

Remarks:
  VFREE: only the y2 coordinate may be set to VFREE, not x2.

  Keywords in the RTF file override any VPE API properties. For example, if you set "\f1 = Arial" and in the RTF file it is defined as "\f1 = Times New Roman", then "\f1" will be "Times New Roman".

  In case of an error, LastError is set.

  VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
  For details please see "Dynamic Positioning" in the Programmer's Manual.

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

14.5 VpeWriteBoxRTFFile

Same as VpeWriteRTFFile(), but pen- and box-settings are used. Outputs RTF text, which is read from the file specified in "file_name", within a rectangle at position x, y, with the right border at x2 and the bottom border at y2. This is very useful for processing text already created by your end user with an RTF editor.

VpeCoord VpeWriteBoxRTFFile(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *file_name*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions

LPCSTR file_name
the file with RTF text that is imported

Returns:

VFREE: only the y2 coordinate may be set to VFREE, not x2.

the bottom y-coordinate generated by the output

Remarks:

Keywords in the RTF file override any VPE API properties. For example, if you set "\f1 = Arial" and in the RTF file it is defined as "\f1 = Times New Roman", then "\f1" will be "Times New Roman".

In case of an error, LastError is set.

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

See also:

"RTF - Rich Text Format" in the Programmer's Manual

14.6 VpeWriteRTFStream

Outputs RTF text, which is read from a supplied stream, within a rectangle at position x, y, with the right border at x2 and the bottom border at y2. This is very useful for processing rich text stored in a database.

The pen is invisible.

VpeCoord VpeWriteRTFStream(
  VpeHandle *hDoc*,
  VpeHandle *hStream*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*
)

VpeHandle hDoc
  Document Handle

VpeHandle hStream
  Stream Handle

VpeCoord x, y, x2, y2
  position and dimensions

Returns:
  the bottom y-coordinate generated by the output

Remarks:
  VFREE: only the y2 coordinate may be set to VFREE, not x2.

  Keywords in the RTF stream override any VPE API properties. For example, if you set "\f1 = Arial" and in the RTF stream it is defined as "\f1 = Times New Roman", then "\f1" will be "Times New Roman".

  In case of an error, LastError is set.

  VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
  For details please see "Dynamic Positioning" in the Programmer's Manual.

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

14.7 VpeWriteBoxRTFStream

Same as VpeWriteRTFStream(), but pen- and box-settings are used. Outputs RTF text, which is read from a supplied stream, within a rectangle at position x, y, with the right border at x2 and the bottom border at y2. This is very useful for processing rich text stored in a database.

VpeCoord VpeWriteBoxRTFStream(
VpeHandle *hDoc*,
VpeHandle *hStream*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*
)

VpeHandle hDoc
Document Handle

VpeHandle hStream
Stream Handle

VpeCoord x, y, x2, y2
position and dimensions

Returns:

VFREE: only the y2 coordinate may be set to VFREE, not x2.

the bottom y-coordinate generated by the output

Remarks:

Keywords in the RTF stream override any VPE API properties. For example, if you set "\f1 = Arial" and in the RTF stream it is defined as "\f1 = Times New Roman", then "\f1" will be "Times New Roman".

In case of an error, LastError is set.

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

See also:

"RTF - Rich Text Format" in the Programmer's Manual

14.8 VpeSetRTFFont

Modify the build-in RTF font table.

void VpeSetRTFFont(
VpeHandle *hDoc*,
int *ID*,
LPCSTR *name*
)

VpeHandle hDoc
Document Handle

int id
the entry ID

LPCSTR name
the name of the font that shall be assigned to the given entry-id

Remarks:

The current value of the CharSet property is also stored in the font table entry when calling this method.

VPE has build-in a predefined font table:

\f1 is predefined as "Arial"
\f2 is predefined as "Times New Roman"

Examples:

For example change \f1 with:

```
VpeSetCharSet(hDoc, SYMBOL_CHARSET)
VpeSetRTFFont(hDoc, 1, "Wingdings")
```

Or you can add as many fonts to the table as you like, for example:

```
VpeSetCharSet(hDoc, SYMBOL_CHARSET)
VpeSetRTFFont(hDoc, 3, "Wingdings")
```

See also:

"RTF - Rich Text Format" in the Programmer's Manual

14.9 VpeSetRTFColor

Modify the build-in RTF color table.

void VpeSetRTFColor(
  VpeHandle *hDoc*,
  int *ID*,
  COLORREF *color*
)

VpeHandle hDoc
  Document Handle

int id
  the entry ID

COLORREF color
  one of the predefined "COLOR_xyz" constants described in Programmer's Manual or any RGB value that shall be assigned to the given entry-id

Remarks:

RTF offers no way to set a transparent background mode, like VPE does. So we decided to set the background mode to transparent, in that case you use COLOR_WHITE = RGB(255, 255, 255) as background color.

VPE has build-in a predefined color table:

color1	BLACK	RGB(0, 0, 0)
color2	DKGRAY	RGB(128, 128, 128)
color3	GRAY	RGB(192, 192, 192)
color4	LTGRAY	RGB(230, 230, 230)
color5	WHITE	RGB(255, 255, 255)
color6	DKRED	RGB(128, 0, 0)
color7	RED	RGB(192, 0, 0)
color8	LTRED	RGB(255, 0, 0)
color9	DKORANGE	RGB(255, 64, 0)
color10	ORANGE	RGB(255, 128, 0)
color11	LTORANGE	RGB(255, 192, 0)
color12	DKYELLOW	RGB(224, 224, 0)
color13	YELLOW	RGB(242, 242, 0)
color14	LTYELLOW	RGB(255, 255, 0)
color15	DKGREEN	RGB(0, 128, 0)
color16	GREEN	RGB(0, 192, 0)
color17	LTGREEN	RGB(0, 255, 0)
color18	HIGREEN	RGB(0, 255, 128)
color19	BLUEGREEN	RGB(0, 128, 128)
color20	OLIVE	RGB(128, 128, 0)
color21	BROWN	RGB(128, 80, 0)
color22	DKBLUE	RGB(0, 0, 128)

color23	BLUE	RGB(0, 0, 255)
color24	LTBLUE	RGB(0, 128, 255)
color25	LTLTBLUE	RGB(0, 160, 255)
color26	HIBLUE	RGB(0, 192, 255)
color27	CYAN	RGB(0, 255, 255)
color28	DKPURPLE	RGB(128, 0, 128)
color29	PURPLE	RGB(192, 0, 192)
color30	MAGENTA	RGB(255, 0, 255)

Examples:

change color1 with

```
VpeSetRTFColor(hDoc, 1, RGB(10, 10, 10))
```

Or you can add as many colors to the table as you like, for example:

```
VpeSetRTFColor(hDoc, 31, RGB(10, 20, 30))
```

See also:

"RTF - Rich Text Format" in the Programmer's Manual

14.10 Build-In Paragraph Settings

VPE RTF has build-in a predefined paragraph setting. Note, whilst in RTF text the values for paragraph settings are in twips (= 1 / 1440 inch), all values for the properties and methods are specified in metric or inch units (depending on the Unit Transformation).

The following sections are a list of properties and methods to access those settings and their default values:

See also:

"RTF - Rich Text Format" in the Programmer's Manual

14.11 VpeSetFirstIndent

RTF: First-Line indent. Specifies the left indent of the first line of a new paragraph.

void VpeSetFirstIndent(
  VpeHandle *hDoc*,
  VpeCoord *indent*
)

VpeHandle hDoc
  Document Handle

VpeCoord indent
  the left indent of the first line

Default:
  0

Remarks:
  The left indent is relative to the left border of the object's rectangle. For example: VpeSetFirstIndent(hDoc, 2) means, that the left margin is set to:

  [left border of object rectangle] + 2 cm

  The first-line indent and the left indent are used together (they are added) for the first line. To gain a hanging indent, specify for the first-line indent the negative value of the left indent, which accumulates to zero then.

Example:

```
VpeSetFirstIndent(hDoc, -1)
VpeSetLeftIndent(hDoc, 1)
```

  Will result in a paragraph formatting like this:

This is the first line and
    this is the second line and
    this is the third line

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

14.12 VpeSetLeftIndent

RTF: Left indent. Specifies the left indent of all lines of a new paragraph.

void VpeSetLeftIndent(
    VpeHandle *hDoc*,
    VpeCoord *indent*
)

VpeHandle hDoc
    Document Handle

VpeCoord indent
    the left indent

Default:
    0

Remarks:
    The left indent is relative to the left border of the object's rectangle. For example:
    VpeSetLeftIndent(hDoc, 2) means, that the left margin is set to:

    [left border of object rectangle] + 2 cm

    The first-line indent and the left indent are used together (they are added) for the first line. To gain a hanging indent, specify for the first-line indent the negative value of the left indent, which accumulates to zero then.

Example:

```
VpeSetFirstIndent(hDoc, -1)
VpeSetLeftIndent(hDoc, 1)
```

Will result in a paragraph formatting like this:

This is the first line and
        this is the second line and
        this is the third line

See also:
    "RTF - Rich Text Format" in the Programmer's Manual

14.13 VpeSetRightIndent

RTF: Right indent. Specifies the right indent of all lines of a new paragraph.

void VpeSetRightIndent(
VpeHandle *hDoc*,
VpeCoord *indent*
)

VpeHandle hDoc
Document Handle

VpeCoord indent
the right indent

Default:
0

Remarks:
The right indent is relative to the right border of the object’s rectangle. For example: VpeSetRightIndent(hDoc, 2) means, that the right margin is set to:

[right border of object rectangle] - 2 cm

Example:

```
VpeSetRightIndent(hDoc, 2)
```

See also:
"RTF - Rich Text Format" in the Programmer's Manual

14.14 VpeSetSpaceBefore

The space before a new RTF paragraph. Note that VPE does intentionally NOT use this value for the first paragraph of a new object and additionally after a \page, \pagebb or \sect keyword - this means: it is also not used on a new page after an Auto Break has occurred (as normal RTF editors do).

void VpeSetSpaceBefore(
VpeHandle *hDoc*,
VpeCoord *space*
)

VpeHandle hDoc
Document Handle

VpeCoord space
the space before a new paragraph

Default:
0

See also:
"RTF - Rich Text Format" in the Programmer's Manual

14.15 VpeSetSpaceAfter

The space after a RTF paragraph. Note that VPE does intentionally NOT use this value for the last paragraph of an object and if a \page, \pagebb or \sect keyword follows - this means: before an Auto Break occurs. Also note that most RTF editors emit a \par command after the very last line of a document / section. This is not ignored by VPE. You will recognize it as a gap behind the last line, especially if you draw a surrounding box.

void VpeSetSpaceAfter(
  VpeHandle *hDoc*,
  VpeCoord *space*
)

VpeHandle hDoc
  Document Handle

VpeCoord space
  the space after a paragraph

Default:
  0

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

14.16 VpeSetSpaceBetween

RTF: Space between lines; if 0 is specified, the line spacing is automatically determined by the tallest character in the line; else this size is used only if it is taller than the tallest character.

void VpeSetSpaceBetween(
  VpeHandle *hDoc*,
  VpeCoord *space*
)

VpeHandle hDoc
  Document Handle

VpeCoord space
  the space between paragraphs

Default:
  0

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

14.17 VpeSetDefaultTabSize

This is the RTF default advance width, if a \tab is processed. The default tab size is only used in case that no individual tab is defined for the next tab position.

void VpeSetDefaultTabSize(
    VpeHandle *hDoc*,
    VpeCoord *default_tab_size*
)

VpeHandle hDoc
    Document Handle

VpeCoord default_tab_size
    the default TAB advance width

Default:
    1.25 (= 1.25 cm)

See also:
    "RTF - Rich Text Format" in the Programmer's Manual

14.18 VpeSetTab

Sets an individual RTF tab position. You may specify as many tab positions as you like.

void VpeSetTab(
  VpeHandle *hDoc*,
  VpeCoord *tab_position*,
  int *reserved*
)

VpeHandle hDoc
  Document Handle

VpeCoord tab_position
  tab position

int reserved
  for future use and should be set to zero

Example:

```
VpeSetTab(hDoc, 1)
VpeSetTab(hDoc, 2)
VpeSetTab(hDoc, 5)
```

Sets three individual tab positions at 1cm, 2cm and 5cm

```
VpeClearTab(hDoc, 2)
```

Removes the tab-stop at position 2cm (which was previously set by VpeSetTab()).

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

14.19 VpeClearTab

Removes an individual RTF tab position.

void VpeClearTab(
  VpeHandle *hDoc*,
  VpeCoord *tab_position*
)

VpeHandle hDoc
  Document Handle

VpeCoord tab_position
  tab position

Example:

```
VpeSetTab(hDoc, 1)
VpeSetTab(hDoc, 2)
VpeSetTab(hDoc, 5)
```

Sets three individual tab positions at 1cm, 2cm and 5cm

```
VpeClearTab(hDoc, 2)
```

Removes the tab-stop at position 2cm (which was previously set by VpeSetTab()).

See also:

"RTF - Rich Text Format" in the Programmer's Manual

14.20 VpeClearAllTabs

Removes all RTF tab settings, which had been set previously by VpeSetTab().

void VpeClearAllTabs(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

See also:
    "RTF - Rich Text Format" in the Programmer's Manual

14.21 VpeResetParagraph

Resets all RTF paragraph settings to their default values.

void VpeResetParagraph(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

14.22 Build-In Paragraph Settings: RTF Auto Page Break

VPE processes the three most important RTF paragraph styles (described in the next sections) to have the best control over text formatting when automatic page breaks occur.

Auto Page Breaks only occur if the property AutoBreakMode is set to AUTO_BREAK_ON or AUTO_BREAK_FULL.

The RTF properties KeepLines, KeepNextParagraph and ParagraphControl do not work between different RTF objects. They are only in effect within one and the same RTF object.
Example: You insert an RTF object and below a second one which is too long for the current page, so an AutoBreak is fired. In such a case the second RTF object is treated independently of the first one and paragraph control between the two objects is not in effect.

See also:

"RTF - Rich Text Format" in the Programmer's Manual
"Automatic Text Break" in the Programmer's Manual

14.23 VpeSetKeepLines

Keep RTF paragraph intact. A page break may not occur between the lines of a paragraph. Instead the whole paragraph is moved to the next page.

void VpeSetKeepLines(
  VpeHandle *hDoc*,
  int *yes_no*
)

VpeHandle hDoc
  Document Handle

int yes_no

Value	Description
True	Yes
False	No

Default:
  False

Remarks:
  Page breaks may occur between the lines of paragraphs, e.g. at their exact borders (see VpeSetKeepNextParagraph()).

  VpeSetKeepLines(), VpeSetKeepNextParagraph() and VpeSetParagraphControl() may be used together.

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

14.24 VpeSetKeepNextParagraph

Keep RTF paragraph with the next paragraph. A page break may not occur between the following paragraphs. This means, one paragraph may not stand alone at the top of a new page. Instead the paragraphs are moved to the next page.

void VpeSetKeepNextParagraph(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	Yes
False	No

Default:
False

Remarks:
Note: Page breaks may occur between the lines of paragraphs (see VpeSetKeepLines).

VpeSetKeepLines(), VpeSetKeepNextParagraph() and VpeSetParagraphControl() may be used together.

See also:
"RTF - Rich Text Format" in the Programmer's Manual

14.25 VpeSetParagraphControl

Keep RTF paragraph intact, a single line may not remain on the current page or on the next page. A page break may not occur for a following paragraph, if only the first line of the paragraph would remain on the **current page**, or if only the last line of the paragraph would be placed on the next page. The whole paragraph is moved to the next page instead. KeepLines, KeepNextPar and ParControl may be used together.

void VpeSetParagraphControl(
  VpeHandle *hDoc*,
  int *yes_no*
)

VpeHandle hDoc
  Document Handle

int yes_no

Value	Description
True	Yes
False	No

Default:
  False

Remarks:
  VpeSetKeepLines(), VpeSetKeepNextParagraph() and VpeSetParagraphControl() may be used together.

See also:
  "RTF - Rich Text Format" in the Programmer's Manual

15 Clickable Objects

15.1 Clickable Objects

[Windows platform only, Professional Edition and above]

Objects can be made clickable by assigning them a unique Object ID. Moving with the mouse over such an object in the preview changes the cursor to a pointing hand. If the user clicks onto such an object, VPE fires the event VPE_OBJECTCLICKED to your application which includes the assigned Object ID.

In reaction to the event you can for example open a separate dialog, showing more detailed information about the clicked text or image.

See also the source code of the "Clickable Objects" demo.

15.2 VpeEnableClickEvents

[Windows platform only, Professional Edition and above]

Specifies, whether clickable objects shall be activated (= be clickable) or not. This property is valid for the **whole document**. If you set it to False, no object in the preview will be clickable, regardless whether you assigned an Object ID to an object or not.

void VpeEnableClickEvents(
    VpeHandle *hDoc*,
    int *yes_no*
)

VpeHandle hDoc
    Document Handle

int yes_no

Value	Description
True	enable
False	disable

Default:
    True

15.3 VpeSetObjectID

[Windows platform only, Professional Edition and above]

Defines an Object ID that will be assigned to the next object(s), that are inserted into the document. With the Object ID you are able to identify a clicked object later when showing the preview. When the user clicks onto such an object with an assigned ID, the VPE_OBJECTCLICKED event is sent together with the Object ID to your application.

The Object ID may have any value that can be represented with a long integer. The Object ID has to be different from 0 (zero). Setting the Object ID to zero means, that the object is not clickable.

void VpeSetObjectID(
    VpeHandle *hDoc*,
    int *id*
)

VpeHandle hDoc
    Document Handle

int id
    an Object ID to identify the clicked object when processing the VPE_OBJECTCLICKED event

Default:
    0

Remarks:
    You may assign one and the same Object ID to any number of objects. It is the whole responsibility of your application, how it will work with Object ID's.

    The following objects can be assigned an Object ID (and therefore can be made clickable):

- Box
- Ellipse
- Pie
- Print(-Box)
- Write(-Box)
- Picture
- Barcodes (1D and 2D)
- RTF
- Charts
- UDO
- FormField

Example:

```
VpeSetObjectID(hDoc, 1)
VpePrint(hDoc, 1, 1, "Monthly Report")
```

Assigns the ObjectID 1 to the text object "Monthly Report".

```
VpeSetObjectID(hDoc, 2)
VpePicture(hDoc, 1, 1, VFREE, VFREE, "data.bmp")
```

Assigns the ObjectID 2 to the image "data.bmp".

```
VpeSetObjectID(hDoc, 0)
VpePrint(hDoc, 1, VBOTTOM, "This image shows data.")
```

Assigns NO ObjectID to the text object "This image shows data.". Therefore this object is not clickable.

Processing of the VPE_OBJECTCLICKED event in C/C++
(in the window-procedure of the parent window of the VPE preview):

```
case VPE_OBJECTCLICKED:
  wsprintf(s, "Object #%d was clicked", VpeGetObjectID(lParam));
  MessageBox(hWnd, s, "CLICK:", MB_OK);
  break;
```

The event handler in the example above will show a message box with the Object ID of the clicked object.

15.4 VpeGetObjectID

[Windows platform only, Professional Edition and above]

Returns the Object ID of the clicked object.

int VpeGetObjectID(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle

Returns:
  the Object ID of the clicked object

Remarks:
  This property is only accessible while your application is processing the event VPE_OBJECTCLICKED. Calling this function while not processing the event will return invalid - and therefore useless - data.

Example:
  see VpeSetObjectID

15.5 VpeGetClickedObject

[Windows platform only, Enterprise Edition and above]

Returns the **VPE Object handle** - instead of the ObjectID, which is only an integer - of the object that fired the VPE_OBJECTCLICKED event.

VpeHandle VpeGetClickedObject(
    VpeHandle hDoc
)

VpeHandle hDoc
    Document Handle

Returns:
    the VPE Object handle of the clicked object

Remarks:
    This property is only accessible while your application is processing the event VPE_OBJECTCLICKED.

    If you call this function while not processing the event VPE_OBJECTCLICKED, it will return NULL.

16 UDO - User Defined Objects

16.1 UDO - User Defined Objects

[Windows platform only, Professional Edition and above]

With User Defined Objects it is possible to draw any kind of object within VPE - and therefore to preview and print it!

A User Defined Object is first of all the same like a Box-Object (see "The Object-Oriented Style" in the Programmer's Manual). It inherits all properties of the Box object, including BkgMode, PenSize, PenStyle, etc.

UDO's are event driven, that means: VPE sends an event to your application when the object needs to be painted on the output device (i.e. the preview, the printer, the fax or whatsoever).

Therefore you specify additionally a long integer value (named lParam) when creating a UDO, which will help you identifying the object later during the paint event.

At the moment VPE paints the UDO, VPE first of all draws the Box object. Afterwards VPE fires the event VPE_UDO_PAINT. In that moment, your application can access the device context of VPE to draw everything you want into the rectangle of the UDO!

Moreover, with the ActiveX you have the powerful property "UDOPicture", which can be assigned an OLE / COM object so it is painted automatically into the rectangle of the UDO!

16.2 VpeCreateUDO

[Windows platform only, Professional Edition and above]

Creates a User Defined Object (UDO). lParam can be of any value and is for your private use to identify the object later during processing the VPE_UDO_PAINT event.

void VpeCreateUDO(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
int *lParam*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions of the UDO

int lParam
numeric ID to identify the UDO during the VPE_UDO_PAINT event

Remarks:
Special care must be taken, when creating VPE files that contain pictures or UDO's.
For details please see VpeWriteDoc.

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

Example:
see VpeGetUDODC

16.3 VpeGetUDOlParam

[Windows platform only, Professional Edition and above]

Returns the long integer value lParam from a UDO.

int VpeGetUDOlParam(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the ID of the UDO that needs to be drawn

Remarks:
    Only in case you specify a VPE Document handle: this property is only accessible while your application is processing the event VPE_UDO_PAINT. The value returned should be used to identify the object which needs to be drawn. Calling this function while not processing the event will return invalid - and therefore useless - data.

Example:
    see VpeGetUDODC

16.4 VpeSetUDOlParam

[Windows platform only, Enterprise Edition and above]

Sets the integer value lParam of a UDO.

void VpeSetUDOlParam(
    VpeHandle *hObject*,
    int *lParam*
)

VpeHandle hObject
    VPE Object Handle

int lParam
    the new ID for the UDO

Remarks:

This function only works if you specify a **VPE Object handle**. If you specify a handle to a different object the function does nothing.

16.5 VpeGetUDODC

[Windows platform only, Professional Edition and above]

This property is only accessible while your application is processing the event VPE_UDO_PAINT. Retrieving this value while not processing the event will return invalid - and therefore useless - data.

It returns the current Device Context VPE is painting on (this maybe the screen or a printer Device Context). Using this Device Context, you can call any Windows GDI function and paint yourself into Device Context. Be careful, do not destroy the Device Context and make sure to free all used system resources. In case of misuse the system may hang or fail to work correctly.

HDC VpeGetUDODC(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Returns:
the HDC of the UDO that needs to be drawn

Remarks:
Before VPE fires the event VPE_UDO_PAINT, it draws the Box object the UDO object is inherited from. Afterwards VPE saves the Device Context with the Windows GDI function SaveDC( ) and creates a clipping rectangle around the object.

After you have finished painting and return control to VPE by returning from your event-handler, VPE will delete the clipping rectangle and restore the Device Context with the Windows GDI function RestoreDC( ).

Example:

```
VpeCreateUDO(hDoc, 1, 1, -18, -18, 1);
```

Creates a UDO with the current properties for the Box-Object and the ID "1" (=lParam)

Processing of the VPE_UDO_PAINT event in C/C++
(in the window-procedure of the parent window of the VPE preview):

```
case VPE_UDO_PAINT:
   UdoPaint(lParam);
   break;
```

The function UdoPaint() looks like this:

```
void UdoPaint(VpeHandle hDoc)
{
   HDC hDC;
   RECT rc;
   HPEN hpen, holdpen;

   hDC = VpeGetUDODC(hDoc);
   VpeGetUDODrawRect(hDoc, &rc);

   if (VpeGetUDOlParam(hDoc) == 1)
   {
         hpen = CreatePen(PS_SOLID, 1, COLOR_RED);
```

```
       holdpen = SelectObject(hDC, hpen);
       MoveToEx(hDC, rc.left, rc.top, NULL);
       // LineTo() draws a line from the current position up to,
       // but not including, the specified point.
       // Therefore we add 1 to each coordinate.
       LineTo(hDC, rc.right + 1, rc.bottom + 1);
       MoveToEx(hDC, rc.right, rc.top, NULL);
       LineTo(hDC, rc.left - 1, rc.bottom + 1);
       SelectObject(hDC, holdpen);
       DeleteObject(hpen);
  }
}
```

The example draws two crossing lines within the UDO.

16.6 VpeGetUDOIsPrinting

[Windows platform only, Professional Edition and above]

If you want to be able, to scale the visual content of a UDO accordingly to the scale of the Preview, the scale of a printing device or the scale of a virtual surface (i.e. when exporting to PDF or an image) you need to be able to determine onto what surface the UDO has to be drawn. Use this property as well as the UDOIsExporting property for such a task. Do not use the property IsPrinting because while the print job is running and *IsPrinting = true*, it can still happen that the UDO needs to be painted onto the screen Device Context, for example because you moved a window of another application over the preview of VPE.

int VpeGetUDOIsPrinting(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
True	yes, the UDO has to be drawn onto a printer's Device Context
False	no, the UDO has not to be drawn onto a printer's Device Context

16.7 VpeGetUDOIsExporting

[Windows platform only, Professional Edition and above]

If you want to be able, to scale the visual content of a UDO accordingly to the scale of the Preview, the scale of a printing device or the scale of a virtual surface (i.e. when exporting to PDF or an image) you need to be able to determine onto what surface the UDO has to be drawn. Use this property as well as the UDOIsPrinting property for such a task.

int VpeGetUDOIsExporting(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
True	yes, the UDO has to be drawn onto a virtual surface
False	no, the UDO has not to be drawn onto a virtual surface

16.8 VpeGetUDODpiX

[Windows platform only, Professional Edition and above]

Returns the X-resolution of the UDO’s current output surface in DPI (Dots Per Inch).

int VpeGetUDODpiX(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    The X-resolution of the UDO’s current output surface in DPI (Dots Per Inch).

16.9 VpeGetUDODpiY

[Windows platform only, Professional Edition and above]

Returns the Y-resolution of the UDO’s current output surface in DPI (Dots Per Inch).

int VpeGetUDODpiY(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    The Y-resolution of the UDO’s current output surface in DPI (Dots Per Inch).

16.10 VpeGetUDODrawRect

[Windows platform only, Professional Edition and above]

This method only returns valid information while your application is processing the event VPE_UDO_PAINT. Calling this method while not processing the event will return invalid - and therefore useless - data.

The function returns the rectangle (this is exactly the area you may paint in) of the UDO in **DEVICE COORDINATES** (!) (pixels, not metric units). Therefore, independently from the resolution of the output device or the scaling factor of the preview, the coordinates are computed by VPE accordingly. Be it the 96 DPI screen, a 200 DPI fax, a 600 DPI printer or a 300 DPI exported image (yes, UDO works with Picture Export, too!).

void VpeGetUDODrawRect(
  VpeHandle *hDoc*,
  RECT **rc*
)

VpeHandle hDoc
  Document Handle

*RECT *rc*
  receive parameter for the rectangle of the UDO that needs to be drawn (coordinates are in Device Units)

Returns:
  the rectangle of the UDO that needs to be drawn in parameter "rc" (coordinates are in Device Units)

Remarks:
  Your application may draw inside of the returned rectangle **including** the coordinates returned in parameter "rc".

Example:
  see VpeGetUDODC

16.11 VUDO_XYZ Flags

[Windows platform only, Professional Edition and above]

In addition to the method VpeGetUDODrawRect() you can retrieve each coordinate of the UDO rectangle with the method VpeGet() using one of the following V-Flags:

Constant Name	Value	Description
VUDO_LEFT	-104	only usable for VpeGet()
VUDO_RIGHT	-105	only usable for VpeGet()
VUDO_TOP	-106	only usable for VpeGet()
VUDO_BOTTOM	-107	only usable for VpeGet()
VUDO_WIDTH	-108	only usable for VpeGet()
VUDO_HEIGHT	-109	only usable for VpeGet()

The V-Flags above are only accessible while your application is processing the event VPE_UDO_PAINT. Retrieving the values while not processing the event will return invalid - and therefore useless - data.

In contrast to the other V-Flags, the VUDO_xyz Flags return the bounding rectangle of the UDO in **DEVICE COORDINATES** (!) (pixels, not metric units). Therefore, independently from the resolution of the output device or the scaling factor of the preview, the coordinates are computed by VPE accordingly. Be it the 96 DPI screen, a 200 DPI fax, a 600 DPI printer or a 300 DPI export image (yes, UDO works with Picture Export, too!).

17 Picture Export

17.1 Picture Export

[Windows platform only, Professional Edition and above]

VPE is able to export pages or rectangular parts of pages from the document to image files.

The supported formats for Picture Export are:

- BMP
- WMF (Windows only)
- EMF (Windows only)
- JPEG (compression ratio can be set freely)
- PNG
- TIFF 6.0 (Fax G3, Fax G4, LZW, Packbits, Deflate, JPEG, Multipage)
- GIF (Multipage)

For all bitmap formats you can specify the color depth and the resolution (in DPI). Additionally dithering is possible.

Exported Metafiles:

Text justification and included bitmaps may not be correctly ***displayed,*** if you re-import the generated metafiles into the preview, because exported metafiles are always rendered to a virtual 600 DPI resolution. So displaying them on the 96 DPI screen brings tolerances into the object placement due to coordinate rounding problems. Nevertheless if you print them on 300 DPI printers, or a multiple of 300 DPI printers (600 DPI, 1200 DPI, etc.), they are printed perfectly.

The following restrictions apply **only** to exported WMF files:

- Barcodes are not exported
- Metafiles inside the document are not exported
- Some software does not rely onto the information provided in the header of the WMF about its dimensions, instead it scans the WMF file to compute its dimensions. This can cause problems, if a part of a page is only exported, when objects which intersect the part are also outside this area. The computing software will include the whole object and distort the WMF. This is a problem of the software, which tries to be too clever.

17.2 VpeSetJpegExportOptions

[Windows platform only, Professional Edition and above]

Specifies options for exported JPEG images.

void VpeSetJpegExportOptions(
    VpeHandle *hDoc*,
    long *options*
)

VpeHandle hDoc
    Document Handle

long options
    several options

Constant Name	Value	Decimal	Comment
PICEXP_JPEG_DEFAULT	0x0000	0	Default, saves with good quality (75:1)
PICEXP_JPEG_HI_QUALITY	0x0080	128	Saves with superb quality (100:1)
PICEXP_JPEG_GOOD_QUALITY	0x0100	256	Saves with good quality (75:1)
PICEXP_JPEG_MID_QUALITY	0x0200	512	Saves with normal quality (50:1)
PICEXP_JPEG_LO_QUALITY	0x0400	1024	Saves with average quality (25:1)
PICEXP_JPEG_BAD_QUALITY	0x0800	2048	Saves with bad quality (10:1)
Integer X in [0..100]	0x0 – 0x64	0 – 100	Save with quality X:1
PICEXP_JPEG_PROGRESSIVE	0x2000	8192	If set, JPG files are written progressive (interlaced)

Default:
    PICEXP_JPEG_DEFAULT

Remarks:
    The flag PICEXP_JPEG_PROGRESSIVE can be combined with any of the other flags.

Example:

```
VpeSetJpegExportOptions(hDoc, PICEXP_JPEG_DEFAULT + PICEXP_JPEG_PROGRESSIVE)
```

17.3 VpeGetJpegExportOptions

[Windows platform only, Professional Edition and above]

Returns the current settings for exported JPEG images.

long VpeGetJpegExportOptions(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Returns:
the current settings for exported JPEG images

17.4 VpeSetTiffExportOptions

[Windows platform only, Professional Edition and above]

Specifies options for exported TIFF images.

void VpeSetTiffExportOptions(
  VpeHandle *hDoc*,
  long *options*
)

VpeHandle hDoc
  Document Handle

long options
  several options

Constant Name	Value	Decimal	Comment
PICEXP_TIFF_DEFAULT	0x0000	0	Default, save using CCITTFAX4 compression for 1-bit bitmaps and LZW compression for any other bitmaps
PICEXP_TIFF_PACKBITS	0x0100	256	Save using PACKBITS compression
PICEXP_TIFF_DEFLATE	0x0200	512	Save using DEFLATE compression (a.k.a. ZLIB compression)
PICEXP_TIFF_ADOBE_DEFLATE	0x0400	1024	Save using ADOBE DEFLATE compression
PICEXP_TIFF_NONE	0x0800	2048	Save without any compression
PICEXP_TIFF_CCITTFAX3	0x1000	4096	Save using CCITT Group 3 fax encoding
PICEXP_TIFF_CCITTFAX4	0x2000	8192	Save using CCITT Group 4 fax encoding
PICEXP_TIFF_LZW	0x4000	16384	Save using LZW compression
PICEXP_TIFF_JPEG	0x8000	32768	Save using JPEG compression (8-bit greyscale and 24-bit only. Default to LZW for other bitdepths.)
PICEXP_TIFF_APPEND	0x40000000	1073741824	Append file to multi-page TIFF

Default:
  PICEXP_TIFF_DEFAULT

Remarks:
  The flag PICEXP_TIFF_APPEND can be combined with other flags.

Example:

```
VpeSetTiffExportOptions(hDoc, PICEXP_TIFF_DEFAULT + PICEXP_TIFF_APPEND)
```

17.5 VpeGetTiffExportOptions

[Windows platform only, Professional Edition and above]

Returns the current settings for exported TIFF images.

long VpeGetTiffExportOptions(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current settings for exported TIFF images

17.6 VpeSetBmpExportOptions

[Windows platform only, Professional Edition and above]

Specifies options for exported BMP images.

void VpeSetBmpExportOptions(
  VpeHandle *hDoc*,
  long *options*
)

VpeHandle hDoc
  Document Handle

long options
  several options

Constant Name	Value	Decimal	Comment
PICEXP_BMP_DEFAULT	0x00	0	Default, save without any compression
PICEXP_BMP_RLE	0x01	1	Save RLE compressed

Default:
  PICEXP_BMP_DEFAULT

Example:

```
VpeSetBmpExportOptions(hDoc, PICEXP_BMP_RLE)
```

17.7 VpeGetBmpExportOptions

[Windows platform only, Professional Edition and above]

Returns the current settings for exported BMP images.

long VpeGetBmpExportOptions(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current settings for exported BMP images

17.8 VpeSetPnmExportOptions

[Windows platform only, Professional Edition and above]

Specifies options for exported PBM, PGM, PPM images.

void VpeSetPnmExportOptions(
  VpeHandle *hDoc*,
  long *options*
)

VpeHandle hDoc
  Document Handle

long options
  several options

Constant Name	Value	Decimal	Comment
PICEXP_PNM_DEFAULT	0x00	0	PBM, PGM, PPM files are written RAW
PICEXP_PNM_ASCII	0x01	1	PBM, PGM, PPM are written ASCII

Default:
  PICEXP_PNM_DEFAULT

Example:

```
VpeSetPnmExportOptions(hDoc, PICEXP_PNM_ASCII)
```

17.9 VpeGetPnmExportOptions

[Windows platform only, Professional Edition and above]

Returns the current settings for exported PBM, PGM, PPM images.

long VpeGetPnmExportOptions(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current settings for exported PBM, PGM, PPM images

17.10 VpeSetGifExportOptions

[Windows platform only, Professional Edition and above]

Specifies options for exported GIF images.

void VpeSetGifExportOptions(
    VpeHandle *hDoc*,
    long *options*
)

VpeHandle hDoc
    Document Handle

long options
    several options

Constant Name	Value	Decimal	Comment
PICEXP_GIF_DEFAULT	0x0000	0	nothing special
PICEXP_GIF_APPEND	0x40000000	1073741824	append file to multi-page GIF

Default:
    PICEXP_GIF_DEFAULT

Remarks:
    The flag PICEXP_GIF_APPEND can be combined with any of the other flags.

Example:

```
VpeSetGifExportOptions(hDoc, PICEXP_GIF_DEFAULT + PICEXP_GIF_APPEND)
```

17.11 VpeGetGifExportOptions

[Windows platform only, Professional Edition and above]

Returns the current settings for exported GIF images.

long VpeGetGifExportOptions(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current settings for exported GIF images

17.12 VpeSetPictureExportColorDepth

[Windows platform only, Professional Edition and above]

Specifies the color depth of an exported image.

void VpeSetPictureExportColorDepth(
VpeHandle *hDoc*,
int *depth*
)

VpeHandle hDoc
Document Handle

int depth
possible values are:

Constant Name	Value	Comment
PICEXP_COLOR_MONO	1	
PICEXP_COLOR_16	4	
PICEXP_COLOR_256	8	
PICEXP_COLOR_HI	16	
PICEXP_COLOR_TRUE	24	

Default:
PICEXP_COLOR_TRUE

Remarks:
The higher the color depth, the more memory is needed during export to create the image.
The color depth can not be specified for metafiles (WMF / EMF), metafiles are always exported in true-color.

See also:
PictureExportDither

17.13 VpeSetPictureExportDither

[Windows platform only, Professional Edition and above]

Specifies, if an exported image shall be dithered to a lower color resolution when written to file. This is very useful, if you want to export smaller images (regarding their size in bytes) - or if you want to fax (in b/w) a true-color image - and to keep the visual information at a high quality.

void VpeSetPictureExportDither(
VpeHandle *hDoc*,
int *dither*
)

VpeHandle hDoc
Document Handle

int dither
possible values are:

Constant Name	Value	Comment
PICEXP_DITHER_NONE	0	
PICEXP_DITHER_MONO	1	
PICEXP_DITHER_256	3	
PICEXP_DITHER_256_WU	4	

Default:
PICEXP_DITHER_NONE

Remarks:
For PICEXP_DITHER_MONO: images are dithered with the Floyd-Steinberg dithering algorithm. The source bitmap may have any of the following color depths: 1, 4, 8, 16, 24, 32 bits. If the source bitmap is a monochrome bitmap, VPE creates a copy without dithering.

For PICEXP_DITHER_256: this only works, if the source bitmap is 24-bit, i.e. PictureExportColorDepth is PICEXP_COLOR_TRUE. Images are not dithered, instead VPE uses color-reduction (NeuQuant neural-net quantization algorithm by Anthony Dekker). This algorithm creates very good results but is rather slow.

For PICEXP_DITHER_256_WU: this only works, if the source bitmap is 24-bit, i.e. PictureExportColorDepth is PICEXP_COLOR_TRUE. Images are not dithered, instead VPE uses color-reduction (Xiaolin Wu color quantization algorithm). This algorithm is much faster than the NeuQuant algorithm, but does not create as good results. In some cases it might lead to false colors.

Example:

```
VpeSetPictureExportColorDepth(hDoc, PICEXP_COLOR_256)
VpeSetPictureExportDither(hDoc, PICEXP_DITHER_MONO)
VpePictureExportPage(hDoc, "test.bmp", 1)
```

Instructs VPE, to generate from page one of the document internally a 256 color image first, and to dither it down afterwards to b / w (monochrome) when writing it to file.

17.14 VpePictureExportPage

[Windows platform only, Professional Edition and above]

Exports the page specified in parameter "page_no" to file. The settings for the export options of the specific file type, PictureExportColorDepth and PictureExportDither are used.

The type of image is automatically determined by the suffix of FileName (e.g. ".JPG" = JPEG). If FileName contains no suffix or a suffix which does not specify an image type (e.g. ".001"), the image type can be specified with the property PictureType.

The resolution of the image is defined with VpeSetPictureDefaultDPI().

int VpePictureExportPage(
    VpeHandle *hDoc*,
    LPCSTR *file_name*,
    int *page_no*
)

VpeHandle hDoc
    Document Handle

LPCSTR file_name
    (path- and) filename of exported image

int page_no
    page number of the page that shall be exported from the document

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set.

During export, the BusyProgressBar is shown.

The higher the color depth and the higher the resolution, the more memory is needed during export to create the image. The color depth and resolution can not be specified for metafiles (WMF / EMF), metafiles are always exported in true-color with a resolution of 600 x 600 DPI.

Example:

```
VpeSetPictureExportColorDepth(hDoc, PICEXP_COLOR_256)
VpeSetPictureExportDither(hDoc, PICEXP_DITHER_MONO)
VpeSetDefaultPictureDPI(hDoc, 300, 300)
VpePictureExportPage(hDoc, "test.bmp", 1)
```

Instructs VPE, to generate from page one of the document internally a 256 color image with 300 x 300 DPI resolution first, and to dither it down afterwards to b/w (monochrome) in the same 300 x 300 DPI resolution when writing it to a BMP file named "test.bmp".

Example 2:

```
VpeSetPictureExportColorDepth(hDoc, PICEXP_COLOR_TRUE)

// VPE determines the file type from the filename suffix. Because we use
// here for the suffix ".001", ".002", etc., we tell VPE the type of file:
VpeSetPictureType(hDoc, PIC_TYPE_PNG)

// PNG is always written with Deflate (ZLib) compression

// Set 96 DPI resolution:
VpeSetDefaultPictureDPI(hDoc, 96, 96)
VpePictureExportPage(hDoc, "image.001", 2);
```

Instructs VPE, to export page 2 of the document as true-color Deflate-compressed PNG image with 96 x 96 DPI resolution named "image.001".

Example 3:

```
VpeSetPictureExportDither(hDoc, PICEXP_DITHER_MONO)
VpeSetPictureType(hDoc, PIC_TYPE_TIFF);

// Select FaxG4 as export format:
VpeSetPictureExportOptions(hDoc, PICEXP_TIFF_CCITTFAX4)

// Set 200 DPI resolution:
VpeSetDefaultPictureDPI(hDoc, 200, 200)
VpePictureExportPage(hDoc, "image.002", 3)
```

Instructs VPE, to export page 3 of the document as dithered b / w Fax G4 TIFF image with 200 x 200 DPI resolution named "image.002".

See also:

VpePictureExport

17.15 VpePictureExportPageStream

[Windows platform only, Professional Edition and above]

Identical to VpePictureExportPage(), but exports the picture to a stream.

int VpePictureExportPageStream(
  VpeHandle *hDoc*,
  VpeHandle *hStream*,
  int *page_no*
)

VpeHandle hDoc
  Document Handle

VpeHandle hStream
  The handle of the stream where the picture is written to. The stream must have been created before by calling VpeCreateMemoryStream().

int page_no
  page number of the page that shall be exported from the document

Returns:

Value	Description
True	success
False	failure

Remarks:
  WMF and EMF files are written to a temporary file internally first and afterwards to the stream. This is, because the Windows API does not allow to stream metafiles.

See also:
  VpePictureExport

17.16 VpePictureExport

[Windows platform only, Professional Edition and above]

Exports a rectangular part specified by x, y, x2, y2 of the page specified in parameter "page_no" to file. The settings for the export options of the specific file type, PictureExportColorDepth and PictureExportDither are used.

The type of image is automatically determined by the suffix of FileName (e.g. ".JPG" = JPEG). If FileName contains no suffix or a suffix which does not specify an image type (e.g. ".001"), the image type can be specified with the property PictureType. The resolution of the image is defined with VpeSetPictureDefaultDPI().

int VpePictureExport(
    VpeHandle *hDoc*,
    LPCSTR *file_name*,
    int *page_no*,
    VpeCoord *x*,
    VpeCoord *y*,
    VpeCoord *x2*,
    VpeCoord *y2*
)

VpeHandle hDoc
    Document Handle

LPCSTR file_name
    (path- and) filename of exported image

int page_no
    page number of the page that shall be exported from the document

VpeCoord x, y, x2, y2
    rectangle of the page in metric or inch units that shall be exported as image

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError is set.

During export, the BusyProgressBar is shown.

For x, y, x2, y2 you can also specify the V-Flags and for x2, y2 negative values to specify a delta value instead of an absolute coordinate.

The higher the color depth and the higher the resolution, the more memory is needed during export to create the image.

Metafiles are only supported on the Windows platform. The color depth and resolution can not be specified for metafiles (WMF / EMF), metafiles are always exported in true-color with a resolution of 600 x 600 DPI. The maximum dimensions for exported WMF is limited to 60 x 60 cm, this does not apply to EMF.

Example:

```
VpeSetPictureExportColorDepth(hDoc, PICEXP_COLOR_256)
VpeSetPictureExportDither(hDoc, PICEXP_DITHER_MONO)
VpeSetDefaultPictureDPI(hDoc, 300, 300)
VpePictureExport(hDoc, "test.bmp", 1, 1, 1, -5, -5)
```

Instructs VPE to generate from the rectangle (1, 1, 6, 6) from page one of the document internally a 256 color image with 300 x 300 DPI resolution first, and to dither it down afterwards to b/w (monochrome) in the same 300 x 300 DPI resolution when writing it to the BMP file named "test.bmp".

```
VpePictureExport(hDoc, "test.bmp", 1, VLEFT, VTOP, VRIGHT, VBOTTOM)
```

Instructs VPE to export the last inserted object to an image file.

See also:

VpePictureExportStream

17.17 VpePictureExportStream

[Windows platform only, Professional Edition and above]

Identical to VpePictureExport(), but exports the picture to a stream.

int VpePictureExportPageStream(
VpeHandle *hDoc*,
VpeHandle *hStream*,
int *page_no*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*
)

VpeHandle hDoc
Document Handle

VpeHandle hStream
The handle of the stream where the picture is written to. The stream must have been created before by calling VpeCreateMemoryStream().

int page_no
page number of the page that shall be exported from the document

int x, y, x2, y2
rectangle of the page in metric or inch units that shall be exported as image

Returns:

Value	Description
True	success
False	failure

Remarks:
WMF and EMF files are written to a temporary file internally first and afterwards to the stream. This is, because the Windows API does not allow to stream metafiles.

See also:
VpePictureExport

18 Memory Streams

18.1 Memory Streams

[Professional Edition and above]

This chapter explains special memory streams of VPE, which allow to read and write documents and images from / to memory. This is especially useful, if you want to store VPE documents or images in databases as BLOBs, or if you wish to create for example PDF documents on a web server in memory - without writing them to disk - in order to send them directly via HTTP to clients.

A memory stream is like a file, which is held in memory only.

See Also:

VpeWriteDocStream
VpeWriteDocStreamPageRange

VpeReadDocStream
VpeReadDocStreamPageRange

VpePictureStream
VpeRenderPictureStream
VpePictureExportPageStream
VpePictureExportStream

VpeWriteRTFStream
VpeWriteBoxRTFStream
VpeRenderRTFStream
VpeRenderBoxRTFStream

18.2 VpeCreateMemoryStream

[Professional Edition and above]

Creates a new empty stream in memory. A memory stream can grow as large as you wish, the size is only limited by available memory. VPE divides a memory stream into chunks of equal size - by default 16 KB. This means that an initial stream only occupies 16 KB of memory. If you write more data to a memory stream, the memory stream is enlarged by additional chunks as required. Internally VPE stores each chunk separately and links each chunk with the next.

VpeHandle VpeCreateMemoryStream(
  VpeHandle *hDoc*,
  long *chunk_size*
)

VpeHandle hDoc
  Document Handle

long chunk_size
  The size of each chunk, if you set this parameter to zero, VPE uses a default chunk size of 16 KB, which is a reasonable value.

Returns:
  A handle to the created memory stream. In case of an error, the returned handle is NULL.

  You must ALWAYS close all streams you ever opened or created, when you have done using them. Not doing so will result in memory leaks (only at the moment your application is closed, all streams will be released from memory). Calling CloseDoc() will not close streams.

Remarks:
  sets LastError

18.3 VpeCloseStream

[Professional Edition and above]

Closes the given stream.

void VpeCloseStream(
    VpeHandle *hStream*
)

VpeHandle hStream
    stream handle

Remarks:
    When closing a stream, the internal stream handle is destroyed. You may NOT supply the stream handle in any further calls to a method, which takes a stream handle as parameter.

18.4 VpeStreamRead

[Professional Edition and above]

Reads data from the supplied stream.

int VpeStreamRead(
  VpeHandle *hStream*,
  BYTE **buffer*,
  int *size*
)

VpeHandle hStream
  stream handle

*BYTE *buffer*
  pointer to the buffer, to which the data will be copied from the stream

int size
  the number of bytes that shall be read

Returns:
  The number of bytes read from the stream.

18.5 VpeStreamWrite

[Professional Edition and above]

Writes data to the supplied stream.

int VpeStreamWrite(
VpeHandle *hStream*,
BYTE **buffer*,
int *size*
)

VpeHandle hStream
stream handle

*BYTE *buffer*
pointer to the buffer, from which the data will be copied to the stream

int size
the number of bytes that shall be written

Returns:

Value	**Description**
True	success
False	failure

18.6 VpeGetStreamSize

[Professional Edition and above]

Returns the size - in bytes - of the given stream.

int VpeGetStreamSize(
  VpeHandle *hStream*
)

VpeHandle hStream
  stream handle

Returns:
  The size of the stream in bytes.

18.7 VpeStreamIsEof

[Professional Edition and above]

Returns the End Of File status of the given stream.

The Eof() function returns true after the first read operation that attempts to read past the end of the file. Eof() continues to report true, until Seek() is called.

int VpeStreamIsEof(
VpeHandle *hStream*
)

VpeHandle hStream
stream handle

Returns:

Value	Description
True	EOF
False	not EOF

18.8 VpeGetStreamState

[Professional Edition and above]

Returns the status of the stream.

For memory streams the state can become false, if the system is out-of-memory, or if you try to read past the end of file, or if you seek outside the file.

int VpeGetStreamState(
    VpeHandle *hStream*
)

VpeHandle hStream
    stream handle

Returns:

Value	Description
True	status ok
False	failure

18.9 VpeGetStreamPosition

[Professional Edition and above]

Returns the current position - in bytes - of the internal file pointer.

long VpeGetStreamPosition(
    VpeHandle *hStream*
)

VpeHandle hStream
    stream handle

Returns:
    The current position within the stream.

18.10 VpeStreamSeek

[Professional Edition and above]

Moves the file pointer to the given position.

void VpeStreamSeek(
    VpeHandle hStream,
    long pos
)

VpeHandle hStream
    stream handle

long pos
    the new position

18.11 VpeStreamSeekEnd

[Professional Edition and above]

Moves the file pointer to the given position, which is relative to the end of the stream. Therefore the parameter *pos* must always be negative or null.

void VpeStreamSeekEnd(
  VpeHandle *hStream*,
  long *pos*
)

VpeHandle hStream
  stream handle

long pos
  the new position, must always be negative or null

18.12 VpeStreamSeekRel

[Professional Edition and above]

Moves the file pointer to the given position, which is relative to the current position.

void VpeStreamSeekRel(
    VpeHandle *hStream*,
    long *offset*
)

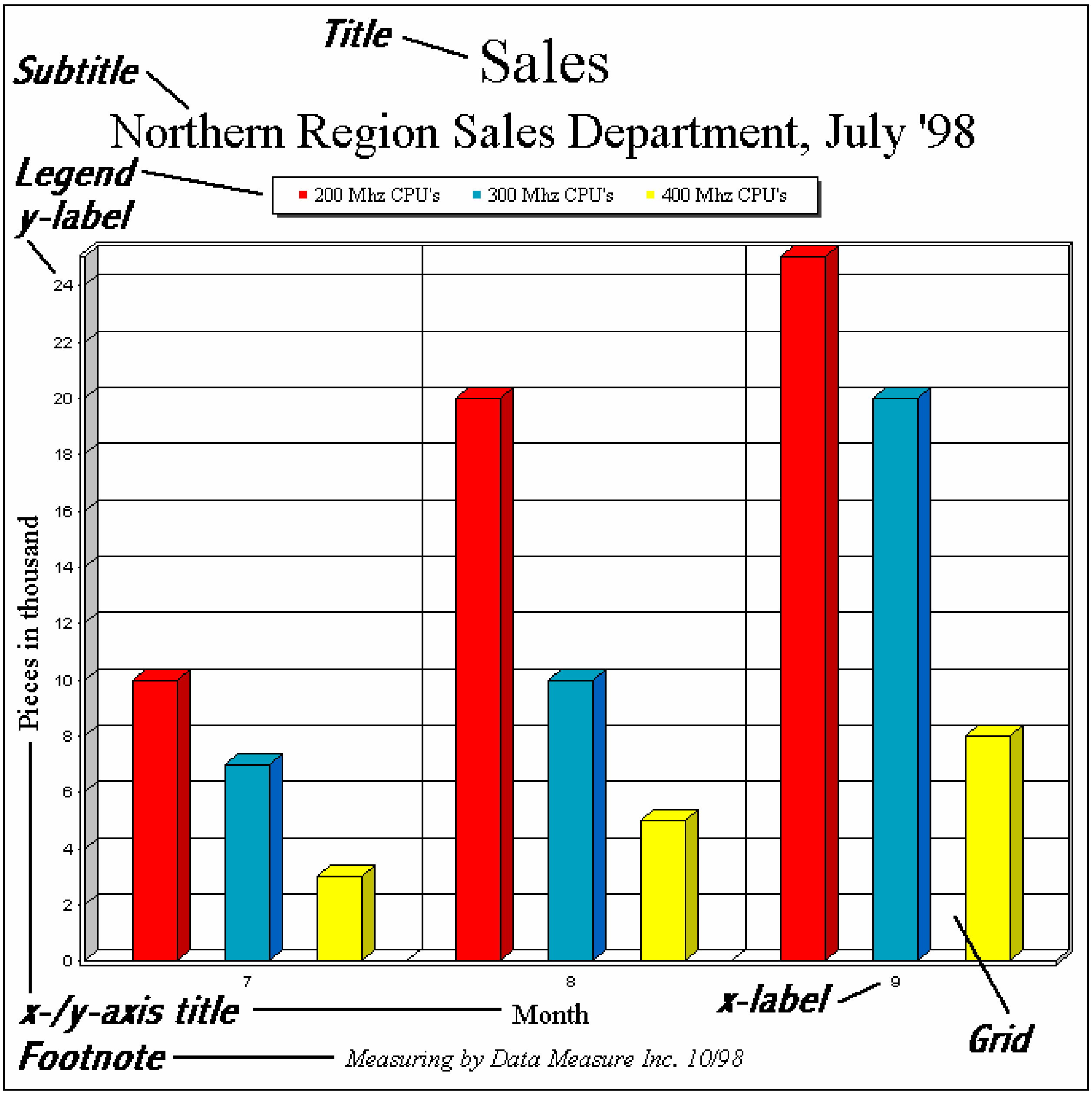
VpeHandle hStream
    stream handle

long pos
    the new position, relative to the current position

19 Charts

19.1 Charts

[Professional Edition and above]



The chart was created with the following short code sequence:

```
VpeHandle hData;
hData = VpeChartDataCreate(hdoc, 3, 3);

VpeSetChartTitle(hdoc, "Sales");
VpeSetChartSubTitle(hdoc, "Northern Region Sales Department, July '98");
VpeSetChartFootNote(hdoc, "Measuring by Data Measure Inc. 10/98");

VpeChartDataSetXAxisTitle(hdoc, hData, "Month");
VpeChartDataSetYAxisTitle(hdoc, hData, "Pieces in thousand");

VpeSetChartLegendPosition(hdoc, VCHART_LEGENDPOS_TOP);
VpeChartDataAddLegend(hdoc, hData, "200 Mhz CPU's");
VpeChartDataAddLegend(hdoc, hData, "300 Mhz CPU's");
VpeChartDataAddLegend(hdoc, hData, "400 Mhz CPU's");

VpeChartDataAddValue(hdoc, hData, 0, 10);
VpeChartDataAddValue(hdoc, hData, 0, 20);
VpeChartDataAddValue(hdoc, hData, 0, 25);
VpeChartDataAddValue(hdoc, hData, 1, 7);
VpeChartDataAddValue(hdoc, hData, 1, 10);
VpeChartDataAddValue(hdoc, hData, 1, 20);
VpeChartDataAddValue(hdoc, hData, 2, 3);
VpeChartDataAddValue(hdoc, hData, 2, 5);
VpeChartDataAddValue(hdoc, hData, 2, 8);

VpeSetChartXLabelStartValue(hdoc, 7);
VpeChart(hdoc, 1, 1, -18, -18, hData, VCHART_3D_BAR);
```

19.1.1 The SmartChart Technology

The dimensions of the different chart elements - like title, subtitle, etc. - depend on the total size of the chart. The bigger the dimensions of a chart are, the bigger are the font sizes and line thicknesses (for line charts). Vice versa the smaller the dimensions of a chart are, the smaller are the font sizes and line thicknesses.

VPE computes the font- and line sizes optimized depending on the available space. We call that ***SmartChart Technology:*** depending on the items displayed (for example with Title, but without Subtitle and the Legend positioned on the Left / Top position) the chart algorithm computes the available space for the visible items and computes font sizes and line thicknesses correspondingly.

VPE offers several properties to specify *factors* for the font sizes and line thicknesses. With those factors you can not define absolute font- or pen sizes, but relative font- / pen sizes compared to the original default values.

19.1.2 In VPE, Charts internally consist of two basic parts

1) The Chart-Data Object

This is the set of numeric values you want to visualize. VPE organizes the data internally in a table. As with SQL, this table has rows and columns:

	Column 0 Apples	Column 1 Bananas
row 0	10	5
row 1	20	10
row 2	30	15
row 3	40	20

Note, that rows and columns start with the index 0.

Additionally, the following data related elements are part of the Chart-Data:

- Legend
- Labels of the x- and y-axis
- x- and y-unit signs
- colors and appearance of each data column

2) The Chart-Properties

This is the Title, Subtitle, Footnote and colors for graphical elements like grid and text, etc.

Chart Properties behave like all other properties in VPE. This means you can create charts of different types (e.g. Line Chart, Bar Chart, Pie Chart, etc.) that are all using the same ChartData Object, but each Chart can have its individual properties - for example its own title.

The Chart is - like text, barcodes, etc. - inherited from the Box Object (see "The Object-Oriented Style" in the Programmer's Manual), so the properties for a a Box(like BkgMode, PenSize, PenStyle, etc.) apply to the Chart Object.

19.2 VpeChartDataCreate

Creates an empty ChartData-Object, prepared to hold the numeric data and properties related to the numeric data. The ChartData object stores the numeric data of a chart. You can create multiple charts of different Chart-Types (like: Line Chart, Bar Chart, Pie Chart, etc.) out of one ChartData object. All charts which is assigned the same ChartData object, will display the same ChartData in a different style.

VpeHandle VpeChartDataCreate(
VpeHandle *hDoc*,
int *columns*,
int *rows*
)

VpeHandle hDoc
Document Handle

int columns
number of columns that shall be reserved

int rows
number of rows that shall be reserved

Returns:
The handle of the created ChartData object.

Example:

```
VpeHandle hData
hData = VpeChartDataCreate(hDoc, 2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

19.3 VpeChartDataAddValue

Adds a new value to the specified column of the ChartData object specified in the Parameter hData.

void VpeChartDataAddValue(
  VpeHandle *hDoc*,
  VpeHandle *hData*,
  int *column*,
  double *value*
)

VpeHandle hDoc
  Document Handle

VpeHandle hData
  handle of ChartData object

int column
  column, where to add the value

double value
  value to add

Example:

```
VpeHandle hData
hData = VpeChartDataCreate(hDoc, 2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

```
VpeChartDataAddValue(hDoc, hData, 0, 10)
VpeChartDataAddValue(hDoc, hData, 0, 20)
VpeChartDataAddValue(hDoc, hData, 0, 30)
VpeChartDataAddValue(hDoc, hData, 0, 40)

VpeChartDataAddValue(hDoc, hData, 1, 5)
VpeChartDataAddValue(hDoc, hData, 1, 10)
VpeChartDataAddValue(hDoc, hData, 1, 15)
VpeChartDataAddValue(hDoc, hData, 1, 20)
```

Now the internal data table of the ChartData object will look like this:

	Column 0	Column 1
row 0	10	5
row 1	20	10
row 2	30	15
row 3	40	20

19.4 VpeChartDataAddLegend

Adds a new descriptive string for a column to the legend.

void VpeChartDataAddLegend(
  VpeHandle *hDoc*,
  VpeHandle *hData*,
  LPCSTR *legend*
)

VpeHandle hDoc
  Document Handle

VpeHandle hData
  handle of ChartData object

LPCSTR legend
  legend string

Example:

```
VpeHandle hData
hData = VpeChartDataCreate(hDoc, 2, 4)
VpeChartDataAddLegend(hDoc, hData, "Apples")
VpeChartDataAddLegend(hDoc, hData, "Bananas")
```

Creates a ChartData object with two columns, titled "Apples" and "Bananas" in the legend.

19.5 VpeChartDataSetXAxisTitle

Sets the x-axis title of the chart. This is the descriptive text of the x-axis.

void VpeChartDataSetXAxisTitle(
  VpeHandle *hDoc*,
  VpeHandle *hData*,
  LPCSTR *x_axis_title*
)

VpeHandle hDoc
  Document Handle

VpeHandle hData
  handle of ChartData object

LPCSTR x_axis_title
  title for x-axis

19.6 VpeChartDataSetYAxisTitle

Sets the y-axis title of the chart. This is the descriptive text of the y-axis.

void VpeChartDataSetYAxisTitle(
    VpeHandle *hDoc*,
    VpeHandle *hData*,
    LPCSTR *y_axis_title*
)

VpeHandle hDoc
    Document Handle

VpeHandle hData
    handle of ChartData object

LPCSTR y_axis_title
    title for y-axis

19.7 VpeChartDataAddXLabel

Adds a new X-Label to the chart. By default, the x-axis is automatically labeled. With VpeSetChartXLabelState() you can switch to user defined labels and then add X-Labels to the chart. (see also VpeSetChartXLabelStartValue())

void VpeChartDataAddXLabel(
    VpeHandle *hDoc*,
    VpeHandle *hData*,
    LPCSTR *xlabel*
)

VpeHandle hDoc
    Document Handle

VpeHandle hData
    handle of ChartData object

LPCSTR xlabel
    string of label to add

Example:

```
VpeHandle hData
hData = VpeChartDataCreate(hDoc, 2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

```
VpeChartDataAddValue(hDoc, hData, 0, 10)
VpeChartDataAddValue(hDoc, hData, 0, 20)
VpeChartDataAddValue(hDoc, hData, 0, 30)
VpeChartDataAddValue(hDoc, hData, 0, 40)

VpeChartDataAddValue(hDoc, hData, 1, 5)
VpeChartDataAddValue(hDoc, hData, 1, 10)
VpeChartDataAddValue(hDoc, hData, 1, 15)
VpeChartDataAddValue(hDoc, hData, 1, 20)
```

```
VpeChartDataAddLegend(hDoc, hData, "Apples")
VpeChartDataAddLegend(hDoc, hData, "Bananas")
```

Now the internal data table of the ChartData object will look like this:

	Column 0 "Apples"	Column 1 "Bananas"
row 0	10	5
row 1	20	10
row 2	30	15
row 3	40	20

With the following code

```
VpeSetChartXLabelState(hDoc, VCHART_LABEL_USER)
VpeChartDataAddXLabel(hDoc, hData, "1. Quarter")
VpeChartDataAddXLabel(hDoc, hData, "2. Quarter")
VpeChartDataAddXLabel(hDoc, hData, "3. Quarter")
VpeChartDataAddXLabel(hDoc, hData, "4. Quarter")
```

the internal data table of the ChartData object will look like this:

	Column 0 "Apples"	Column 1 "Bananas"
row 0 "1. Quarter"	10	5
row 1 "2. Quarter"	20	10
row 2 "3. Quarter"	30	15
row 3 "4. Quarter"	40	20

19.8 VpeChartDataAddYLabel

Adds a new Y-Label to the chart. By default, the y-axis is labeled automatically. With VpeSetChartYLabelState() you can switch to user defined labels and then add Y-Labels to the chart.

Each label that would automatically be drawn can be replaced with this method.

void VpeChartDataAddYLabel(
    VpeHandle *hDoc*,
    VpeHandle *hData*,
    LPCSTR y*label*
)

VpeHandle hDoc
    Document Handle

VpeHandle hData
    handle of ChartData object

LPCSTR ylabel
    string of label to add

Remarks:
    You can control if and on what position y-labels are drawn with the following properties:

- VpeSetChartYLabelState()
- VpeSetChartYGridStep()
- VpeSetChartYLabelStep()
- VpeSetChartYLabelDivisor()
- VpeChartDataSetMinimum()
- VpeChartDataSetMaximum()

19.9 VpeChartDataSetColor

By default, VPE assigns itself default colors to each data column. With this method you can specify individual colors for each column.

void VpeChartDataSetColor(
  VpeHandle *hDoc*,
  VpeHandle *hData*,
  int *column*,
  COLORREF *color*
)

VpeHandle hDoc
  Document Handle

VpeHandle hData
  handle of ChartData object

int column
  column the color shall be assigned to

COLORREF color
  one of the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

19.10 VpeChartDataSetLineStyle

With this method you can specify individual line styles for each column.

void VpeChartDataSetLineStyle(
VpeHandle *hDoc*,
VpeHandle *hData*,
int *column*,
int *pen_style*
)

VpeHandle hDoc
Document Handle

VpeHandle hData
handle of ChartData object

int column
column the line style shall be assigned to

int pen_style
one of the windows pen styles; possible values are:

Constant Name	Value	Comment
psSolid	0	
psDash	1	-------
psDot	2	....
psDashDot	3	_._._._
psDashDotDot	4	_.._.._

Default:
PS_SOLID

Example:

```
VpeChartDataSetLineStyle(hDoc, hData, 0, PS_SOLID)
```

19.11 VpeChartDataSetHatchStyle

If you create a Bar-, Area- or Pie Chart from the ChartData, you can specify individual hatch styles for each column with this method. By default, no hatching is used.

void VpeChartDataSetHatchStyle(
VpeHandle *hDoc*,
VpeHandle *hData*,
int *column*,
int *style*
)

VpeHandle hDoc
Document Handle

VpeHandle hData
handle of ChartData object

int column
column the line style shall be assigned to

int style
column the hatch style shall be assigned to; possible values are:

Constant Name	Value	Comment
hsNone	-1	see HatchStyle – possible styles
hsHorizontal	0	see HatchStyle – possible styles
hsVertical	1	see HatchStyle – possible styles
hsFDiagonal	2	see HatchStyle – possible styles
hsBDiagonal	3	see HatchStyle – possible styles
hsCross	4	see HatchStyle – possible styles
hsDiagCross	5	see HatchStyle – possible styles

Default:
HS_NONE (no hatching is used)

Example:

```
VpeChartDataSetHatchStyle(hDoc, hData, 0, HS_DIAGCROSS)
```

19.12 VpeChartDataSetPointType

If you create a Point Chart from the ChartData, you can specify individual point styles for each column with this method.

void VpeChartDataSetPointType(
  VpeHandle *hDoc*,
  VpeHandle *hData*,
  int *column*,
  int *pointtype*
)

VpeHandle hDoc
  Document Handle

VpeHandle hData
  handle of ChartData object

int column
  column the color shall be assigned to

int pointtype
  possible values are:

Constant Name	Value	Comment
VCHART_SYMBOL_NONE	-1	
VCHART_SYMBOL_SQUARE	0	
VCHART_SYMBOL_TRIANGLE	1	
VCHART_SYMBOL_CIRCLE	2	
VCHART_SYMBOL_CROSS	3	
VCHART_SYMBOL_X	4	
VCHART_SYMBOL_POINT	5	

Default:
  VCHART_SYMBOL_X

Example:

```
VpeChartDataSetPointType(hDoc, hData, 0, VCHART_SYMBOL_SQUARE)
```

19.13 VpeChartDataSetMinimum

VPE determines the minimum value of the numeric data stored in a ChartData object itself. You can change the minimum value with this method.

void VpeChartDataSetMinimum(
  VpeHandle *hDoc*,
  VpeHandle *hData*,
  double *minimum*
)

VpeHandle hDoc
  Document Handle

VpeHandle hData
  handle of ChartData object

double minimum
  the minimum value of the numeric data in a ChartData object

Remarks:
  Because VPE compares - and maybe updates - the current determined minimum value with each value added by VpeChartDataAddValue(), you should call this method **after** you have finished adding values.

19.14 VpeChartDataSetMaximum

VPE determines the maximum value of the numeric data stored in a ChartData object itself. You can change the maximum value with this method.

void VpeChartDataSetMaximum(
    VpeHandle *hDoc*,
    VpeHandle *hData*,
    double *maximum*
)

VpeHandle hDoc
    Document Handle

VpeHandle hData
    handle of ChartData object

double maximum
    the maximum value of the numeric data in a ChartData object

Remarks:
    Because VPE compares - and maybe updates - the current determined minimum value with each value added by VpeChartDataAddValue(), you should call this method **after** you have finished adding values.

19.15 VpeChartDataAddGap

Adds a gap to the data. This is especially useful for line charts in case a data value is missing. If you would add a zero, the line would be drawn to the zero-value, but this wouldn’t reflect, that there is NO value.

void VpeChartDataAddGap(
    VpeHandle *hDoc*,
    VpeHandle *hData*,
    int *column*
)

VpeHandle hDoc
    Document Handle

VpeHandle hData
    handle of ChartData object

int column
    the column where to add the gap

19.16 VpeChartDataAddRow

Resizes the internal data table of the ChartData object specified in the Parameter hData, so it can hold one more new row per column. This is especially useful if you don't know in advance, how many rows the ChartData data table will have and you need to add rows while querying your data source (or database).

void VpeChartDataAddRow(
    VpeHandle *hDoc*,
    VpeHandle *hData*,
)

VpeHandle hDoc
    Document Handle

VpeHandle hData
    handle of ChartData object

19.17 VpeChartDataAddColumn

Resizes the internal data table of the ChartData object specified in the Parameter hData, and adds a new empty column. This is especially useful if you don't know in advance, how many columns the ChartData data table will have and you need to add columns while querying your data source (or database).

void VpeChartDataAddColumn(
    VpeHandle *hDoc*,
    VpeHandle *hData*,
)

VpeHandle hDoc
    Document Handle

VpeHandle hData
    handle of ChartData object

19.18 VpeSetChartTitle

(Chart Property) Sets the title property for the next created chart object.

void VpeSetChartTitle(
    VpeHandle *hDoc*,
    LPCSTR *title*
)

VpeHandle hDoc
    Document Handle

LPCSTR title
    title of the chart

19.19 VpeSetChartTitleFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart title.

void VpeSetChartTitleFontSizeFactor(
  VpeHandle *hDoc*,
  double *factor*
)

VpeHandle hDoc
  Document Handle

double factor
  possible values 0.1 <= factor <= 10

Default:
  1.0

19.20 VpeSetChartSubTitle

(Chart Property) Sets the subtitle property for the next created chart object.

void VpeSetChartSubTitle(
    VpeHandle *hDoc*,
    LPCSTR *subtitle*
)

VpeHandle hDoc
    Document Handle

LPCSTR subtitle
    subtitle of the chart

19.21 VpeSetChartSubTitleFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart's subtitle.

void VpeSetChartSubTitleFontSizeFactor(
    VpeHandle *hDoc*,
    double *factor*
)

VpeHandle hDoc
    Document Handle

double factor
    possible values 0.1 <= factor <= 10

Default:
    0.6

19.22 VpeSetChartFootNote

(Chart Property) Sets the footnote property for the next created chart object.

void VpeSetChartFootNote(
    VpeHandle *hDoc*,
    LPCSTR *footnote*
)

VpeHandle hDoc
    Document Handle

LPCSTR footnote
    footnote of the chart

19.23 VpeSetChartFootNoteFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart's footnote.

void VpeSetChartFootNoteFontSizeFactor(
VpeHandle *hDoc*,
double *factor*
)

VpeHandle hDoc
Document Handle

double factor
possible values 0.1 <= factor <= 10

Default:
0.5

19.24 VpeSetChartAxisTitleFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart's x- and y-axis titles.

void VpeSetChartAxisTitleFontSizeFactor(
  VpeHandle *hDoc*,
  double *factor*
)

VpeHandle hDoc
  Document Handle

double factor
  possible values 0.1 <= factor <= 3

Default:
  1.0

19.25 VpeSetChartLegendFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart's legend.

void VpeSetChartLegendFontSizeFactor(
 VpeHandle *hDoc*,
 double *factor*
)

VpeHandle hDoc
 Document Handle

double factor
 possible values 0.1 <= factor <= 3

Default:
 1.4

19.26 VpeSetChartLineWidthFactor

(Chart Property) Line Charts only: Specifies a factor for the line thickness.

void EXPO VpeSetChartLineWidthFactor(
  VpeHandle *hDoc*,
  double *factor*
)

VpeHandle hDoc
  Document Handle

double factor
  possible values 0.1 <= factor <= 8

Default:
  3.0

19.27 VpeSetChartBarWidthFactor

(Chart Property) Bar Charts only (2D and 3D): Specifies a factor for bar width.

property double ChartBarWidthFactor

write; runtime only

Possible Values:
possible values 0.2 <= factor <= 1.9

Default:
1.0

19.28 VpeSetChartRow

(Chart Property) For Pie Chart only. Selects the row of the ChartData object which shall be used by a pie chart. A Pie Chart can only visualize one row of data.

void VpeSetChartRow(
VpeHandle *hDoc*,
int *row*
)

VpeHandle hDoc
Document Handle

int row
the row that shall be visualized by a Pie Chart

Default:
0 (the pie chart is created out of the first row of the ChartData object)

Example:

```
VpeSetChartRow(hDoc, 0)
VpeChart(hDoc, 1, 1, -18, -18, hData, VCHART_3D_PIE)
```

Creates a 3D Pie Chart from the 1st row of a ChartData object.

```
VpeSetChartRow(hDoc, 1)
VpeChart(hDoc, 1, 1, -18, -18, hData, VCHART_PIE)
```

Creates a 2D Pie Chart from the 2nd row of a ChartData object.

19.29 VpeSetChartGridBkgColor

(Chart Property) Sets the chart grid background color for the next created chart object.

void VpeSetChartGridBkgColor(
  VpeHandle *hDoc*,
  COLORREF *bkgcolor*
)

VpeHandle hDoc
  Document Handle

COLORREF bkgcolor
  one of the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
  COLOR_BLACK

19.30 VpeSetChartGridBkgMode

(Chart Property) Sets the **chart grid** background mode for the next created chart object.

void VpeSetChartGridBkgMode(
  VpeHandle *hDoc*,
  int *mode*
)

VpeHandle hDoc
  Document Handle

int mode
  possible values are:

Constant Name	Value	Comment
VBKG_SOLID	0	solid background color
VBKG_TRANSPARENT	1	transparent background

Default:
  VBKG_TRANSPARENT for 2-D charts and VBKG_SOLID for 3-D charts.

Remarks:
  The chart grid background is the area within the grid. The backgound mode for the whole chart object is set with VpeSetBkgMode().

19.31 VpeSetChartGridType

(Chart Property) Sets the grid type for the next created chart object.

void VpeSetChartGridType(
    VpeHandle *hDoc*,
    int *gridtype*
)

VpeHandle hDoc
    Document Handle

int gridtype
    possible values are:

Constant Name	**Value**	**Comment**
VCHART_GRID_NONE	-1	No Grid is drawn
VCHART_GRID_BOTH_AXIS	0	Grid is drawn for both axis
VCHART_GRID_X_AXIS	1	Grid is only drawn for the x-axis
VCHART_GRID_Y_AXIS	2	Grid is only drawn for the y-axis

Default:
    VCHART_GRID_BOTH_AXIS

19.32 VpeSetChartGridColor

(Chart Property) Sets the grid color for the next created chart object.

void VpeSetChartGridColor(
VpeHandle *hDoc*,
COLORREF *gridcolor*
)

VpeHandle hDoc
Document Handle

COLORREF gridcolor
one of the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

19.33 VpeSetChartYGridStep

(Chart Property) Sets the y-grid-step for the next created chart object.

The y-grid-step determines at what value on the y-axis a y-grid-line is drawn.

void VpeSetChartYGridStep(
VpeHandle *hDoc*,
double *gridstepy*
)

VpeHandle hDoc
Document Handle

double gridstepy
the grid stepping

Default:
Auto (see: VpeSetChartYAutoGridStep())

Example:

```
VpeSetChartYGridStep(hDoc, 5)
```

means, a y-grid line is drawn at y-values in a distance of 5
(e.g. a y-grid-line is drawn at -10, -5, 0, 5, 10, 15, 20, ...)

```
VpeSetChartYGridStep(hDoc, 8)
```

means, a y-grid line is drawn at y-values in a distance of 10
(e.g. a y-grid-line is drawn at -20, -10, 0, 10, 20, 30, ...)

19.34 VpeSetChartYAutoGridStep

(Chart Property) Instructs VPE to determine itself the grid stepping (see VpeSetChartYGridStep()) to gain readable results depending on the scale of the chart.

void VpeSetChartYAutoGridStep(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Remarks:
This is the default behavior of VPE. You can turn the auto stepping off by setting ChartYGridStep to any value.

19.35 VpeSetChartLegendPosition

(Chart Property) Sets the legend position for the next created chart object.

void VpeSetChartLegendPosition(
    VpeHandle *hDoc*,
    int *legendpos*
)

VpeHandle hDoc
    Document Handle

int legendpos
    possible values are:

Constant Name	**Value**	**Comment**
VCHART_LEGENDPOS_NONE	-1	No legend drawn
VCHART_LEGENDPOS_RIGHT	0	
VCHART_LEGENDPOS_RIGHT_TOP	1	
VCHART_LEGENDPOS_RIGHT_BOTTOM	2	
VCHART_LEGENDPOS_LEFT	3	
VCHART_LEGENDPOS_LEFT_TOP	4	
VCHART_LEGENDPOS_LEFT_BOTTOM	5	
VCHART_LEGENDPOS_TOP	6	
VCHART_LEGENDPOS_BOTTOM	7	

Default:
    VCHART_LEGENDPOS_LEFT_TOP

19.36 VpeSetChartLegendBorderStat

(Chart Property) Determines, whether the legend is drawn with a frame and a shadow or not for the next created chart object.

void VpeSetChartLegendBorderStat(
    VpeHandle *hDoc*,
    int *legendborderstat*
)

VpeHandle hDoc
    Document Handle

int legendborderstat
    possible values are:

Value	**Description**
True	drawn
False	invisible

Default:
    True

19.37 VpeSetChartXLabelState

(Chart Property) Determines, how the x-axis labels are drawn for the next created chart object.

void VpeSetChartXLabelState(
VpeHandle *hDoc*,
int *xlabelstate*
)

VpeHandle hDoc
Document Handle

int xlabelstate
possible values are:

Constant Name	Value	Comment
VCHART_LABEL_NONE	-1	No labels are drawn
VCHART_LABEL_USER	0	User defined labels are drawn
VCHART_LABEL_AUTO	1	Labeling is done automatically

Default:
VCHART_LABEL_AUTO

19.38 VpeSetChartPieLegendWithPercent

(Chart Property) Pie Chart only: Determines, whether the percent values of each pie element are shown in the legend.

void VpeSetChartPieLegendWithPercent(
  VpeHandle *hDoc*,
  int *yes_no*
)

VpeHandle hDoc
  Document Handle

int yes_no

Value	**Description**
True	yes, the percent values are shown
False	no, they are not shown

Default:
  True

19.39 VpeSetChartPieLabelType

(Chart Property) Pie Chart only: Determines, how the labels of a pie chart are drawn.

void EXPO VpeSetChartPieLabelType(
VpeHandle *hDoc*,
int *label_type*
)

VpeHandle hDoc
Document Handle

int label_type
possible values are:

Constant Name	Value	Comment
VCHART_PIE_LABEL_NONE	0	no labels are drawn
VCHART_PIE_LABEL_PERCENTAGE	1	the percentage values are drawn
VCHART_PIE_LABEL_LEGEND	2	the legend texts are used for the labels
VCHART_PIE_LABEL_XLABELS	3	the user-defined x-labels are used

Default:
VCHART_PIE_LABEL_PERCENTAGE

19.40 VpeSetChartXLabelFontSizeFactor

(Chart Property) Specifies a factor for the font size of the x-axis labels.

void VpeSetChartXLabelFontSizeFactor(
  VpeHandle *hDoc*,
  double *factor*
)

VpeHandle hDoc
  Document Handle

double factor
  possible values 0.1 <= factor <= 3

Default:
  1.5

19.41 VpeSetChartXLabelAngle

(Chart Property) Sets the drawing angle (clockwise) for the x-axis labels.

void VpeSetChartXLabelAngle(
VpeHandle *hDoc*,
int *xlabelangle*
)

VpeHandle hDoc
Document Handle

int xlabelangle
possible values are:

Value	Description
0	0 degrees, labels are drawn horizontally
900	90 degrees, labels are drawn vertically
1800	180 degrees, labels are drawn horizontally
2700	270 degrees, labels are drawn vertically

Default:
0

19.42 VpeSetChartXLabelStartValue

(Chart Property) Sets the start value for the x-labels. If the x-labeling is done automatically by VPE (i.e. ChartXLabelState = VCHART_LABEL_AUTO; see VpeSetChartXLabelState()), this is the start value where VPE will start to number each row.

void VpeSetChartXLabelStartValue(
    VpeHandle *hDoc*,
    int *xlabelstartvalue*
)

VpeHandle hDoc
    Document Handle

int xlabelstartvalue
    the start value for x-labels

Default:
    0 (= labeling will start at 0)

Example:
    By default, VPE will label the x-axis with "0, 1, 2, 3, ..."

    with:

```
VpeSetChartXLabelStartValue(hDoc, 7)
```

    VPE will label the x-axis with "7, 8, 9, 10, ..."

19.43 VpeSetChartYLabelState

(Chart Property) Determines, how the y-axis labels are drawn for the next created chart object.

void VpeSetChartYLabelState(
VpeHandle *hDoc*,
int *ylabelstate*
)

VpeHandle hDoc
Document Handle

int ylabelstate
possible values are:

Constant Name	Value	Comment
VCHART_LABEL_NONE	-1	No labels are drawn
VCHART_LABEL_USER	0	User defined labels are drawn
VCHART_LABEL_AUTO	1	Labeling is done automatically

Default:
VCHART_LABEL_AUTO

19.44 VpeSetChartYLabelFontSizeFactor

(Chart Property) Specifies a factor for the font size of the y-axis labels.

void VpeSetChartYLabelFontSizeFactor(
  VpeHandle *hDoc*,
  double *factor*
)

VpeHandle hDoc
  Document Handle

double factor
  possible values 0.1 <= factor <= 3

Default:
  1.5

19.45 VpeSetChartYLabelStep

(Chart Property) Sets the y-label-step for the next created chart object. The y-label-step determines the n-th y-grid-line where a label is drawn.

void VpeSetChartYLabelStep(
    VpeHandle *hDoc*,
    int *ylabelstep*
)

VpeHandle hDoc
    Document Handle

int ylabelstep
    the n-th y-grid line where a label is drawn

Default:
    1 (= each y-grid line a label is drawn)

Remarks:
    This property is always in effect, regardless if ChartYLabelState is VCHART_LABEL_USER or VCHART_LABEL_AUTO.

Example:

```
VpeSetChartYGridStep(hDoc, 1)
```

means, at each y-grid line a label is drawn at the y-axis.

```
VpeSetChartYGridStep(hDoc, 2)
```

means, that a label is only drawn each 2nd y-grid line.

19.46 VpeSetChartYLabelDivisor

(Chart Property) Sets the divisor for y-labels, if the y-labels are drawn automatically by VPE (ChartYLabelState = VCHART_LABEL_AUTO).

void VpeSetChartYLabelDivisor(
VpeHandle *hDoc*,
double *ylabeldivisor*
)

VpeHandle hDoc
Document Handle

double ylabeldivisor
the divisor for automatically drawn y-labels

Default:
1

Example:
Imagine, the ChartData object consists of one column with the values 1000, 2000, 3000

for ChartYLabelDivisor = 1 the y-labels are drawn as: "1000, 2000, 3000"

for ChartYLabelDivisor = 1000 the y-labels are drawn as: "1, 2, 3"

19.47 VpeSetChartGridRotation

(Chart Property) Sets the grid rotation (clockwise) for the next created chart object. The chart grid (i.e. the chart itself, not the title or legend, etc.) can be rotated by 90 degrees to the right only. This property has no effect on Pie Charts.

void VpeSetChartGridRotation(
    VpeHandle *hDoc*,
    int *axisangle*
)

VpeHandle hDoc
    Document Handle

int axisangle
    possible values are:

Value	Description
0	0 degrees
900	90 degrees, the chart is rotated by 90 degrees clockwise

Default:
    0 (not rotated)

19.48 VpeSetChartYAxisAngle

(Chart Property) Only 3D charts. Sets the view angle of the y-axis. Values are in 1/10°.

void VpeSetChartYAxisAngle(
VpeHandle *hDoc*,
int *yangle*
)

VpeHandle hDoc
Document Handle

int yangle
possible values are:150 <= yangle <= 780

Default:
300 (= 30 degrees)

Remarks:
With this property you can change the view perspective, it does not rotate the chart itself.

19.49 VpeSetChartXAxisAngle

(Chart Property) Only 3D charts. Sets the view angle of the x-axis. Values are in 1/10°.

void VpeSetChartXAxisAngle(
VpeHandle *hDoc*,
int *xangle*
)

VpeHandle hDoc
Document Handle

int xangle
possible values are: 150 <= xangle <= 780

Default:
450 (= 45 degrees)

Remarks:
With this property you can change the view perspective, it does not rotate the chart itself.

19.50 VpeChart

Creates a chart object at the given position from the given ChartData object. The type of the chart is specified in the parameter chart_type. All current chart property settings apply to the created chart.

The property TextColor - which is the generic foreground color - specifies the color for the following chart elements:

- title, subtitle, footnote
- x- / y-axis titles
- x- / y-labels

void VpeChart(
    VpeHandle *hDoc*,
    VpeCoord *x*,
    VpeCoord *y*,
    VpeCoord *x2*,
    VpeCoord *y2*,
    VpeHandle *hData*,
    int *chart_type*
)

VpeHandle hDoc
    Document Handle

VpeCoord x, y, x2, y2
    position and dimensions of the chart

VpeHandle hData
    handle of ChartData object

int chart_type
    possible values are:

Constant Name	**Value**	**Comment**
VCHART_POINT	0	
VCHART_LINE	1	
VCHART_BAR	2	
VCHART_STACKED_BAR_ABSOLUTE	3	
VCHART_STACKED_BAR_PERCENT	4	
VCHART_3D_BAR	5	
VCHART_3D_STACKED_BAR_ABSOLUTE	6	
VCHART_3D_STACKED_BAR_PERCENT	7	
VCHART_PIE	8	
VCHART_3D_PIE	9	
VCHART_AREA_ABSOLUTE	10	
VCHART_AREA_PERCENT	11	

Remarks:

VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

Example:

```
VpeHandle hData
hData = VpeChartDataCreate(hDoc, 2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

```
VpeChartDataAddValue(hDoc, hData, 0, 10)
VpeChartDataAddValue(hDoc, hData, 0, 20)
VpeChartDataAddValue(hDoc, hData, 0, 30)
VpeChartDataAddValue(hDoc, hData, 0, 40)

VpeChartDataAddValue(hDoc, hData, 1, 5)
VpeChartDataAddValue(hDoc, hData, 1, 10)
VpeChartDataAddValue(hDoc, hData, 1, 15)
VpeChartDataAddValue(hDoc, hData, 1, 20)
VpeChartDataAddLegend(hDoc, hData, "Apples")
VpeChartDataAddLegend(hDoc, hData, "Bananas")

VpeSetChartXLabelState(hDoc, VCHART_LABEL_USER)
VpeChartDataAddXLabel(hDoc, hData, "1. Quarter")
VpeChartDataAddXLabel(hDoc, hData, "2. Quarter")
VpeChartDataAddXLabel(hDoc, hData, "3. Quarter")
VpeChartDataAddXLabel(hDoc, hData, "4. Quarter")
```

Now the internal data table of the ChartData object will look like this:

	Column 0 "Apples"	Column 1 "Bananas"
row 0 "1. Quarter"	10	5
row 1 "2. Quarter"	20	10
row 2 "3. Quarter"	30	15
row 3 "4. Quarter"	40	20

Now some different charts are created from the same ChartData object:

```
VpeSetBkgMode(hDoc, VBKG_GRD_LINE)
VpeSetChartTitle(hDoc, "Bar Chart")
VpeChart(hDoc, 1, 1, -18, -18, hData, VCHART_BAR)
VpePageBreak(hDoc)

VpeSetChartTitle(hDoc, "3-D Bar Chart")
VpeChart(hDoc, 1, 1, -18, -18, hData, VCHART_3D_BAR)
VpePageBreak(hDoc)

VpeSetChartTitle(hDoc, "3-D Pie Chart")
VpeSetChartSubTitle(hDoc, "Row 1")
VpeSetChartRow(hDoc, 0)
VpeChart(hDoc, 1, 1, -18, -18, hData, VCHART_3D_PIE)
VpePageBreak(hDoc)

VpeSetChartTitle(hDoc, "3-D Pie Chart")
VpeSetChartSubTitle(hDoc, "Row 2")
VpeSetChartRow(hDoc, 1)
VpeChart(hDoc, 1, 1, -18, -18, hData, VCHART_3D_PIE)
```

20 FormFields

20.1 FormField

[Enterprise Edition and above]

You are familiar to such kind of fields from the many paper forms that are used today. For an overview and explanation, please see see "FormFields" in the Programmer's Manual

20.2 VpeFormField

Creates a Form Field at the given position, using the number of character cells specified with VpeSetCharCount(). Accordingly, each character cell has the

width = ((x2 - x) / CharacterCount) - max(DividerPenSize, AltDividerPenSize)

Example:
If x = 1 and x2 = 11, then the total width of the Form Field is 11cm - 1cm = 10cm. If CharacterCount = 10, then each character cell is 1 cm wide. Afterwards, VPE will subtract the pen size of the Divider or Alternative Divider - whichever is thicker - from the available cell width.

Additionally, the Form Field can compute a best matching font size itself (see VpeSetFormFieldFlags - VEDIT_FLAG_AUTO_FONTSIZE): since VPE computes the width of a single character cell, it will search for a font size so that the widest character of the font will best fit into the available cell width and height.

If y2 = VFREE, the height of the FormField will be computed accordingly to the height of the currently selected font. Moreover, if y2 = VFREE and VEDIT_FLAG_AUTO_FONTSIZE is used, VPE will compute the height of the Form Field accordingly to the width of a single character cell with the same algorithm as explained above. When computing the height, VPE will also consider the thickness of the bottom line, of course.

VpeCoord VpeFormField(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *s*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions of the FormField

LPCSTR s
the content string of the FormField

Returns:
the bottom y-coordinate generated by the output

Remarks:

- Do not misunderstand FormFields, they can't be used for data input by the user. They are intended to be used to display data.
(The VPE Interactive Edition offers objects for data input.)
- A Form Field can only consist of one single line of text, it can not have multiple lines.
- The text alignment can only be left or right.
- Form Fields can not be rotated.
- Form Fields do not fire AutoBreaks.
- If x2 = VFREE, the FormField will create a normal Plain Text Object as if VpeWrite(Box) was used.
- The bottom line will only be drawn, if the normal PenSize (used for lines, frames, etc.) is zero. If PenSize > 0, a FormField will have instead a surrounding frame (with the thickness of PenSize) as with VpeWriteBox().
- VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

See Also:
VpeRenderFormField
"FormFields" in the Programmer's Manual

20.3 VpeSetCharCount

Sets the maximum number of characters that can be displayed in a FormField. This value equals the number of character cells drawn in a FormField.

[Interactive Edition only:]
If the value of CharCount is negative, an Interactive FormField is painted without dividers as if CharCount was zero, but the absolute value of CharCount specifies the maximum number of characters, which may be entered into the control by the user.
e.g. CharCount = -15 would mean, that a user may only enter up to 15 characters into the Interactive FormField.

void VpeSetCharCount(
  VpeHandle *hDoc*,
  int *count*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

int count
  the number of characters

Default:
  15

Remarks:
  If CharCount is set to zero, the FormField will behave the same as if VpeWrite(Box) was used.

See Also:
  "FormFields" in the Programmer's Manual

20.4 VpeGetCharCount

Returns the maximum number of characters that can be displayed in a FormField. This value equals the number of character cells drawn in a FormField.

[Interactive Edition only:]
If the value of CharCount is negative, an Interactive FormField is painted without dividers as if CharCount was zero, but the absolute value of CharCount specifies the maximum number of characters, which may be entered by the user.
e.g. CharCount = -15 would mean, that a user may only enter up to 15 characters into the Interactive FormField.

```
int  VpeGetCharCount(
    VpeHandle hDoc
)
```

VpeHandle hDoc
Document Handle or VPE Object Handle

Returns:
the character count

See Also:
"FormFields" in the Programmer's Manual

20.5 VpeSetDividerPenSize

Sets the pen size for the dividers of a FormField.

void VpeSetDividerPenSize(
  VpeHandle *hDoc*,
  VpeCoord *pen_size*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

VpeCoord pen_size
  the pen size

Default:
  0.01 (= 0.1mm)

Remarks:
  The pen size influences the available width of each character cell. If the font size is computed automatically (see VpeSetFormFieldFlags), the value for the pen size influences the computed font size.

See Also:
  "FormFields" in the Programmer's Manual

20.6 VpeGetDividerPenSize

Returns the pen size of the dividers of a FormField.

VpeCoord VpeGetDividerPenSize(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the pen size used for dividers

See Also:
    "FormFields" in the Programmer's Manual

20.7 VpeSetDividerPenColor

Sets the pen color for the dividers of a FormField.

void VpeSetDividerPenColor(
  VpeHandle *hDoc*,
  COLORREF *pen_color*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

COLORREF pen_color
  one of the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
  COLOR_BLACK

See Also:
  "FormFields" in the Programmer's Manual

20.8 VpeGetDividerPenColor

Returns the pen color for the dividers of a FormField.

COLORREF VpeGetDividerPenColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the pen color for dividers

See Also:
    "FormFields" in the Programmer's Manual

20.9 VpeSetAltDividerNPosition

Sets the position where Alternative Dividers are drawn instead of normal Dividers.

void VpeSetAltDividerNPosition(
    VpeHandle *hDoc*,
    int *position*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

int position
    the position of Alternative Dividers

Default:
    0 (no Alternative Dividers are drawn)

See Also:
    "FormFields" in the Programmer's Manual

20.10 VpeGetAltDividerNPosition

Returns the position where Alternative Dividers are drawn instead of normal Dividers.

int VpeGetAltDividerNPosition(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the position where Alternative Dividers are drawn

See Also:
    "FormFields" in the Programmer's Manual

20.11 VpeSetAltDividerPenSize

Sets the pen size for the Alternative Dividers of a FormField.

void VpeSetAltDividerPenSize(
    VpeHandle *hDoc*,
    VpeCoord *pen_size*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

VpeCoord pen_size
    the pen size

Default:
    0.03 (= 0.3mm)

Remarks:
    The pen size influences the available width of each character cell. If the font size is computed automatically (see VpeSetFormFieldFlags), the value for the pen size influences the computed font size.

See Also:
    "FormFields" in the Programmer's Manual

20.12 VpeGetAltDividerPenSize

Returns the pen size for the Alternative Dividers of a FormField.

VpeCoord VpeGetAltDividerPenSize(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the pen size for Alternative Dividers

See Also:
    "FormFields" in the Programmer's Manual

20.13 VpeSetAltDividerPenColor

Sets the pen color for the Alternative Dividers of a FormField.

void VpeSetAltDividerPenColor(
    VpeHandle *hDoc*,
    COLORREF *pen_color*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

COLORREF pen_color
    one of the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
    COLOR_BLACK

See Also:
    "FormFields" in the Programmer's Manual

20.14 VpeGetAltDividerPenColor

COLORREF VpeGetAltDividerPenColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the pen color for Alterantive Dividers

See Also:
    "FormFields" in the Programmer's Manual

20.15 VpeSetBottomLinePenSize

Sets the pen size for the bottom line. A bottom line is only drawn, if the global PenSize (used for lines, frames, etc.) is set to zero. Otherwise a framed FormField will be drawn similar to VpeWriteBox().

void VpeSetBottomLinePenSize(
VpeHandle *hDoc*,
VpeCoord *pen_size*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

VpeCoord pen_size
the pen size for the bottom line

Default:
0.03 (= 0.3 mm)

See Also:
"FormFields" in the Programmer's Manual

20.16 VpeGetBottomLinePenSize

Returns the pen size of the bottom line.

VpeCoord VpeGetBottomLinePenSize(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the pen size for the bottom line

See Also:
    "FormFields" in the Programmer's Manual

20.17 VpeSetBottomLinePenColor

Sets the pen color for the bottom line. A bottom line is only drawn, if the global PenSize is set to zero. Otherwise a framed FormField will be drawn similar to VpeWriteBox().

void VpeSetBottomLinePenColor(
VpeHandle *hDoc*,
COLORREF *pen_color*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

COLORREF pen_color
one of the "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
COLOR_BLACK

See Also:
"FormFields" in the Programmer's Manual

20.18 VpeGetBottomLinePenColor

Returns the pen color of the bottom line.

COLORREF VpeGetBottomLinePenColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the pen color of the bottom line

See Also:
    "FormFields" in the Programmer's Manual

20.19 VpeSetDividerStyle

Sets the paint-style for dividers.

void VpeSetDividerStyle(
VpeHandle *hDoc*,
long *style*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

long style
the style of dividers, possible values are:

Constant Name	Value	Comment
VFF_STYLE_NONE	0	no dividers
VFF_STYLE_1_4	1	1/4 height
VFF_STYLE_1_3	2	1/3 height
VFF_STYLE_1_2	3	1/2 height (default)
VFF_STYLE_2_3	4	2/3 height
VFF_STYLE_3_4	5	3/4 height
VFF_STYLE_1_1	6	full height (default for AltDivider)

Default:
VFF_STYLE_1_2, dividers are drawn at ½ height of the FormField's height

See Also:
"FormFields" in the Programmer's Manual

20.20 VpeGetDividerStyle

Returns the paint-style of dividers.

long VpeGetDividerStyle(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the paint-style of dividers

See Also:
    "FormFields" in the Programmer's Manual

20.21 VpeSetAltDividerStyle

Sets the paint-style for Alternative Dividers.

void VpeSetAltDividerStyle(
VpeHandle *hDoc*,
long *style*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

long style
the style of Alternative Dividers, possible values are:

Constant Name	Value	Comment
VFF_STYLE_NONE	0	no dividers
VFF_STYLE_1_4	1	1/4 height
VFF_STYLE_1_3	2	1/3 height
VFF_STYLE_1_2	3	1/2 height (default)
VFF_STYLE_2_3	4	2/3 height
VFF_STYLE_3_4	5	3/4 height
VFF_STYLE_1_1	6	full height (default for AltDivider)

Default:
VFF_STYLE_1_1, Alternative Dividers are drawn at the full height of the FormField's height

See Also:
"FormFields" in the Programmer's Manual

20.22 VpeGetAltDividerStyle

Returns the paint-style for Alternative Dividers.

long VpeGetAltDividerStyle(
  VpeHandle *hDoc*
)

VpeHandle hDoc
  Document Handle or VPE Object Handle

Returns:
  the paint-style for Alternative Dividers

See Also:
  "FormFields" in the Programmer's Manual

20.23 VpeSetFormFieldFlags

Sets additional control flags for FormFields.

void VpeSetFormFieldFlags(
    VpeHandle *hDoc*,
    long *flags*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

long flags
    possible values are any combination of:

Constant Name	Value	Comment
VFF_FLAG_DIV_FIRST	1	first divider is painted (only, if no frame)
VFF_FLAG_DIV_LAST	2	last divider is painted (only, if no frame)
VFF_FLAG_ALT_FIRST	4	(default) first divider is painted as AltDivider (overrides _DIV_FIRST) (only, if no frame)
VFF_FLAG_ALT_LAST	8	(default) last divider is painted as AltDivider (overrides _DIV_LAST) (only, if no frame)
VFF_FLAG_AUTO_FONTSIZE	16	(default) font size is computed automatically

Default:
    VFF_FLAG_ALT_FIRST + VFF_FLAG_ALT_LAST + VFF_FLAG_AUTO_FONTSIZE

Remarks:
    You can use combinations of the above flags by adding their values. Please note that if you use VFF_FLAG_ALT_FIRST, VFF_FLAG_DIV_FIRST is ignored. Also if you use VFF_FLAG_ALT_LAST, VFF_FLAG_DIV_LAST is ignored.

    If you specify VFF_FLAG_AUTO_FONTSIZE, the font size for the FormField is computed automatically according to the available space for a single character. See VpeFormField for details.

See Also:
    "FormFields" in the Programmer's Manual

20.24 VpeGetFormFieldFlags

Returns the additional control flags of FormFields.

long VpeGetFormFieldFlags(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    the additional control flags of FormFields

See Also:
    "FormFields" in the Programmer's Manual

21 Templates

21.1 Templates

[Enterprise Edition and above]

This section describes the methods and properties related to Templates.

See Also:

dycodoc Template Processing in the Programmer's Manual

21.2 VpeLoadTemplate

Loads a template file and creates a Template Object from it in memory.

The function returns a handle of the template which has to be used as hTemplate-parameter in the other template-processing functions.

Using the VPE API, you can query the layout- and data source structure from a template. In addition you are able to perform several operations and modifications on a Template Object by code, including the assignment of values to fields (i.e. variables).

The parameter hDoc must be a valid VPE document handle. The template is logically linked to the specified VPE document and it will be removed automatically from memory when the VPE document is closed. Of course the template will be removed latest from memory when your application terminates.

VpeHandle VpeLoadTemplate(
    VpeHandle *hDoc*,
    LPCSTR *file_name*
)

VpeHandle hDoc
    Document Handle

LPCSTR file_name
    the path and file name of the file to load

Returns:
    Handle of the Template Object, if the template file was found and could be loaded.
    Otherwise NULL is returned and LastError is set.

Remarks:
    In case of an error, LastError is set accordingly to the status of the load process.

    If the template has assigned an *Authenticity Key*, this method will fail.
    Use LoadTemplateAuthKey() for loading templates with an *Authenticity Key.*

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.3 VpeLoadTemplateAuthKey

This method is identical to LoadTemplate(), except that it validates the *Authenticity Key* which is assigned to a template by using *dycodoc*.

Loads a template file and creates a Template Object from it in memory.

The function returns a handle of the template which has to be used as hTemplate-parameter in the other template-processing functions.

Using the VPE API, you can query the layout- and data source structure from a template. In addition you are able to perform several operations and modifications on a Template Object by code, including the assignment of values to fields (i.e. variables).

The parameter hDoc must be a valid VPE document handle. The template is logically linked to the specified VPE document and it will be removed automatically from memory when the VPE document is closed. Of course the template will be removed latest from memory when your application terminates.

VpeHandle VpeLoadTemplateAuthKey(
VpeHandle *hDoc*,
LPCSTR *file_name*,
long *key1*,
long *key2*,
long *key3*,
long *key4*
)

VpeHandle hDoc
Document Handle

LPCSTR file_name
the path and file name of the file to load

long key1, key2, key3, key4
the Authenticity Key

Returns:
Handle of the Template Object, if the template file was found, could be loaded and the Authenticity Key validation was successful, i.e. the Authenticity Key is present and checks ok.

Otherwise NULL is returned and LastError is set.

Remarks:
In case of an error, LastError is set accordingly to the status of the load process.

See Also:
dycodoc Template Processing in the Programmer's Manual
Validating the Template Authenticity Key in the Programmer's Manual

21.4 VpeDumpTemplate

After the values for all required Fields have been set, the whole template is dumped into the current and all following pages of the VPE document with this method.

int VpeDumpTemplate(
  VpeHandle *hDoc*,
  VpeHandle *hTemplate*
)

VpeHandle hDoc
  Document Handle into which the template shall be dumped

VpeHandle hTemplate
  Template Handle of the template that shall be dumped

Returns:
  Error Status, this is either VERR_OK if no error occurred, or any of the VERR_xyz values.

Remarks:
  The DumpTemplate()-methods do not make any differences between the AutoBreakMode AUTO_BREAK_ON and AUTO_BREAK_FULL, i.e. objects are **always** inserted into the current VPE Document with the given left and right coordinates if one of either modes is set.

  In case of an error, LastError is set.

  A template may only be dumped into the VPE document into which it was loaded, otherwise LastError will be set to VERR_TPL_OWNERSHIP.

  [Interactive Edition only]
  A template which contains Controls can only be dumped once into a VPE Document. If you wish to dump it more than once into one and the same VPE Document, you need to load the same template as often into memory as you want to dump it. Otherwise *DumpTemplate()* will return the error code VERR_TPL_PAGE_ALREADY_DUMPED. A template which does not contain Controls can be dumped as often into one and the same VPE Document as you like.

See Also:
  dycodoc Template Processing in the Programmer's Manual

21.5 VpeDumpTemplatePage

After the values for all required Fields have been set, the template page specified by the parameter "page" is dumped into the current and - if an auto break occurs - all following pages of the VPE document with this method.

int VpeDumpTemplatePage(
VpeHandle *hDoc*,
VpeHandle *hTemplate*,
int *page*
)

VpeHandle hDoc
Document Handle into which the template shall be dumped

VpeHandle hTemplate
Template Handle of the template that shall be dumped

int page
The page of the template that shall be dumped, may be in the range of
0 (= first page) to PageCount - 1 (= last page).

Returns:
Error Status, this is either VERR_OK if no error occurred, or any of the VERR_xyz values.

Remarks:
The DumpTemplate()-methods do not make any differences between the AutoBreakMode AUTO_BREAK_ON and AUTO_BREAK_FULL, i.e. objects are **always** inserted into the current VPE Document with the given left and right coordinates if one of either modes is set.

In case of an error, LastError is set.

A template may only be dumped into the VPE document into which it was loaded, otherwise LastError will be set to VERR_TPL_OWNERSHIP.

[Interactive Edition only]
A template which contains Controls can only be dumped once into a VPE Document. If you wish to dump it more than once into one and the same VPE Document, you need to load the same template as often into memory as you want to dump it. Otherwise *DumpTemplate()* will return the error code VERR_TPL_PAGE_ALREADY_DUMPED. A template which does not contain Controls can be dumped as often into one and the same VPE Document as you like.

See Also:
dycodoc Template Processing in the Programmer's Manual

21.6 VpeUseTemplateMargins

Retrieves the margin settings from the specified page out of the template and sets the margins of the VPE document correspondingly for the current page **only**.

This function is useful if you need to copy margin settings from a template to a page of a VPE document, e.g. because you create some pages in addition to the template by code using the VPE API.

void VpeUseTemplateMargins(
    VpeHandle *hDoc*,
    VpeHandle *hTemplate*,
    int *page*
)

VpeHandle hDoc
    Document Handle, the margins of this document will be set

VpeHandle hTemplate
    Template Handle

int page
    The page of the template whose margins shall be retrieved, may be in the range of 0 (= first page) to PageCount - 1 (= last page).

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.7 VpeUseTemplateSettings

Retrieves the whole page settings (page dimensions, margins, orientation, paper bin) from the specified page of the template and sets them correspondingly in the VPE document for the current page **only**.

This function is useful if you need to copy page settings from a template to a page of a VPE document, e.g. because you create some pages in addition to the template by code using the VPE API.

void VpeUseTemplateSettings(
VpeHandle *hDoc*,
VpeHandle *hTemplate*,
int *page*
)

VpeHandle hDoc
Document Handle, the page settings of this document will be set

VpeHandle hTemplate
Template Handle

int page
The page of the template whose page settings shall be retrieved, may be in the range of 0 (= first page) to PageCount - 1 (= last page).

See Also:
dycodoc Template Processing in the Programmer's Manual

21.8 VpeSetTplMaster

If the page settings (page dimensions, margins, orientation, paper bin) of the template shall be used when dumping the whole template or a single page of the template into a VPE document, specify True for the parameter yes_no, False otherwise.

If you are using a template which is not defined as master, strange results can happen while dumping, if the margins or page dimensions of the VPE document differ from the template, because it can cause unwanted page breaks (objects in the template are outside of the VPE document margins, and therefore an auto page break occurs.)

void VpeSetTplMaster(
VpeHandle *hTemplate*,
int *yes_no*
)

VpeHandle hTemplate
Template Handle

int yes_no

Value	Description
True	yes, the template's page settings are used
False	no, the template's page settings are not used

Default:
True = yes, this template is master
(this is the default after a template has been loaded into memory using LoadTemplate())

Remarks:
In dycodoc, objects can not be made dependent if they exist on different pages. Only objects on one and the same page can be made dependent. It is very useful to set Master = False if you wish to have a lot of objects to be dependent, which do not fit onto a single page in dycodoc. Solution: Create a very long page in dycodoc and place the objects on it. Load this page into VPE (which has a smaller page format defined) and set Master of the template to False: the objects that do not fit onto the current page are automatically broken to the next page(s).

See Also:
dycodoc Template Processing in the Programmer's Manual

21.9 VpeGetTplFieldAsString

Returns the value of a Field as string.

int VpeGetTplFieldAsString(
  VpeHandle *hTemplate*,
  LPCSTR *data_source_prefix*,
  LPCSTR *field_name*,
  LPSTR *value*,
  UINT **size*
)

VpeHandle hTemplate
  Template Handle

LPCSTR data_source_prefix
  prefix of the Data Source, this is the unique name of a Data Source as defined in *FieldStudio*

LPCSTR field_name
  name of the Data Source Field

LPSTR value
  pointer to a buffer that receives the string. This parameter can be NULL, if the data is not required, in such case no data is copied

*UINT *size*
  pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
  If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example, the field was not found, or out of memory)

Example:

```
UINT uSize;
char szRecipientCompany[256];
uSize = sizeof(szRecipientCompany);
VpeGetTplFieldAsString(hTpl, "Sample2", "Recipient:Company",
                       szRecipientCompany, &uSize);
```

See Also:

  dycodoc Template Processing in the Programmer's Manual

21.10 VpeSetTplFieldAsString

Sets the value of a Field as string.

int VpeSetTplFieldAsString(
  VpeHandle *hTemplate*,
  LPCSTR *data_source_prefix*,
  LPCSTR *field_name*,
  LPCSTR *value*
)

VpeHandle hTemplate
  Template Handle

LPCSTR data_source_prefix
  prefix of the Data Source, this is the unique name of a Data Source as defined in *FieldStudio*

LPCSTR field_name
  name of the Data Source Field

LPCSTR value
  pointer to a buffer that holds the string that shall be assigned to the Field

Returns:

Value	Description
True	if the field was found and its value could be set
False	otherwise

Example:

```
VpeSetTplFieldAsString(hTpl, "Sample2", "Recipient:Company", "IDEAL
Software");
```

See Also:
  dycodoc Template Processing in the Programmer's Manual

21.11 VpeGetTplFieldAsInteger

Returns the value of a Field as integer.

int VpeGetTplFieldAsInteger(
VpeHandle *hTemplate*,
LPCSTR *data_source_prefix*,
LPCSTR *field_name*,
int **value*
)

VpeHandle hTemplate
Template Handle

LPCSTR data_source_prefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *FieldStudio*

LPCSTR field_name
name of the Data Source Field

*int *value*
pointer to a variable that receives the value of the Field as integer when the function returns

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example if the field contained pure text that could not be converted to an integer value)

Example:

```
VpeGetTplFieldAsInteger(hTpl, "SomeTable", "Year", 1982);
```

See Also:

dycodoc Template Processing in the Programmer's Manual

21.12 VpeSetTplFieldAsInteger

Sets the value of a Field as integer.

int VpeSetTplFieldAsInteger(
VpeHandle *hTemplate*,
LPCSTR *data_source_prefix*,
LPCSTR *field_name*,
int *value*
)

VpeHandle hTemplate
Template Handle

LPCSTR data_source_prefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *FieldStudio*

LPCSTR field_name
name of the Data Source Field

int value
the integer value that shall be assigned to the Field

Returns:

Value	Description
True	if the field was found and its value could be set
False	otherwise

Example:

```
VpeSetTplFieldAsInteger(hTpl, "Sample2", "Recipient:Company Number", 25);
```

See Also:

dycodoc Template Processing in the Programmer's Manual

21.13 VpeClearTplFields

Resets all Fields of the specified template to NULL.

void VpeClearTplFields(
  VpeHandle *hTemplate*
)

VpeHandle hTemplate
  Template Handle

See Also:
  dycodoc Template Processing in the Programmer's Manual

21.14 VpeGetTplDataSourceCount

Returns the total number of Data Sources used within the given template.

long VpeGetTplDataSourceCount(
VpeHandle *hTemplate*
)

VpeHandle hTemplate
Template Handle

Returns:
the total number of Data Sources used within the given template

See Also:
dycodoc Template Processing in the Programmer's Manual

21.15 VpeGetTplDataSourceObject

Returns the handle of a Data Source Object.

VpeHandle VpeGetTplDataSourceObject(
VpeHandle *hTemplate*,
long *data_source_index*
)

VpeHandle hTemplate
Template Handle

long data_source_index
index into the array of available Data Sources
this value must be in the range between 0 and VpeGetTplDataSourceCount() - 1

Returns:
the handle of a Data Source Object, or NULL if data_source_index is out of range

Remarks:
The function can be used to access Data Source Objects directly.

See Also:
dycodoc Template Processing in the Programmer's Manual

21.16 VpeGetTplDataSourcePrefix

Returns the prefix of the Data Source as defined in *FieldStudio*.

void VpeGetTplDataSourcePrefix(
VpeHandle *hTemplate*,
long *data_source_index*,
LPSTR *value*,
UINT **size*
)

VpeHandle hTemplate
Template Handle

long data_source_index
index into the array of available Data Sources
this value must be in the range between 0 and VpeGetTplDataSourceCount() - 1

LPSTR value
Pointer to a buffer that receives the string with the prefix of the Data Source.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Example:

```
UINT uSize;
char szPrefix[256];
uSize = sizeof(szPrefix);
VpeGetTplDataSourcePrefix(hTpl, 0, szPrefix, &uSize);
```

See Also:

dycodoc Template Processing in the Programmer's Manual

21.17 VpeGetTplDataSourceFileName

Returns the file name of the Data Source. This is the name of the *FieldStudio* FLD file.

void VpeGetTplDataSourceFileName(
VpeHandle *hTemplate*,
long *data_source_index*,
LPSTR *value*,
UINT **size*
)

VpeHandle hTemplate
Template Handle

long data_source_index
index into the array of available Data Sources
this value must be in the range between 0 and VpeGetTplDataSourceCount() - 1

LPSTR value
Pointer to a buffer that receives the string with the file name of the Data Source.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Example:

```
UINT uSize;
char szFileName[256];
uSize = sizeof(szFileName);
VpeGetTplDataSourceFileName(hTpl, 0, szFileName, &uSize);
```

See Also:

dycodoc Template Processing in the Programmer's Manual

21.18 VpeGetTplDataSourceDescription

Returns the description of the Data Source as defined in *FieldStudio*.

void VpeGetTplDataSourceDescription(
VpeHandle *hTemplate*,
long *data_source_index*,
LPSTR *value*,
UINT **size*
)

VpeHandle hTemplate
Template Handle

long data_source_index
index into the array of available Data Sources
this value must be in the range between 0 and VpeGetTplDataSourceCount() - 1

LPSTR value
Pointer to a buffer that receives the string with the description of the Data Source.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Example:

```
UINT uSize;
char szDescription[256];
uSize = sizeof(szDescription);
VpeGetTplDataSourceDescription(hTpl, 0, szDescription, &uSize);
```

See Also:

dycodoc Template Processing in the Programmer's Manual

21.19 VpeGetTplFieldCount

Returns the total number of Fields used within the given Data Source of the given template.

long VpeGetTplFieldCount(
    VpeHandle *hTemplate*,
    long *data_source_index*,
)

VpeHandle hTemplate
    Template Handle

long data_source_index
    index into the array of available Data Sources
    this value must be in the range between 0 and VpeGetTplDataSourceCount() - 1

Returns:
    the total number of Fields used within the given Data Source of the given template

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.20 VpeGetTplFieldObject

Returns the handle of a Field Object.

VpeHandle VpeGetTplFieldObject(
    VpeHandle *hTemplate*,
    long *data_source_index*,
    long *field_index*,
)

VpeHandle hTemplate
    Template Handle

long data_source_index
    index into the array of available Data Sources
    this value must be in the range between 0 and VpeGetTplDataSourceCount() - 1

long field_index
    index into the array of available Fields
    this value must be in the range between 0 and VpeGetTplFieldCount() - 1

Returns:
    the handle of a Field Object, or NULL if data_source_index or field_index are out of range

Remarks:
    The function can be used to access Field Objects directly.

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.21 VpeFindTplFieldObject

Searches for a given Field and returns the handle of the Field Object.

VpeHandle VpeFindTplFieldObject(
VpeHandle *hTemplate*,
LPCTSTR *data_source_prefix*,
LPCTSTR *field_name*,
)

VpeHandle hTemplate
Template Handle

LPCTSTR data_source_prefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *FieldStudio*

LPCTSTR field_name
name of the Data Source Field

Returns:
Handle of the Field Object, if the field was found
NULL otherwise

Remarks:
The function can be used to access Field Objects directly.

Example:

```
VpeHandle hField;
hField = VpeFindTplFieldObject(hTpl, "SomeTable", "Year");
if (hField)
   VpeSetFieldAsInteger(hField, 1982);
```

See Also:
dycodoc Template Processing in the Programmer's Manual

21.22 VpeGetTplFieldName

Returns the name of the Field as defined in *FieldStudio*.

void VpeGetTplFieldName(
    VpeHandle *hTemplate*,
    long *data_source_index*,
    long *field_index*,
    LPSTR *value*,
    UINT **size*
)

VpeHandle hTemplate
    Template Handle

long data_source_index
    index into the array of available Data Sources
    this value must be in the range between 0 and VpeGetTplDataSourceCount() - 1

long field_index
    index into the array of available Fields
    this value must be in the range between 0 and VpeGetTplFieldCount() - 1

LPSTR value
    Pointer to a buffer that receives the string with the Field name.
    This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
    Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
    If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Example:

```
UINT uSize;
char szFieldName[256];
uSize = sizeof(szFieldName);
VpeGetTplFieldName(hTpl, 0, szFieldName, &uSize);
```

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.23 VpeGetTplFieldDescription

Returns the description of the Field as defined in *FieldStudio*.

void VpeGetTplFieldDescription(
VpeHandle *hTemplate*,
long *data_source_index*,
long *field_index*,
LPSTR *value*,
UINT **size*
)

VpeHandle hTemplate
Template Handle

long data_source_index
index into the array of available Data Sources
this value must be in the range between 0 and VpeGetTplDataSourceCount() - 1

long field_index
index into the array of available Fields
this value must be in the range between 0 and VpeGetTplFieldCount() - 1

LPSTR value
Pointer to a buffer that receives the string with the description of the Field.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Example:

```
UINT uSize;
char szDescription[256];
uSize = sizeof(szDescription);
VpeGetTplFieldDescription(hTpl, 0, szDescription, &uSize);
```

See Also:

dycodoc Template Processing in the Programmer's Manual

21.24 VpeGetTplPageCount

Returns the number of pages in the given template.

int VpeGetTplPageCount(
VpeHandle *hTemplate*
)

VpeHandle hTemplate
Template Handle

Returns:
the number of pages in the given template.

See Also:
dycodoc Template Processing in the Programmer's Manual

21.25 VpeGetTplPageObject

Returns the handle of a Page Object.

VpeHandle VpeGetTplPageObject(
    VpeHandle *hTemplate*,
    long *page_index*
)

VpeHandle hTemplate
    Template Handle

long page_index
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
    Handle of the Page Object, or NULL if index is out of range

Remarks:
    The function can be used to access Page Objects directly.

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.26 VpeGetTplPageWidth

Returns the width in 0.1mm of the specified template page as defined in *dycodoc*.

VpeCoord VpeGetTplPageWidth(
    VpeHandle *hTemplate*,
    int *page*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
    the width of the specified template page as defined in *dycodoc*

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.27 VpeSetTplPageWidth

Sets the width of the specified template page in 0.1mm.

void VpeSetTplPageWidth(
    VpeHandle *hTemplate*,
    int *page*,
    VpeCoord *width*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

VpeCoord width
    the width

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.28 VpeGetTplPageHeight

Returns the height in 0.1mm of the specified template page as defined in *dycodoc*.

VpeCoord VpeGetTplPageHeight(
VpeHandle *hTemplate*,
int *page*
)

VpeHandle hTemplate
Template Handle

int page
index into the array of available Pages
this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
the height of the specified template page as defined in *dycodoc*

See Also:
dycodoc Template Processing in the Programmer's Manual

21.29 VpeSetTplPageHeight

Sets the height of the specified template page in 0.1mm.

void VpeSetTplPageHeight(
    VpeHandle *hTemplate*,
    int *page*,
    VpeCoord *height*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

VpeCoord height
    the height

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.30 VpeGetTplPageOrientation

Returns the orientation of the specified template page as defined in *dycodoc*.

int VpeGetTplPageOrientation(
    VpeHandle *hTemplate*,
    int *page*,
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
    the orientation of the specified template page as defined in *dycodoc*.

Constant Name	Value	Comment
VORIENT_PORTRAIT	1	
VORIENT_LANDSCAPE	2	

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.31 VpeSetTplPageOrientation

Sets the orientation of the specified template page.

int VpeSetTplPageOrientation(
    VpeHandle *hTemplate*,
    int *page*,
    int *orientation*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

int orientation
    the orientation of the specified template page as defined in *dycodoc*, possible values are:

Constant Name	Value	Comment
VORIENT_PORTRAIT	1	
VORIENT_LANDSCAPE	2	

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.32 VpeGetTplPaperBin

Returns the printer's input paper bin of the specified template page as defined in *dycodoc*.

int VpeGetTplPaperBin(
    VpeHandle *hTemplate*,
    int *page*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
    The printer's input paper bin of the specified template page as defined in *dycodoc.*
    Possible values are:

Constant Name	Value	Comment
VBIN_UNTOUCHED	-1	
VBIN_UPPER	1	
VBIN_ONLYONE	1	
VBIN_LOWER	2	
VBIN_MIDDLE	3	
VBIN_MANUAL	4	
VBIN_ENVELOPE	5	
VBIN_ENVMANUAL	6	
VBIN_AUTO	7	
VBIN_TRACTOR	8	
VBIN_SMALLFMT	9	
VBIN_LARGEFMT	10	
VBIN_LARGECAPACITY	11	
VBIN_CASSETTE	14	

Not all of the bin options are available on every printer. Check the printer's documentation for more specific descriptions of these options.

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.33 VpeSetTplPaperBin

Sets the printer's input paper bin of the specified template page.

void VpeSetTplPaperBin(
VpeHandle *hTemplate*,
int *page*,
int *bin*
)

VpeHandle hTemplate
Template Handle

int page
index into the array of available Pages
this value must be in the range between 0 and VpeGetTplPageCount() - 1

int bin
possible values are:

Constant Name	Value	Comment
VBIN_UNTOUCHED	-1	
VBIN_UPPER	1	
VBIN_ONLYONE	1	
VBIN_LOWER	2	
VBIN_MIDDLE	3	
VBIN_MANUAL	4	
VBIN_ENVELOPE	5	
VBIN_ENVMANUAL	6	
VBIN_AUTO	7	
VBIN_TRACTOR	8	
VBIN_SMALLFMT	9	
VBIN_LARGEFMT	10	
VBIN_LARGECAPACITY	11	
VBIN_CASSETTE	14	

Not all of the bin options are available on every printer. Check the printer's documentation for more specific descriptions of these options.

Remarks:
This property is independent from DevPaperBin. You should always use this function to specify the paper bin.
The value VBIN_UNTOUCHED is a VPE internal constant. It instructs VPE not to change the bin from the setting the current selected device has.
Changing the bin during the print-job doesn’t work with some (buggy) printer drivers. Changing the bin with such drivers for other pages than the very first page might not work. Most printer drivers will work.
Printer drivers are manufactured by vendors independent of IDEAL Software; we make no warranty, implied or otherwise, regarding these product’s performance or reliability.

Example:

```
VpeSetTplPaperBin(hTemplate, 2, VBIN_UPPER)
```

Will instruct the printer during printing, to print the **third** page (pages in the template are counted starting with zero) of the template on the upper paper bin (if available).

See Also:

dycodoc Template Processing in the Programmer's Manual

21.34 VpeGetTplLeftMargin

Returns the position of the left margin in 0.1mm of the specified template page as defined in *dycodoc*.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

VpeCoord VpeGetTplLeftMargin(
    VpeHandle *hTemplate*,
    int *page*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
    the position of the left margin of the specified template page as defined in *dycodoc*

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.35 VpeSetTplLeftMargin

Sets the position of the left margin in 0.1mm of the specified template page.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

void VpeSetTplLeftMargin(
VpeHandle *hTemplate*,
int *page*,
VpeCoord *margin*
)

VpeHandle hTemplate
Template Handle

int page
index into the array of available Pages
this value must be in the range between 0 and VpeGetTplPageCount() - 1

VpeCoord margin
the position of the left margin of the specified template page

See Also:
dycodoc Template Processing in the Programmer's Manual

21.36 VpeGetTplRightMargin

Returns the position of the right margin in 0.1mm of the specified template page as defined in *dycodoc*.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

VpeCoord VpeGetTplRightMargin(
    VpeHandle *hTemplate*,
    int *page*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
    the position of the right margin of the specified template page as defined in *dycodoc*

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.37 VpeSetTplRightMargin

Sets the position of the right margin in 0.1mm of the specified template page.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

void VpeSetTplRightMargin(
    VpeHandle *hTemplate*,
    int *page*,
    VpeCoord *margin*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

VpeCoord margin
    the position of the right margin of the specified template page

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.38 VpeGetTplTopMargin

Returns the position of the top margin in 0.1mm of the specified template page as defined in *dycodoc*.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

VpeCoord VpeGetTplTopMargin(
VpeHandle *hTemplate*,
int *page*
)

VpeHandle hTemplate
Template Handle

int page
index into the array of available Pages
this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
the position of the top margin of the specified template page as defined in *dycodoc*

See Also:
dycodoc Template Processing in the Programmer's Manual

21.39 VpeSetTplTopMargin

Sets the position of the top margin in 0.1mm of the specified template page.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

void VpeSetTplTopMargin(
    VpeHandle *hTemplate*,
    int *page*,
    VpeCoord *margin*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

VpeCoord margin
    the position of the top margin of the specified template page

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.40 VpeGetTplBottomMargin

Returns the position of the bottom margin in 0.1mm of the specified template page as defined in *dycodoc*.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

VpeCoord VpeGetTplBottomMargin(
    VpeHandle *hTemplate*,
    int *page*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
    the position of the bottom margin of the specified template page as defined in *dycodoc*

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.41 VpeSetTplBottomMargin

Sets the position of the bottom margin in 0.1mm of the specified template page.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

void VpeSetTplBottomMargin(
VpeHandle *hTemplate*,
int *page*,
VpeCoord *margin*
)

VpeHandle hTemplate
Template Handle

int page
index into the array of available Pages
this value must be in the range between 0 and VpeGetTplPageCount() - 1

VpeCoord margin
the position of the bottom margin of the specified template page

See Also:
dycodoc Template Processing in the Programmer's Manual

21.42 VpeGetTplVpeObjectCount

Returns the number of VPE Objects in the given page of the given template.

long VpeGetTplVpeObjectCount(
    VpeHandle *hTemplate*,
    int *page*
)

VpeHandle hTemplate
    Template Handle

int page
    index into the array of available Pages
    this value must be in the range between 0 and VpeGetTplPageCount() - 1

Returns:
    the number of VPE Objects in the given page of the given template

See Also:
    dycodoc Template Processing in the Programmer's Manual

21.43 VpeGetTplVpeObject

Returns the handle of a VPE Object from a template page.

VpeHandle VpeGetTplVpeObject(
VpeHandle *hTemplate*,
int *page*,
long *index*
)

VpeHandle hTemplate
Template Handle

int page
index into the array of available Pages

long index
index into the array of available objects

Returns:
the handle of a VPE Object from a template page, or NULL if page or index are out of range

Remarks:
The function can be used to access VPE Objects within the template directly.

See Also:
dycodoc Template Processing in the Programmer's Manual

21.44 VpeFindTplVpeObject

In *dycodoc* a unique name is assigned to each VPE Object. This function searches for a given VPE Object name in the template and returns the handle of the VPE Object.

VpeHandle EXPO VpeFindTplVpeObject(
    VpeHandle *hTemplate*,
    LPCSTR *object_name*
)

VpeHandle hTemplate
    Template Handle

LPCSTR object_name
    name of the VPE Object as defined in *dycodoc*

Returns:
    Handle of the VPE Object, if the object was found
    NULL otherwise

Remarks:
    The function can be used to access VPE Objects within the template directly.

Example:

```
VpeHandle hVPEObject;
hVPEObject = VpeFindTplVpeObject(hTpl, "Text1");
if (hVPEObject)
{
   VpeSetBkgMode(hVPEObject, VBKG_SOLID);
   VpeSetBkgColor(hVPEObject, COLOR_RED);
}
```

The above example searches for the VPE Object named "Text1" in the template and changes its background mode to solid and its color to red.

See Also:
    dycodoc Template Processing in the Programmer's Manual

22 DataSource

22.1 DataSource

[Enterprise Edition and above]

This section describes the methods and properties of the Data Source Object.

See Also:

dycodoc Template Processing in the Programmer's Manual

22.2 VpeGetDataSourcePrefix

Returns the prefix of the Data Source as defined in *FieldStudio*.

void VpeGetDataSourcePrefix(
VpeHandle *hDataSource*,
LPSTR *value*,
UINT **size*
)

VpeHandle hDataSource
Data Source Handle

LPSTR value
Pointer to a buffer that receives the string with the prefix of the Data Source.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example out of memory)

Example:

```
char szPrefix[256];
VpeHandle hDataSource = VpeGetTplDataSourceObject(hTemplate, 0);
UINT uSize = sizeof(szPrefix);
VpeGetDataSourcePrefix(hDataSource, szPrefix, &uSize);
```

See Also:

dycodoc Template Processing in the Programmer's Manual

22.3 VpeGetDataSourceFileName

Returns the file name of the Data Source. This is the name of the *FieldStudio* FLD file.

void VpeGetDataSourceFileName(
    VpeHandle *hDataSource*,
    LPSTR *value*,
    UINT **size*
)

VpeHandle hDataSource
    Data Source Handle

LPSTR value
    Pointer to a buffer that receives the string with the file name of the Data Source.
    This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
    Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
    If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example out of memory)

Example:

```
char szPrefix[256];
VpeHandle hDataSource = VpeGetTplDataSourceObject(hTemplate, 0);
UINT uSize = sizeof(szPrefix);
VpeGetDataSourceFileName(hDataSource, szFileName, &uSize);
```

See Also:
    dycodoc Template Processing in the Programmer's Manual

22.4 VpeGetDataSourceDescription

Returns the description of the Data Source as defined in *FieldStudio*.

void VpeGetDataSourceDescription(
VpeHandle *hDataSource*,
LPSTR *value*,
UINT **size*
)

VpeHandle hDataSource
Data Source Handle

LPSTR value
Pointer to a buffer that receives the string with the description of the Data Source.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example out of memory)

Example:

```
char szPrefix[256];
VpeHandle hDataSource = VpeGetTplDataSourceObject(hTemplate, 0);
UINT uSize = sizeof(szPrefix);
VpeGetDataSourceDescription(hDataSource, szDescription, &uSize);
```

See Also:
dycodoc Template Processing in the Programmer's Manual

22.5 VpeGetDataSourceFieldCount

Returns the total number of Fields used within the given Data Source.

long VpeGetDataSourceFieldCount(
    VpeHandle *hDataSource*
)

VpeHandle hDataSource
    Data Source Handle

Returns:
    the total number of Fields used within the given Data Source

See Also:
    dycodoc Template Processing in the Programmer's Manual

22.6 VpeGetDataSourceFieldObject

Returns the handle of a Field Object.

VpeHandle VpeGetDataSourceFieldObject(
    VpeHandle *hDataSource*,
    long *field_index*
)

VpeHandle hDataSource
    Data Source Handle

long field_index
    index into the array of available Fields
    this value must be in the range between 0 and VpeGetTplFieldCount()- 1

Returns:
    the handle of a Field Object, or NULL if data_source_index or field_index are out of range

Remarks:
    The function can be used to access Field Objects directly.

See Also:
    dycodoc Template Processing in the Programmer's Manual

23 Field

23.1 Field

[Enterprise Edition and above]

This section describes the methods and properties of the Field Object.

See Also:

dycodoc Template Processing in the Programmer's Manual

23.2 VpeGetFieldAsString

Returns the value of a Field as string.

int VpeGetFieldAsString(
    VpeHandle *hField*,
    LPSTR *value*,
    UINT **size*
)

VpeHandle hField
    Field Handle

LPSTR value
    pointer to a buffer that receives the string. This parameter can be NULL, if the data is not required, in such case no data is copied

*UINT *size*
    pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
    If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example out of memory)

Remarks:

The Template Object itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See VpeGetTplFieldAsString.

Example:

```
UINT uSize;
char szRecipientCompany[256];
uSize = sizeof(szRecipientCompany);
VpeGetFieldAsString(hField, szRecipientCompany, &uSize);
```

See Also:

dycodoc Template Processing in the Programmer's Manual

23.3 VpeSetFieldAsString

Sets the value of a Field as string.

int VpeSetFieldAsString(
    VpeHandle *hField*,
    LPCSTR *value*,
)

VpeHandle hField
    Field Handle

LPCSTR value
    pointer to a buffer that holds the string that shall be assigned to the Field

Returns:

Value	Description
True	if the field's value could be set
False	otherwise

Remarks:
The Template Object itself offers a simple method to access a Field's value directly, i.e. without retrieving a Field Object first. See VpeSetTplFieldAsString.

Example:

```
VpeSetFieldAsString(hField, "IDEAL Software");
```

See Also:
dycodoc Template Processing in the Programmer's Manual

23.4 VpeGetFieldAsInteger

Returns the value of a Field as integer.

int VpeGetFieldAsInteger(
  VpeHandle *hField*,
  int **value*
)

VpeHandle hField
  Field Handle

*int *value*
  pointer to a variable that receives the value of the Field as integer when the function returns

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example if the field contained pure text that could not be converted to an integer value)

Remarks:
  The Template Object itself offers a simple method to access a Field's value directly, i.e. without retrieving a Field Object first. See VpeGetTplFieldAsInteger.

Example:

```
int value;
VpeGetFieldAsInteger(hField, &x);
```

See Also:
  dycodoc Template Processing in the Programmer's Manual

23.5 VpeSetFieldAsInteger

Sets the value of a Field as integer.

int VpeSetFieldAsInteger(
  VpeHandle *hField*,
  int *value*
)

VpeHandle hField
  Field Handle

int value
  the integer value that shall be assigned to the Field

Returns:

Value	Description
True	if the field's value could be set
False	otherwise

Remarks:
The Template Object itself offers a simple method to access a Field's value directly, i.e. without retrieving a Field Object first. See VpeSetTplFieldAsInteger.

Example:

```
VpeSetFieldAsInteger(hField, 1982);
```

See Also:
dycodoc Template Processing in the Programmer's Manual

23.6 VpeGetFieldName

Returns the name of the Field as defined in *FieldStudio*.

void VpeGetFieldName(
VpeHandle *hField*,
LPSTR *value*,
UINT **size*
)

VpeHandle hField
Field Handle

LPSTR value
Pointer to a buffer that receives the string with the Field name.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example out of memory)

Example:

```
UINT uSize;
char szFieldName[256];
uSize = sizeof(szFieldName);
VpeGetFieldName(hField, szFieldName, &uSize);
```

See Also:
dycodoc Template Processing in the Programmer's Manual

23.7 VpeGetFieldDescription

Returns the description of the Field as defined in *FieldStudio*.

void VpeGetFieldDescription(
VpeHandle *hField*,
LPSTR *value*,
UINT **size*
)

VpeHandle hField
Field Handle

LPSTR value
Pointer to a buffer that receives the string with the description of the Field.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example out of memory)

Example:

```
UINT uSize;
char szDescription[256];
uSize = sizeof(szDescription);
VpeGetFieldDescription(hField, szDescription, &uSize);
```

See Also:

dycodoc Template Processing in the Programmer's Manual

23.8 VpeGetFieldDataSourceObject

Returns the handle of the Data Source Object where the Field belongs to.

VpeHandle VpeGetFieldDataSourceObject(
    VpeHandle *hField*
)

VpeHandle hField
    Field Handle

Returns:
    the handle of the Data Source Object where the Field belongs to

Remarks:
    The function can be used to access the Field's Data Source Object directly.

See Also:
    dycodoc Template Processing in the Programmer's Manual

24 Template Page

24.1 Template Page

[Enterprise Edition and above]

This section describes the methods and properties of the Page Object.

See Also:

dycodoc Template Processing in the Programmer's Manual

24.2 VpeGetTplPageObjWidth

Returns the width of the specified template page as defined in *dycodoc*.

VpeCoord VpeGetTplPageObjWidth(
    VpeHandle *hPage*
)

VpeHandle hPage
    Page Handle

Returns:
    the width of the specified template page as defined in *dycodoc*

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.3 VpeSetTplPageObjWidth

Sets the width of the specified template page.

void VpeSetTplPageObjWidth(
    VpeHandle *hPage*,
    VpeCoord *width*
)

VpeHandle hPage
    Page Handle

VpeCoord width
    the width

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.4 VpeGetTplPageObjHeight

Returns the height of the specified template page as defined in *dycodoc*.

VpeCoord VpeGetTplPageObjHeight(
VpeHandle *hPage*
)

VpeHandle hPage
Page Handle

Returns:
the height of the specified template page as defined in *dycodoc*

See Also:
dycodoc Template Processing in the Programmer's Manual

24.5 VpeSetTplPageObjHeight

Sets the height of the specified template page.

void VpeSetTplPageObjHeight(
VpeHandle *hPage*,
VpeCoord *height*
)

VpeHandle hPage
Page Handle

VpeCoord height
the height

See Also:
dycodoc Template Processing in the Programmer's Manual

24.6 VpeGetTplPageObjOrientation

Returns the orientation of the specified template page as defined in *dycodoc*.

int VpeGetTplPageObjOrientation(
  VpeHandle *hPage*
)

VpeHandle hPage
  Page Handle

Returns:
  the orientation of the specified template page as defined in *dycodoc*.

Constant Name	Value	Comment
VORIENT_PORTRAIT	1	
VORIENT_LANDSCAPE	2	

See Also:
  dycodoc Template Processing in the Programmer's Manual

24.7 VpeSetTplPageObjOrientation

Sets the orientation of the specified template page.

int VpeSetTplPageObjOrientation(
VpeHandle *hPage*,
int *orientation*
)

VpeHandle hPage
Page Handle

int orientation
the orientation of the specified template page as defined in *dycodoc*, possible values are:

Constant Name	Value	Comment
VORIENT_PORTRAIT	1	
VORIENT_LANDSCAPE	2	

See Also:
dycodoc Template Processing in the Programmer's Manual

24.8 VpeGetTplPageObjPaperBin

Returns the printer's input paper bin of the specified template page as defined in *dycodoc*.

int VpeGetTplPageObjPaperBin(
    VpeHandle *hPage*
)

VpeHandle hPage
    Page Handle

Returns:
    The printer's input paper bin of the specified template page as defined in *dycodoc.*
    Possible values are:

Constant Name	Value	Comment
VBIN_UNTOUCHED	-1	
VBIN_UPPER	1	
VBIN_ONLYONE	1	
VBIN_LOWER	2	
VBIN_MIDDLE	3	
VBIN_MANUAL	4	
VBIN_ENVELOPE	5	
VBIN_ENVMANUAL	6	
VBIN_AUTO	7	
VBIN_TRACTOR	8	
VBIN_SMALLFMT	9	
VBIN_LARGEFMT	10	
VBIN_LARGECAPACITY	11	
VBIN_CASSETTE	14	

Not all of the bin options are available on every printer. Check the printer's documentation for more specific descriptions of these options.

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.9 VpeSetTplPageObjPaperBin

Returns the printer's input paper bin of the specified template page as defined in *dycodoc*.

void VpeSetTplPageObjPaperBin(
VpeHandle *hPage*,
int *bin*
)

VpeHandle hPage
Page Handle

int bin
possible values are:

Constant Name	Value	Comment
VBIN_UNTOUCHED	-1	
VBIN_UPPER	1	
VBIN_ONLYONE	1	
VBIN_LOWER	2	
VBIN_MIDDLE	3	
VBIN_MANUAL	4	
VBIN_ENVELOPE	5	
VBIN_ENVMANUAL	6	
VBIN_AUTO	7	
VBIN_TRACTOR	8	
VBIN_SMALLFMT	9	
VBIN_LARGEFMT	10	
VBIN_LARGECAPACITY	11	
VBIN_CASSETTE	14	

Not all of the bin options are available on every printer. Check the printer's documentation for more specific descriptions of these options.

Remarks:

This property is independent from DevPaperBin. You should always use this function to specify the paper bin.
The value VBIN_UNTOUCHED is a VPE internal constant. It instructs VPE not to change the bin from the setting the current selected device has.
Changing the bin during the print-job doesn't work with some (buggy) printer drivers. Changing the bin with such drivers for other pages than the very first page might not work. Most printer drivers will work.
Printer drivers are manufactured by vendors independent of IDEAL Software; we make no warranty, implied or otherwise, regarding these product's performance or reliability.

Example:

```
VpeSetTplPaperBin(hTemplate, 2, VBIN_UPPER)
```

Will instruct the printer during printing, to print the **third** page (pages in the template are counted starting with zero) of the template on the upper paper bin (if available).

See Also:

dycodoc Template Processing in the Programmer's Manual

24.10 VpeGetTplPageObjLeftMargin

Returns the position of the left margin of the specified template page as defined in *dycodoc*.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

VpeCoord VpeGetTplPageObjLeftMargin(
    VpeHandle *hPage*
)

VpeHandle hPage
    Page Handle

Returns:
    the position of the left margin of the specified template page as defined in *dycodoc*

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.11 VpeSetTplPageObjLeftMargin

Sets the position of the left margin of the specified template page.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

void VpeSetTplPageObjLeftMargin(
    VpeHandle *hPage*,
    VpeCoord *margin*
)

VpeHandle hPage
    Page Handle

VpeCoord margin
    the position of the left margin of the specified template page

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.12 VpeGetTplPageObjRightMargin

Returns the position of the right margin of the specified template page as defined in *dycodoc*.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

VpeCoord VpeGetTplPageObjRightMargin(
    VpeHandle *hPage*
)

VpeHandle hPage
    Page Handle

Returns:
    the position of the right margin of the specified template page as defined in *dycodoc*

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.13 VpeSetTplPageObjRightMargin

Sets the position of the right margin of the specified template page.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

void VpeSetTplPageObjRightMargin(
    VpeHandle *hPage*,
    VpeCoord *margin*
)

VpeHandle hPage
    Page Handle

VpeCoord margin
    the position of the right margin of the specified template page

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.14 VpeGetTplPageObjTopMargin

Returns the position of the top margin of the specified template page as defined in *dycodoc*.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

VpeCoord VpeGetTplPageObjTopMargin(
    VpeHandle *hPage*
)

VpeHandle hPage
    Page Handle

Returns:
    the position of the top margin of the specified template page as defined in *dycodoc*

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.15 VpeSetTplPageObjTopMargin

Sets the position of the top margin of the specified template page.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

void VpeSetTplPageObjTopMargin(
    VpeHandle *hPage*,
    VpeCoord *margin*
)

VpeHandle hPage
    Page Handle

VpeCoord margin
    the position of the top margin of the specified template page

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.16 VpeGetTplPageObjBottomMargin

Returns the position of the bottom margin of the specified template page as defined in *dycodoc*.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

VpeCoord VpeGetTplPageObjBottomMargin(
    VpeHandle *hPage*
)

VpeHandle hPage
    Page Handle

Returns:
    the position of the bottom margin of the specified template page as defined in *dycodoc*

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.17 VpeSetTplPageObjBottomMargin

Sets the position of the bottom margin of the specified template page.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

void VpeSetTplPageObjBottomMargin(
  VpeHandle *hPage*,
  VpeCoord *margin*
)

VpeHandle hPage
  Page Handle

VpeCoord margin
  the position of the bottom margin of the specified template page

See Also:
  dycodoc Template Processing in the Programmer's Manual

24.18 VpeGetTplPageObjVpeObjectCount

Returns the number of VPE Objects in the given template page of the given template.

long VpeGetTplPageObjVpeObjectCount(
    VpeHandle *hPage*
)

VpeHandle hPage
    Page Handle

Returns:
    the number of VPE Objects in the given page of the given template

See Also:
    dycodoc Template Processing in the Programmer's Manual

24.19 VpeGetTplPageObjVpeObject

Returns the handle of a VPE Object from a template page.

VpeHandle VpeGetTplPageObjVpeObject(
 VpeHandle *hPage*,
 long *index*
)

VpeHandle hPage
 Page Handle

long index
 index into the array of available VPE Objects in the page
 this value must be in the range between 0 and VpeGetTplPageObjVpeObjectCount() - 1

Returns:
 the handle of a VPE Object from a template page, or NULL if index is out of range

Remarks:
 The function can be used to access VPE Objects within the template directly.

See Also:
 dycodoc Template Processing in the Programmer's Manual

25 VPE Object

25.1 VPE Object

[Enterprise Edition and above]

This section describes the methods and properties of the VPE Object.

In addition the following properties and methods accept a VPE Object handle:

PenSize, PenStyle, PenColor, SetPen, NoPen

BkgMode, TransparentMode, BkgColor, BkgGradientStartColor, BkgGradientEndColor, BkgGradientTriColor, BkgGradientMiddleColorPosition, BkgGradientMiddleColor, BkgGradientRotation, HatchStyle, HatchColor, CornerRadius

FontName, FontSize, CharSet, TextAlignment, TextBold, TextUnderline, TextUnderlined, TextStrikeOut, TextItalic, TextColor, SelectFont, SetFont, SetFontAttr

Rotation

PictureType, PicturePage, PictureEmbedInDoc, PictureKeepAspect, PictureBestFit, PictureX2YResolution, PictureDrawExact

SetBarcodeParms, BarcodeAlignment, BarcodeMainTextParms, BarcodeAddTextParms, BarcodeAutoChecksum, BarcodeThinBar, BarcodeThickBar

Viewable, Printable, Streamable, Shadowed

UDOlParam

CharCount, DividerPenSize, DividerPenColor, AltDividerNPosition, AltDividerPenSize, AltDividerPenColor, BottomLinePenSize, BottomLinePenColor, DividerStyle, AltDividerStyle, FormFieldFlags

CheckmarkColor

See Also:

dycodoc Template Processing in the Programmer's Manual

25.2 VpeGetObjKind

By using this function, you can identify the type of an object.

long VpeGetObjKind(
 VpeHandle *hObject*
)

VpeHandle hObject
 VPE Object Handle

Returns:
 Possible return codes are:

Constant Name	Value	Comment
VOBJID_NULL	16	not identified (should not occur)
VOBJID_LINE	17	
VOBJID_POLYLINE	18	
VOBJID_FRAME	19	
VOBJID_TEXT	20	
VOBJID_PICTURE	21	
VOBJID_BARCODE	22	
VOBJID_ELLIPSE	23	
VOBJID_PIE	24	
VOBJID_POLYGON	25	
VOBJID_RTF	26	
VOBJID_CHART	27	
VOBJID_UDO	28	
VOBJID_FORMFIELD	29	
VOBJID_RESERVED	30	does not occur
VOBJID_DOCDATA	31	VPE document handle
VOBJID_CTRL_FORMFIELD	32	
VOBJID_CTRL_CHECKBOX	33	
VOBJID_CTRL_RADIOBUTTON	34	
VOBJID_CTRL_RADIOBUTTONGROUP	35	
VOBJID_RESERVED2	36	does not occur
VOBJID_RESERVED3	37	does not occur
VOBJID_DATA_MATRIX	38	
VOBJID_MAXI_CODE	39	
VOBJID_PDF417	40	

VOBJID_AZTEC	41	

Example:

```
// Returns the type name of a given VPE Object-ID:
char *GetObjectKindName(VpeHandle hObj)
{
   switch(VpeGetObjKind(hObj))
   {
   case VOBJID_LINE: return "Line";
   case VOBJID_POLYLINE: return "Polyline";
   case VOBJID_FRAME: return "Frame";
   case VOBJID_TEXT: return "Text";
   case VOBJID_PICTURE: return "Picture";
   case VOBJID_BARCODE: return "Barcode";
   case VOBJID_ELLIPSE: return "Ellipse";
   case VOBJID_PIE: return "Pie";
   case VOBJID_POLYGON: return "Polygon";
   case VOBJID_RTF: return "RTF";
   case VOBJID_CHART: return "Chart";
   case VOBJID_UDO: return "UDO";
   case VOBJID_FORMFIELD: return "FormField";
   case VOBJID_DOCDATA: return "Document Data";
   case VOBJID_CTRL_FORMFIELD: return "FormField Control";
   case VOBJID_CTRL_CHECKBOX: return "Checkbox Control";
   case VOBJID_CTRL_RADIOBUTTON: return "RadioButton Control";
   case VOBJID_CTRL_RADIOBUTTONGROUP: return "RadioButton Group Control";
   case VOBJID_DATA_MATRIX: return "DataMatrix 2D-Barcode";
   case VOBJID_MAXI_CODE: return "MaxiCode 2D-Barcode";
   case VOBJID_PDF417: return "PDF417 2D-Barcode";
   case VOBJID_AZTEC: return "Aztec 2D-Barcode";
   }

   return "NULL";
}
```

See Also:

dycodoc Template Processing in the Programmer's Manual

25.3 VpeGetObjName

In *dycodoc* a unique name is assigned to each VPE Object. This function returns the name of a VPE Object as defined in *dycodoc*. The function works also for Field Objects and Control Objects.

void VpeGetObjName(
VpeHandle *hObject*,
LPSTR *value*,
UINT **size*
)

VpeHandle hObject
VPE Object Handle

LPSTR value
Pointer to a buffer that receives the string with the Field name.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Remarks:
This function does only return correct values for VPE Objects, Field Objects and Control Objects that reside in a template, and which have been inserted into the document using VpeDumpTemplate() / VpeDumpTemplatePage(). If you specify an object handle to a different object, the function does nothing.
The function allows that hObject may be NULL, in this case an empty string is returned.

See Also:
dycodoc Template Processing in the Programmer's Manual

25.4 VpeGetObjText

Returns the text content of a Text object.

void VpeGetObjText(
VpeHandle *hObject*,
LPSTR *value*,
UINT **size*
)

VpeHandle hObject
VPE Object Handle

LPSTR value
Pointer to a buffer that receives the string with the Field name.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Remarks:

This function does not support RTF objects.

The returned string does not contain Fields nor the values of Fields.
See VpeGetObjResolvedText

See Also:

dycodoc Template Processing in the Programmer's Manual

25.5 VpeGetObjResolvedText

Returns the text content of a Text object. If Fields are used within the text, the current values of those Fields are merged into the text.

void VpeGetObjResolvedText(
VpeHandle *hObject*,
LPSTR *value*,
UINT **size*
)

VpeHandle hObject
VPE Object Handle

LPSTR value
Pointer to a buffer that receives the string with the Field name.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Remarks:
This function does not support RTF objects.

See Also:
dycodoc Template Processing in the Programmer's Manual

25.6 VpeGetObjLeft

Returns the left coordinate of the VPE Object in **internal units** (1/10.000 mm).

int VpeGetObjLeft(
    VpeHandle *hObject*
)

VpeHandle hObject
    VPE Object Handle

Returns:
    the left coordinate of the specified VPE object in **internal units**

Remarks:
    You must call VpeInternalToExternalUnit() to compute the coordinates according to the current unit transformation.

    The coordinate is normalized, i.e. the following rule is fulfilled:
        right >= left and bottom >= top

    Please note that all VPE API methods (except VpeLine()) require normalized coordinates.

See Also:
    dycodoc Template Processing in the Programmer's Manual

25.7 VpeGetObjTop

Returns the top coordinate of the VPE Object in **internal units** (1/10.000 mm).

int VpeGetObjTop(
    VpeHandle *hObject*
)

VpeHandle hObject
    VPE Object Handle

Returns:
    the top coordinate of the specified VPE object in **internal units**

Remarks:
    You must call VpeInternalToExternalUnit() to compute the coordinates according to the current unit transformation.

    The coordinate is normalized, i.e. the following rule is fulfilled:
        right >= left and bottom >= top
    Please note that all VPE API methods (except VpeLine()) require normalized coordinates.

See Also:
    dycodoc Template Processing in the Programmer's Manual

25.8 VpeGetObjRight

Returns the right coordinate of the VPE Object in **internal units** (1/10.000 mm).

int VpeGetObjRight(
    VpeHandle *hObject*
)

VpeHandle hObject
    VPE Object Handle

Returns:
    the right coordinate of the specified VPE object in **internal units**

Remarks:
    You must call VpeInternalToExternalUnit() to compute the coordinates according to the current unit transformation.

    The coordinate is normalized, i.e. the following rule is fulfilled:
        right >= left and bottom >= top
    Please note that all VPE API methods (except VpeLine()) require normalized coordinates.

See Also:
    dycodoc Template Processing in the Programmer's Manual

25.9 VpeGetObjBottom

Returns the bottom coordinate of the VPE Object in **internal units** (1/10.000 mm).

int VpeGetObjBottom(
  VpeHandle *hObject*
)

VpeHandle hObject
  VPE Object Handle

Returns:
  the bottom coordinate of the specified VPE object in **internal units**

Remarks:
  You must call VpeInternalToExternalUnit() to compute the coordinates according to the current unit transformation.

  The coordinate is normalized, i.e. the following rule is fulfilled:
    right >= left and bottom >= top
  Please note that all VPE API methods (except VpeLine()) require normalized coordinates.

See Also:
  dycodoc Template Processing in the Programmer's Manual

25.10 VpeGetObjPictureFileName

Returns the file name of a Picture Object.

void VpeGetObjPictureFileName(
VpeHandle *hObject*,
LPSTR *value*,
UINT **size*
)

VpeHandle hObject
VPE Object Handle

LPSTR value
Pointer to a buffer that receives the string with the Field name.
This parameter can be NULL, if the data is not required, in such case no data is copied.

*UINT *size*
Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:
the file name of the picture object

See Also:
dycodoc Template Processing in the Programmer's Manual

25.11 VpeSetObjPictureFileName

Returns the file name of a Picture Object.

void VpeSetObjPictureFileName(
    VpeHandle *hObject*,
    LPCSTR *value*,
)

VpeHandle hObject
    VPE Object Handle

LPCSTR value
    pointer to a string with the new file name

Remarks:
    This function does only work for Picture Objects that reside in a template. If you specify an object handle to a different object, the function does nothing.

See Also:
    dycodoc Template Processing in the Programmer's Manual

25.12 VpeGetObjTemplateObject

Returns the handle to the template object the given object belongs to.

VpeHandle VpeGetObjTemplateObject(
  VpeHandle *hObject*
)

VpeHandle hObject
  VPE Object Handle

Returns:
  the handle to the template object the given object belongs to

Remarks:
  This function does only work for VPE Objects that reside in a template, for Field Objects and Control Objects. If you specify an object handle to a different object, the function returns NULL.

See Also:
  dycodoc Template Processing in the Programmer's Manual

25.13 VpeGetInsertedVpeObjectCount

After a call to VpeDumpTemplate() or VpeDumpTemplatePage() the VPE Objects of a template are dumped into the VPE Document. Depending on the amount of text a VPE Object contains, it might have been split over multiple pages. When an object is split over multiple pages, each splitted object is a new distinct object. This function returns the number of VPE Objects in the document that have been created from the provided template VPE Object.

long VpeGetInsertedVpeObjectCount(
  VpeHandle *hTplVpeObject*
)

VpeHandle hTplVpeObject
  Handle of a VPE Object that resides in a template

Returns:
  the number of VPE Objects in the document that have been created from the provided template VPE Object

Remarks:
  This function does only work for VPE Objects that reside in a template. If you specify an object handle to a different object, the function returns NULL.

See Also:
  dycodoc Template Processing in the Programmer's Manual

25.14 VpeGetInsertedVpeObject

Returns the n-th VPE Object in the document that has been created from the provided template VPE Object.

VpeHandle VpeGetInsertedVpeObject(
    VpeHandle *hTplVpeObject*,
    long *index*
)

VpeHandle hTplVpeObject
    Handle of a VPE Object that resides in a template

long index
    index into the array of available VPE objects in the VPE Document
    this value must be in the range between 0 and VpeGetInsertedVpeObjectCount() - 1

Returns:
    the n-th VPE Object in the document that has been created from the provided template VPE Object

Remarks:
    This function does only work for VPE Objects that reside in a template. If you specify an object handle to a different object, the function returns NULL.

See Also:
    dycodoc Template Processing in the Programmer's Manual

25.15 VpeGetInsertedVpeObjectPageNo

Returns the page number where the n-th VPE Object - that has been created from the provided template VPE Object - has been inserted into the document.

int VpeGetInsertedVpeObjectPageNo(
VpeHandle *hTplVpeObject*,
long *index*
)

VpeHandle hTplVpeObject
Handle of a VPE Object that resides in a template

long index
index into the array of available VPE objects in the VPE Document
this value must be in the range between 0 and VpeGetInsertedVpeObjectCount() - 1

Returns:
the page number where the n-th VPE Object - that has been created from the provided template VPE Object - has been inserted into the document

Remarks:
This function does only work for VPE Objects that reside in a template. If you specify an object handle to a different object, the function returns NULL.

See Also:
dycodoc Template Processing in the Programmer's Manual

25.16 VpeGetNextObject

Returns the handle of the succeeding VPE Object in the document.

VpeHandle EXPO VpeGetNextObject(
  VpeHandle *hObject*
)

VpeHandle hObject
  VPE Object Handle

Returns:
  the handle of the succeeding VPE Object in the document

Remarks:
  This method allows in conjunction with the method VpeGetFirstObject() to iterate through all objects of a page / document.

  The last object of a page will return NULL, i.e. there is no "next object" in the current page.

  This function does only work for VPE Objects that reside in a VPE Document. If you specify an object handle to a different object, the function returns NULL.

See Also:
  dycodoc Template Processing in the Programmer's Manual

26 Interactive Objects

26.1 Interactive Objects

[Interactive Edition only]

This section describes the methods and properties related to Interactive Objects.

See Also:

Interactive Documents in the Programmer's Manual.

26.2 VpeFormFieldControl

Inserts either an Interactive FormField or an Interactive Text control into the document. The value of the property CharCount decides, which type of control is inserted:

If CharCount is > 1:	an *Interactive FormField* control is inserted, and CharCount determines the number of character cells.
If CharCount is = 0:	an *Interactive Text* control is inserted, and an unlimited number of characters (until the visible portion of the control is filled-up) may be entered by the user into the control.
If CharCount is < 0:	an *Interactive Text* control is inserted, and CharCount determines the maximum number of characters which may be entered by the user into the control.

In case an *Interactive FormField* control is inserted, the same rules apply as described for the FormField.

In case an *Interactive Text* control is inserted, the control is editable over multiple lines, instead of a single line as with the *Interactive FormField* control. The control can be filled up with as many characters as will fit into the given rectangle of the object. In addition, the text alignment of *Interactive Text* control may be chosen freely (left, right, centered, justified).

VpeHandle VpeFormFieldControl(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
LPCSTR *s*
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions

LPCSTR text
the editable text displayed in the control (the control works with an internal copy of this text, it does not write to the memory pointed at by *text*)

Returns:
The VPE Object handle of the control. This handle can be used later in your code to identify the object (for example, to set the input focus on it or to retrieve its value) and to change some of its properties.

Remarks:
VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

If you are using a multi-line Interactive Text control, lines are separated with the characters "\r\n". (ASCII Code 13 plus ASCII Code 10)

Example:

```
// Insert an Interactive Text Control, where an unlimited number of
// characters may be entered:
VpeSetCharCount(hDoc, 0);
hControl_1 = VpeFormFieldControl(hDoc, 1, 1, -9, VFREE, "Hello\r\nWorld!");

// Insert an Interactive Text Control, limited to max. 14 characters:
VpeSetCharCount(hDoc, -14);
hControl_2 = VpeFormFieldControl(hDoc, 1, 2, -9, VFREE, "Hello\r\nWorld!");

// Insert an Interactive Form Field Control, limited to max. 14 characters:
VpeSetCharCount(hDoc, 14);
hControl_3 = VpeFormFieldControl(hDoc, 1, 3, -9, VFREE, "Hello World!");
```

See Also:

Interactive Documents in the Programmer's Manual.

26.3 VpeCheckbox

Inserts a Checkbox into the document.

VpeHandle VpeCheckbox(
VpeHandle *hDoc*,
VpeCoord *x*,
VpeCoord *y*,
VpeCoord *x2*,
VpeCoord *y2*,
)

VpeHandle hDoc
Document Handle

VpeCoord x, y, x2, y2
position and dimensions

Returns:
The VPE Object handle of the control. This handle can be used later in your code to identify the object (for example, to set the input focus on it or to retrieve its value) and to change some of its properties.

Remarks:
VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
For details please see "Dynamic Positioning" in the Programmer's Manual.

Example:

```
hControl = VpeCheckbox(hDoc, 1, 1, -0.5, -0.5);
```

See Also:
Interactive Documents in the Programmer's Manual.

26.4 VpeRadioButtonGroup

Inserts a Radiobutton Group into the document. It is required that you insert a Radiobutton Group into the document before inserting Radiobuttons. A Radiobutton Group keeps the associated Radiobuttons together, controls the state switching if a button of the group is clicked and holds the value of the group.

VpeHandle VpeRadioButtonGroup(
    VpeHandle *hDoc*,
    int *default_value*
)

VpeHandle hDoc
    Document Handle

int default_value
    the default value of the group
    E.g. if a Radiobutton Group has three associated Radiobuttons, say button A with the value 1, button B with the value 2 and button C with the value 3 and you set default_value = 2, then the button B will be checked automatically when it is created and associated with this Group.

Returns:
    The VPE Object handle of the control.
    The handle is required to associate Radiobuttons with the Radiobutton Group object.
    In addition the handle can be used later in your code to identify the object (for example, to set the input focus on it or to retrieve its value) and to change some of its properties.

Remarks:
    If you assign a value to a group, which is not defined by any Radiobutton, **no** Radiobutton is selected, i.e. the visual state of the group is undefined. This feature can be used to reflect NULL values, for example from databases.

Example:
    see VpeRadioButton()

See Also:
    Interactive Documents in the Programmer's Manual.

26.5 VpeRadioButton

Inserts a Radiobutton into the document. Before inserting a Radiobutton, you need to insert a Radiobutton Group into the document.

VpeHandle VpeRadioButton(
  VpeHandle *hDoc*,
  VpeHandle *hControlRadioButtonGroup*,
  VpeCoord *x*,
  VpeCoord *y*,
  VpeCoord *x2*,
  VpeCoord *y2*,
  int *value*
)

VpeHandle hDoc
  Document Handle

VpeHandle hControlRadioButtonGroup
  the handle of the Radiobutton Group object you want to associate the Radiobutton with

VpeCoord x, y, x2, y2
  position and dimensions

int value
  the value the Radiobutton shall represent within its group

Returns:
  The VPE Object handle of the control. This handle can be used later in your code to identify the object (for example, to set the input focus on it or to retrieve its value) and to change some of its properties.

Remarks:
  VPE offers several methods to attach an object's position to margins and relative to the position of previously inserted objects. In addition Text, Rich Text and Picture objects are able to compute their dimensions automatically depending on their visual content.
  For details please see "Dynamic Positioning" in the Programmer's Manual.

Example:

```
// Create a Radiobutton Group, where the button which represents
// the value "2" shall be initially selected
VpeHandle hGroup = VpeRadioButtonGroup(hDoc, 2);

// Create three Radiobuttons and associate each with the group
VpeRadioButton(hDoc, hGroup, 1, 1, -0.5, -0.5, 1);
VpePrint(hdoc, nRight(hDoc) + 15, VTOP, "lives in forrest");

VpeRadioButton(hdoc, hGroup, VLEFT, nBottom(hDoc) + 0.5, -0.5, -0.5, 2);
VpePrint(hdoc, nRight(hDoc) + 0.15, VTOP, "lives on boat");

VpeRadioButton(hDoc, hGroup, VLEFT, nBottom(hDoc) + 0.5, -0.5, -0.5, 3);
VpePrint(hDoc, nRight(hDoc) + 0.15, VTOP, "lives in house");
```

See Also:
  Interactive Documents in the Programmer's Manual.

26.6 VpeSetCheckmarkColor

Sets the checkmark color used for Checkboxes and Radiobuttons.

void VpeSetCheckmarkColor(
VpeHandle *hDoc*,
COLORREF *color*
)

VpeHandle hDoc
Document Handle or VPE Object Handle

COLORREF color
one of the predefined "COLOR_xyz" constants described in Programmer's Manual or any RGB value

Default:
COLOR_BLACK

See Also:
Interactive Documents in the Programmer's Manual.

26.7 VpeGetCheckmarkColor

Returns the current checkmark color used by Checkboxes and Radiobuttons.

COLORREF VpeGetCheckmarkColor(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle or VPE Object Handle

Returns:
    The current checkmark color.

See Also:
    Interactive Documents in the Programmer's Manual.

26.8 VpeGetDocContainsControls

Determines, whether the document contains controls (i.e. Interactive Objects) or not.

long VpeGetDocContainsControls(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
True	yes, the document contains controls
False	no

See Also:
    Interactive Documents in the Programmer's Manual.

26.9 VpeSetControlsModified

VPE keeps track, if the content (or value) of any control in the document has been modified. This property allows to explicitly set or reset the modification state of the document.

void VpeSetControlsModified(
    VpeHandle *hDoc*,
    int *yes_no*
)

VpeHandle hDoc
    Document Handle

int yes_no

Value	Description
True	mark the document as modified
False	mark the document as unmodified

See Also:
    Interactive Documents in the Programmer's Manual.

26.10 VpeGetControlsModified

VPE keeps track, if the content (or value) of any control in the document has been modified. This porperty returns the current modification state of the document.

int VpeGetControlsModified(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

Returns:

Value	**Description**
True	yes, at least one of the control's value has been modified
False	no

See Also:
    Interactive Documents in the Programmer's Manual.

26.11 VpeEnableInteraction

By default, a document which contains controls is not automatically editable by the user, i.e. all controls are locked for editing.

Set this property in order to enable or disable the whole document for editing.

A disabled control can not receive the focus, therefore the user can not change the control's value, neither with the keyboard, nor with the mouse. It is still possible to change the values of disabled controls by code.

The interaction for a document can not be enabled, if the document is stream based, i.e. it is opened using VpeOpenDocFile(). Stream based documents are not editable. If you want the user to edit a .VPE document file, open a memory based document (i.e. use VpeOpenDoc()) and read the document file into memory using VpeReadDoc().

int VpeEnableInteraction(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	**Description**
True	enable the whole document for editing
False	disable the whole document for editing

Returns:

Value	**Description**
True	success, interaction could be enabled
False	otherwise, i.e. the document is stream based (VpeOpenDocFile() was used)

See Also:
Interactive Documents in the Programmer's Manual.

26.12 VpeIsInteractionEnabled

Returns the current interaction state of the whole document.

int VpeIsInteractionEnabled(
VpeHandle *hDoc*,
)

VpeHandle hDoc
Document Handle

Returns:

Value	Description
True	yes, interaction for the whole document is enabled
False	no

See Also:
Interactive Documents in the Programmer's Manual.

26.13 VpeSetFocusToFirstControl

The focus is set to the very first control in the document (i.e. the control with the lowest TAB-ID).

int VpeSetFocusToFirstControl(
  VpeHandle *hDoc*,
)

VpeHandle hDoc
  Document Handle

Returns:

Value	Description
True	success, the first control in the Tab-Order received the focus
False	otherwise, for example if every object - or the whole document - is disabled, or the object which currently owns the focus did not allow to remove the focus

See Also:
  Interactive Documents in the Programmer's Manual.

26.14 VpeSetFocusControlByName

This method can only be used, if a template containing controls (i.e. Interactive Objects) has been dumped into the VPE Document. In dycodoc, you can assign a name to each object. The name can be supplied to this method in order to set the focus to a named control.

int VpeSetFocusControlByName(
VpeHandle *hDoc*,
VpeHandle *hTemplate*,
LPCSTR *name*
)

VpeHandle hDoc
Document Handle

VpeHandle hTemplate
Template Handle of the template which was dumped into the VPE Document

LPCSTR name
the name of the control as assigned in dycodoc

Returns:

Value	Description
True	success, the control was found and the focus could be set
False	otherwise, for example if every object - or the whole document - is disabled, or the object which currently owns the focus did not allow to remove the focus

Remarks:
If the parameter *name* is NULL or empty, the focus is removed from the control currently having the focus.

Example:

```
// Set the focus on the control named "Street" in dycodoc:
VpeSetFocusControlByName(hDoc, hTemplate, "Street");
```

See Also:
Interactive Documents in the Programmer's Manual.

26.15 VpeSetFocusControl

Sets the focus to a control.

int VpeSetFocusControl(
VpeHandle *hDoc*,
VpeHandle *hControl*
)

VpeHandle hDoc
Document Handle

VpeHandle hControl
Control Handle - the focus is set to the control identified by this handle

Returns:

Value	Description
True	success, the control was found and the focus could be set
False	otherwise, for example if every object - or the whole document - is disabled, or the object which currently owns the focus did not allow to remove the focus

Remarks:
If the parameter *hControl* is NULL, the focus is removed from the control currently having the focus.

Example:

```
hControl = VpeFormFieldControl(hDoc, 1, 1, -9, VFREE, "Hello World!");
VpeSetFocusControl(hDoc, hControl);
```

See Also:
Interactive Documents in the Programmer's Manual.

26.16 VpeGetFocusControl

Returns the Control Handle of the control which currently has the focus.

VpeHandle VpeGetFocusControl(
  VpeHandle *hDoc*,
)

VpeHandle hDoc
  Document Handle

Returns:
  Returns the Control Handle of the control which currently has the focus.
  If no control owns the focus, NULL is returned.

Example:

```
// The following code retrieves the name of the focused control
// (assuming the control was created from a dycodoc template,
// otherwise controls don't have names assigned to them and NULL is
// returned).
// VpeGetObjName() accepts a Control Handle in place of an Object Handle:

LPSTR control_name[256];
VpeGetObjName(VpeGetFocusControl(hDoc), control_name, sizeof(control_name));
```

See Also:
  Interactive Documents in the Programmer's Manual.

26.17 VpeFindControl

Retrieves a control's handle by the supplied object name. Names can only assigned to objects using dycodoc and templates.

VpeHandle VpeFindControl(
  VpeHandle *hTemplate*,
  LPCSTR *name*
)

VpeHandle hTemplate
  Template Handle of the template which was dumped into the VPE Document

LPCSTR name
  the name of the control as assigned in dycodoc

Returns:
  Returns the Control Handle of the control with the given name.
  If no control with the given name was found, NULL is returned.

Example:

```
// Retrieves the Control Handle of the object named "Street"
// and disables the control
VpeSetControlEnabled(VpeFindControl(hTemplate, "Street"), FALSE);
```

See Also:
  Interactive Documents in the Programmer's Manual.

26.18 VpeGetControlEnabled

Returns the current interaction state of the specified control.

int VpeGetControlEnabled(
VpeHandle *hControl*,
)

VpeHandle hControl
Control Handle

Returns:

Value	Description
True	the control is enabled
False	the control is disabled

Remarks:
The function allows that hControl may be NULL, in this case the function returns False.

See Also:
Interactive Documents in the Programmer's Manual.

26.19 VpeSetControlEnabled

Enables / disables the specified control. A disabled control can not receive the focus, therefore the user can not change the control's value, neither with the keyboard, nor with the mouse. It is still possible to change the values of disabled controls by code.

void VpeSetControlEnabled(
VpeHandle *hControl*,
int *yes_no*
)

VpeHandle hControl
Control Handle

int yes_no

Value	Description
True	enable the control for editing
False	disable the control

Example:

```
// Retrieves the Control Handle of the object named "Street"
// and enables the control
VpeSetControlEnabled(VpeFindControl(hTemplate, "Street"), TRUE);
```

Remarks:
The function allows that hControl may be NULL, in this case the function does nothing.

See Also:
Interactive Documents in the Programmer's Manual.

26.20 VpeGetControlTabIndex

Returns the Tab-Index of the specified control. Each control has assigned a unique Tab-Index. The Tab-Index determines the order in which the user can Tab through the controls: if the user presses the Tab key on the keyboard, VPE will search the whole document for the control with the next higher Tab-Index and set the focus on it. The reverse is done, if the user presses Shift + Tab ("Backtab").

If you are using templates, the Tab-Index can be assigned to each object using dycodoc (in the object's properties dialog). If you are creating Controls by code (e.g. by calling VpeFormFieldControl(), VpeCheckbox(), etc.), VPE assigns automatically a Tab-Index to each object in the order of creation.

long VpeGetControlTabIndex(
    VpeHandle *hControl*,
)

VpeHandle hControl
    Control Handle

Returns:
    The Tab-Index of the specified control.

Remarks:
    The function allows that hControl may be NULL, in this case the function returns 0.

See Also:
    Interactive Documents in the Programmer's Manual.

26.21 VpeSetControlTabIndex

Sets the Tab-Index of the specified control. Each control has assigned a unique Tab-Index. The Tab-Index determines the order in which the user can Tab through the controls: if the user presses the Tab key on the keyboard, VPE will search the whole document for the control with the next higher Tab-Index and set the focus on it. The reverse is done, if the user presses Shift + Tab ("Backtab").

If you are using templates, the Tab-Index can be assigned to each object using dycodoc (in the object's properties dialog). If you are creating the controls by code (e.g. by calling VpeFormFieldControl(),VpeCheckbox(), etc.), VPE assigns automatically a Tab-Index to each object in the order of creation.

You can use this property to modify the Tab-Order of a document. But special care must be taken: make sure that you don't assign one and the same Tab-Index to more than one object. The Tab-Index must be unique for each object in the document.

void VpeSetControlTabIndex(
  VpeHandle *hControl*,
  long *tab_index*
)

VpeHandle hControl
  Control Handle

long tab_index
  the Tab-Index that is assigned to the specified control

Remarks:
  The values 0 and -1 (0xffffffff) are **reserved**. You can not set the Tab-Index to either value, in such case the original Tab-Index remains unchanged.

  The function allows that hControl may be NULL, in this case the function does nothing.

See Also:
  Interactive Documents in the Programmer's Manual.

26.22 VpeGetControlGroupObject

Returns the Control Handle of the group object at which the specified control belongs to.

VpeHandle VpeGetControlGroupObject(
VpeHandle *hControl*,
)

VpeHandle hControl
Control Handle

Returns:
The Control Handle of the group object the specified control belongs to.
If the specified control does not belong to a group, or the supplied object handle does not specify a control, NULL is returned.

Remarks:
The function allows that hControl may be NULL, in this case the function returns NULL.

See Also:
Interactive Documents in the Programmer's Manual.

26.23 VpeGetControlFieldObject

If the specified control was created from a template using dycodoc and DumpTemplate(), its value might be associated with a field. In such case this property returns the associated Field.

VpeHandle VpeGetControlFieldObject(
VpeHandle *hControl*,
)

VpeHandle hControl
Control Handle

Returns:
The handle of the associated Field Object of the specified control.
If the specified control is not associated with a Field, or the supplied object handle does not specify a control, NULL is returned.

Remarks:
The function allows that hControl may be NULL, in this case the function returns NULL.

See Also:
Interactive Documents in the Programmer's Manual.

26.24 VpeGetControlTplVpeObject

If the specified control in the document was created from a template using dycodoc and DumpTemplate(), you can retrieve the corresponding VPE Object of the Template (i.e. this is the VPE Object from which the Control was created) using this property.

VpeHandle VpeGetControlTplVpeObject(
  VpeHandle *hControl*,
)

VpeHandle hControl
  Control Handle

Returns:
  The handle of the corresponding VPE Object in the template from which the specified object in the document was created. If the specified object was not created from a template, NULL is returned.

Remarks:
  The function can be used to access VPE Objects within the template directly.
  Normally, there is no need to use this property.

  The function allows that hControl may be NULL, in this case the function returns NULL.

See Also:
  Interactive Documents in the Programmer's Manual.

26.25 VpeGetControlAsString

Returns the value of the specified control as string.

int VpeGetControlAsString(
VpeHandle *hControl*,
LPSTR *value*,
UINT **size*
)

VpeHandle hControl
Control Handle

LPSTR value
pointer to a buffer that receives the string. This parameter can be NULL, if the data is not required, in such case no data is copied

*UINT *size*
pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the value parameter. When the function returns, this variable contains the number of bytes copied to value - including the size of the terminating null character.
If value is NULL, and size is non-NULL, the function returns True and stores the size of the data, in bytes, in the variable pointed to by size. This lets an application determine the best way to allocate a buffer for the value's data.

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example out of memory)

Remarks:

If the control was created from a template and it has an associated field, you should not use this property. It is - from the programmer's view and source code logic - more efficient to use instead the field's methods and properties in order to access the control's value.
The corresponding property of the field is VpeGetFieldAsString.

Example:

```
UINT uSize;
char szRecipientCompany[256];
hControl = VpeFormFieldControl(hDoc, 1, 1, -9, VFREE, "Hello World!");
uSize = sizeof(szRecipientCompany);
VpeGetControlAsString(hControl, szRecipientCompany, &uSize);
```

See Also:

Interactive Documents in the Programmer's Manual.

26.26 VpeSetControlAsString

Sets the value of the specified control as string.

int VpeSetControlAsString(
VpeHandle *hControl*,
LPCSTR *value*
)

VpeHandle hControl
Control Handle

LPCSTR value
pointer to a buffer that holds the string that shall be assigned to the field

Returns:

Value	Description
True	if the control's value could be set
False	otherwise (for example out of memory)

Remarks:

If the control was created from a template and it has an associated Field, you should not use this property. It is - from the programmer's view and source code logic - more efficient to use instead the field's methods and properties in order to access the control's value.
The corresponding property of the field is VpeSetFieldAsString.

Example:

```
hControl = VpeFormFieldControl(hDoc, 1, 1, -9, VFREE, "Hello World!");
VpeSetControlAsString(hControl, "IDEAL Software");
```

See Also:

Interactive Documents in the Programmer's Manual.

26.27 VpeGetControlAsInteger

Returns the value of the specified control as integer.

int VpeGetControlAsInteger(
VpeHandle *hControl*,
int **value*
)

VpeHandle hControl
Control Handle

*int *value*
pointer to a variable that receives the value of the Control as integer when the function returns

Returns:

Value	Description
True	value retrieved successfully
False	value could not be retrieved (for example if the control contained pure text that could not be converted to an integer value)

Remarks:
If the control was created from a template and it has an associated Field, you should not use this property. It is - from the programmer's view and source code logic - more efficient to use instead the field's methods and properties in order to access the control's value.
The corresponding property of the field is VpeGetFieldAsInteger.

Example:

```
int value;
hControl = VpeFormFieldControl(hDoc, 1, 1, -9, VFREE, "123");
VpeGetControlAsInteger(hControl, &x);
```

See Also:
Interactive Documents in the Programmer's Manual.

26.28 VpeSetControlAsInteger

Sets the value of the specified control as integer.

int VpeSetControlAsInteger(
VpeHandle *hControl*,
int *value*
)

VpeHandle hControl
Control Handle

int value
the integer value that shall be assigned to the Field

Returns:

Value	Description
True	if the control's value could be set
False	otherwise (for example out of memory)

Remarks:
If the control was created from a template and it has an associated field, you should not use this property. It is - from the programmer's view and source code logic - more efficient to use instead the field's methods and properties in order to access the control's value.
The corresponding property of the field is VpeSetFieldAsInteger.

Example:

```
hControl = VpeFormFieldControl(hDoc, 1, 1, -9, VFREE, "Hello World!");
VpeSetControlAsInteger(hControl, 1982);
```

See Also:
Interactive Documents in the Programmer's Manual.

27 PDF Export

27.1 PDF Export

This section describes the methods and properties related to PDF Export.

A VPE Document is exported to the Adobe PDF file format by calling the method VpeWriteDoc().

For important details about PDF Export and the features provided by VPE, please see the Programmer's Manual, chapter "The PDF Export Module".

27.2 VpeSetPDFVersion

Sets the current version compatibility for exported PDF documents.

void VpeSetPDFVersion(
    VpeHandle *hDoc*,
    long *version*
)

VpeHandle hDoc
    Document Handle

long version
    possible values are:

Constant Name	Value	Comment
VPE_PDF_ACROBAT_4	1300	PDF Version 1.3
VPE_PDF_ACROBAT_5	1400	PDF Version 1.4

Default:
    VPE_PDF_ACROBAT_4

Remarks:
    The version compatibility is especially important for the encryption key length (see VpeSetEncryption). When you change this property, the encryption key length is set automatically to 40-bits for VPE_PDF_ACROBAT_4 and to 128-bits for VPE_PDF_ACROBAT_5.

Huge documents with Adobe Acrobat 4.0:
When viewing documents with a large number of pages, all pages after page number 34465 are not displayed and an error is reported. The critical page number may alter, depending on the number of other objects in the document.
In order to have documents correctly displayed in Adobe Acrobat 4.0, always keep the number of pages below 8000

27.3 VpeGetPDFVersion

Returns the current setting of the version compatibility for exported PDF documents.

long VpeGetPDFVersion(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

27.4 VpeSetAuthor

Sets the Author for the document. This value is displayed in Acrobat Reader's Document Properties dialog box.

void VpeSetAuthor(
    VpeHandle *hDoc*,
    LPCSTR *author*
)

VpeHandle hDoc
    Document Handle

LPCSTR author
    the author of the document

Default:
    not set (i.e. empty)

27.5 VpeSetTitle

Sets the Title of the exported document. This value is displayed in Acrobat Reader's Document Properties dialog box. If you do not set this property, VPE sets the title of the PDF document by default to the title you have specified when calling VpeOpenDoc().

void VpeSetTitle(
    VpeHandle *hDoc*,
    LPCSTR *title*
)

VpeHandle hDoc
    Document Handle

LPCSTR title
    the title of the document

Default:
    the title you have specified when calling VpeOpenDoc()

27.6 VpeSetSubject

Sets the Subject of the document. This value is displayed in Acrobat Reader's Document Properties dialog box.

void VpeSetSubject(
VpeHandle *hDoc*,
LPCSTR *subject*
)

VpeHandle hDoc
Document Handle

LPCSTR subject
the subject of the document

Default:
not set (i.e. empty)

27.7 VpeSetKeywords

Sets the Keywords for the document. This value is displayed in Acrobat Reader's Document Properties dialog box.

void VpeSetKeywords(
VpeHandle *hDoc*,
LPCSTR *keywords*
)

VpeHandle hDoc
Document Handle

LPCSTR keywords
the Keywords for the document

Default:
not set (i.e. empty)

27.8 VpeSetCreator

Sets the Creator of the document. This should be the name of your application. The value is displayed in Acrobat Reader's Document Properties dialog box.

void VpeSetCreator(
  VpeHandle *hDoc*,
  LPCSTR *creator*
)

VpeHandle hDoc
  Document Handle

LPCSTR creator
  the Creator of the document

Default:
  not set (i.e. empty)

27.9 VpeGetEmbedAllFonts

Returns the current setting for font embedding.

int VpeGetEmbedAllFonts(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
True	the True-Type fonts that are used in the document are embedded
False	the True-Type fonts that are used in the document are not embedded

27.10 VpeSetEmbedAllFonts

[Not supported by the Community Edition]

Sets the current mode for font embedding. Embedded fonts are stored inside the PDF document. Some fonts can not be embedded, because they have a copy-protection bit set.
In this case, VPE will not embed the font.

void VpeSetEmbedAllFonts(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	the True-Type fonts that are used in the document are embedded
False	the True-Type fonts that are used in the document are not embedded

Default:
False

Remarks:
This property controls the default behaviour of VPE. You can override this property individually for each font by using the method VpeSetFontControl().

VPE does only embed True-Type fonts, it will not embed PostScript fonts.

27.11 VpeGetDocExportPictureResolution

Returns the maximum allowed X and Y DPI resolution for images exported to non-VPE documents (like PDF, HTML, etc.).

int VpeGetDocExportPictureResolution(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

Returns:
    the maximum allowed X and Y resolution in DPI (Dots per Inch)

27.12 VpeSetDocExportPictureResolution

Sets the maximum allowed X and Y DPI resolution for images exported to non-VPE documents (like PDF, HTML, etc.).

Only images of higher resolutions are scaled down to the specified resolution, images that have already lower resolutions than the specified are **not** rescaled.

PDF export: This only applies to images that are managed internally as bitmaps. JPG image files are copied in their original form directly to PDF documents, so they are not rescaled.

The lower the image resolution, the smaller are the created images / documents. But especially for printing, too low image resolutions cause low quality printouts.

void VpeSetDocExportPictureResolution(
  VpeHandle *hDoc*,
  int *dpi_resolution*
)

VpeHandle hDoc
  Document Handle

int dpi_resolution
  the maximum allowed image resolution

Default:
  300 DPI

Remarks:
  If the created documents are mainly used for previewing on the screen - and not for printing - we recommend to use 96 DPI. For printing, a resolution of 300 DPI is recommended.

  VPE can rescale images using different algorithms and qualities (see VpeSetDocExportPictureQuality).

  Non-Windows platforms: this option requires the Professional Edition or higher. For lower editions no down-scaling is performed.

27.13 VpeGetDocExportPictureQuality

[VPE Professional Edition and above]

Returns the current setting for the quality of images exported to non-VPE documents (like PDF, HTML, etc.).

int VpeGetDocExportPictureQuality(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

Returns:
    possible values are:

Constant Name	Value	Comment
PICEXP_QUALITY_NORMAL	0	embedded images will have normal quality [default]
PICEXP_QUALITY_HIGH	1	embedded images will have high quality (using Scale2Gray, only available in VPE Professional Edition or higher)

27.14 VpeSetDocExportPictureQuality

[VPE Professional Edition and above]

Sets the current setting for the quality of images exported to non-VPE documents (like PDF, HTML, etc.).

void VpeSetDocExportPictureQuality(
long *hDoc*,
int *quality*
)

VpeHandle hDoc
Document Handle

int quality
possible values are:

Constant Name	Value	Comment
PICEXP_QUALITY_NORMAL	0	embedded images will have normal quality [default]
PICEXP_QUALITY_HIGH	1	embedded images will have high quality (using Scale2Gray, only available in VPE Professional Edition or higher)

Default:
PICEXP_QUALITY_NORMAL

Remarks:
If an image is scaled to a lower resolution during export of the document (see VpeSetDocExportPictureResolution()), this can either be done using a standard algorithm for stretching a bitmap (PICEXP_QUALITY_NORMAL), or it can be done using the Scale2Gray technology (PICEXP_QUALITY_HIGH), which gives much more accurate results.

27.15 VpeGetFastWebView

[VPE Professional Edition and above]

Returns the current type for exported PDF documents. If the document is exported to PDF and Fast Web View is activated, the document is written as web-optimized PDF file. Adobe calls this 'Linearized PDF', an enhanced PDF file format especially for the Internet. It allows to view any given page on a client site as fast as possible without downloading the whole document from a server.

int VpeGetFastWebView(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
True	Fast Web View (linearization) is turned on
False	Fast Web View (linearization) is turned off

27.16 VpeSetFastWebView

[VPE Professional Edition and above]

Sets the current type for exported PDF documents. If the document is exported to PDF and Fast Web View is activated, the document is written as web-optimized PDF file. Adobe calls this 'Linearized PDF', an enhanced PDF file format especially for the Internet. It allows to view any given page on a client site as fast as possible without downloading the whole document from a server.

This feature is especially useful, if documents are created on a server and are transmitted to clients via a network. The client will be able to view the first page of the transmitted document - and to browse to specific pages in the document - as fast as possible, while the transmission is still in process.

Whilst non-optimized PDF documents are created by VPE in one single pass, where every page is written to the PDF file immediately, web-optimized documents are created internally in two passes: VPE can not write any part of the document, before it has analysed the complete document structure, because it needs to arrange the objects in the PDF document in a very special way according to the Adobe PDF specifications. In addition, some special tables (called "Hint Tables"), which contain information about the document structure, have to be created.

The disadvantage of the optimized mode is, that VPE needs more processor time and memory to perform the optimization. During the first pass, VPE keeps all objects of the document in memory and analyses the document structure. In a second pass VPE writes the document to the file. Depending on the number of simultaneous users and the overall workload of the server, this is no problem, if small documents with a few pages are created.

In case you require to create huge documents simultaneously for many users, you can instruct VPE to use a special swapping technique, so instead of using RAM memory, VPE will swap the information for all the objects in the document to a temporary file. This is done with a special technique and works *very fast.* (see VpeSetUseTempFiles() for details)

We recommend that you monitor the workload of the server. Depending on available free ram during a typical workload scenario, you should decide whether to use the swapping technology or not. Some tests also revealed that - depending on the scenario and contents of the document - the use of temporary files is significantly faster than using memory based document creation!
So our second recommendation is to measure the performance during a typical workload scenario with - and without - swapping, to decide what mode of operation to use.

Each temporary swap file is deleted automatically after a PDF document has been created.

void VpeSetFastWebView(
  VpeHandle *hDoc*,
  int *yes_no*
)

VpeHandle hDoc
  Document Handle

int yes_no

Value	Description
True	create linearized PDF documents for Fast Web View
False	create normal documents

Default:

False

27.17 VpeGetUseTempFiles

[VPE Professional Edition and above]

Returns the current setting whether temporary swap files are used or not.
This property is only in effect, if Fast Web View (see VpeSetFastWebView()) is activated.

int VpeGetUseTempFiles(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
True	temporary swap files are used to generate linearized PDF documents
False	RAM memory is used to generate linearized PDF documents

27.18 VpeSetUseTempFiles

[VPE Professional Edition and above]

Sets the current mode of operation, specifiying whether temporary swap files are used or not. This property is only in effect, if Fast Web View (see VpeSetFastWebView()) is activated.

void VpeSetUseTempFiles(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	temporary swap files are used to generate linearized PDF documents
False	RAM memory is used to generate linearized PDF documents

Default:
False

Remarks:
The temporary swap files are created in the directory specified by the TMP or TEMP environment variable. If both variables are not set, or the specified directory does not exist, the current working directory is used.

Each temporary swap file is deleted automatically after a PDF document has been created.

27.19 VpeGetSubsetAllFonts

[VPE Professional Edition and above]

Returns the current setting for font subsetting.

int VpeGetSubsetAllFonts(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

Returns:

Value	**Description**
True	the True-Type fonts that are used in the document are subsetted
False	the True-Type fonts that are used in the document are not subsetted

27.20 VpeSetSubsetAllFonts

[VPE Professional Edition and above]

Sets the current mode for font subsetting. Only fonts that are embedded, can be subsetted.

Font Subsetting means, that VPE assembles on-the-fly a new font from the source font, that contains only the characters which are used in the document. A subsetted font is in regular much smaller than the original font, which results in significantly smaller documents.

void VpeSetSubsetAllFonts(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	the True-Type fonts that are used in the document are subsetted
False	the True-Type fonts that are used in the document are not subsetted

Default:
False

Remarks:
This property controls the default behaviour of VPE. You can override this property individually for each font by using the method VpeSetFontControl().

VPE does only subset True-Type fonts, it will not subset PostScript fonts.

Please note: Font Subsetting does not work correctly for Adobe Acrobat 4, if you are using character sets which are different from ANSI_CHARSET. So subsetting of all western character sets is supported for Acrobat 4, too.

27.21 VpeSetFontControl

[VPE Professional Edition and above]

Allows to set several parameters for an individual font. This function overrides the settings of VpeSetEmbedAllFonts() and VpeSetSubsetAllFonts().

void VpeSetFontControl(
  VpeHandle *hDoc*,
  LPCSTR *font_name*,
  LPCSTR *subst_font_name*,
  int *do_embed*,
  int *do_subset*
)

VpeHandle hDoc
  Document Handle

LPCSTR font_name
  the font you want to set the parameters for

LPCSTR subst_font_name
  the font with which you want to substitute the original font with

int do_embed

Value	Description
True	if the font is a True-Type font, it will be embedded into the document
False	the font will not be embedded into the document

int do_subset

Value	Description
True	if the font is a True-Type font, and it is embedded, it will be subsetted
False	the font will not be subsetted

Remarks:

You can call this method repeatedly for each font you want to set individual parameters for. The settings can be deleted by calling the method VpeResetFontControl().

Whilst the settings for font substitution of this method are only active during the export to PDF documents, there is a second method to substitute fonts VpeSetFontSubstitution(), which is always active.

For important details about fonts and font handling, please see the Programmer's Manual, chapter "Programming Techniques", subchapter "Fonts and Font Handling".

Example:

```
VpeSetFontControl(hDoc, "Arial", "Helvetica", FALSE, FALSE);
VpeSetFontControl(hDoc, "Times New Roman", "Times", FALSE, FALSE);
```

Instructs VPE to substitute "Arial" with "Helvetica" and "Times New Roman" with "Times".

Since Helvetica as well as Times are PostScript fonts, VPE can neither embed nor subset them. But because both fonts are standard Base 14 PostScript fonts, they are supported by PDF Readers on any platform, so embedding (and therefore subsetting) is not required.

27.22 VpeResetFontControl

[VPE Professional Edition and above]

With each call to VpeSetFontControl(), VPE adds internally a new font control structure to a list. VpeResetFontControl() clears the whole list, so all individual font control settings are deleted.

void VpeResetFontControl(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

27.23 VpeGetEncryption

[VPE Professional Edition and above]

Returns the current setting for encryption.

int VpeGetEncryption(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

Returns:

Value	Description
True	encryption is activated
False	encryption is turned off

27.24 VpeSetEncryption

[VPE Professional Edition and above]

Activates or deactivates encryption for exported PDF documents.

void VpeSetEncryption(
  VpeHandle *hDoc*,
  int *enc_type*
)

VpeHandle hDoc
  Document Handle

int enc_type
  possible values are:

Constant Name	**Value**	**Comment**
DOC_ENCRYPT_NONE	0	no encryption [default]
DOC_ENCRYPT_STREAM	1	use stream encryption (compatible to Acrobat 4 and higher)

Default:
  DOC_ENCRYPT_NONE

Remarks:
  See VpeSetEncryptionKeyLength(), VpeSetUserPassword(), VpeSetOwnerPassword() and VpeSetProtection() for details.

  If one and the same VPE document is exported multiple times to encrypted PDF documents, the size of the created PDF files might differ around 10 - 20 bytes. This is normal, because some encrypted characters - for example in the creation date / time string - need to be converted into a special form with additional escape characters.

27.25 VpeGetEncryptionKeyLength

[VPE Professional Edition and above]

Returns the current encryption key length.

int VpeGetEncryptionKeyLength(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

27.26 VpeSetEncryptionKeyLength

[VPE Professional Edition and above]

Sets the encryption key length for exported PDF documents. For documents which are compatible to Acrobat Reader 4, only a value of 40 is allowed. For documents which are compatible to Acrobat Reader 5, a value between 40 and 128 is allowed. If you want to create documents which are compatible to Acrobat Reader 5 only, we recommend to use 128-bit encryption.

void VpeSetEncryptionKeyLength(
  VpeHandle *hDoc*,
  int *key_length*
)

VpeHandle hDoc
  Document Handle

int key_length
  the key length in bits in the range from 40 to 128;
  the value must be a multiple of 8 (e.g. 40, 48, 56, 64, ..., 128)

Default:

40	if PDFVersion is VPE_PDF_ACROBAT_4 (default)
128	if PDFVersion is VPE_PDF_ACROBAT_5

Remarks:

The User Password (see VpeSetUserPassword()) as well as the Owner Password (see VpeSetOwnerPassword()) are used to create the encryption key. To gain safe encryption, at least one of either passwords should be specified - better both.

As you set the property PDFVersion, the encryption key length is adjusted accordingly and any previously assigned value is overwritten.

27.27 VpeSetUserPassword

[VPE Professional Edition and above]

Sets the user password for the document. In order to be able to open an encrypted document (see VpeSetEncryption()), either this password - or the owner password - must be entered in advance. The document can not be opened, if an incorrect password is supplied (see also VpeSetOwnerPassword()).

If the document does not have a user password, no password is requested; Acrobat Reader can simply open, decrypt, and display the document.

Whether additional operations are allowed on a decrypted document depends on if either the user password or the owner password (if any) was supplied when the document was opened and on any access restrictions that were specified when the document was created.

Opening the document with the correct user password (or opening a document that does not have a user password) allows additional operations to be performed according to the user access permissions specified in the document’s encryption dictionary (see VpeSetProtection()).

void VpeSetUserPassword(
  VpeHandle *hDoc*,
  LPCSTR *user_password*
)

VpeHandle hDoc
  Document Handle

LPCSTR user_password
  A string which may contain any characters. Up to 32 characters of the password are used for generating the encryption key (see VpeSetEncryptionKeyLength()).

Default:
  not set (i.e. empty)

27.28 VpeSetOwnerPassword

[VPE Professional Edition and above]

Sets the Owner Password for the document. Opening the document with the correct owner password (assuming it is not the same as the user password) allows full (owner) access to the document. This unlimited access includes the ability to change the document’s passwords and access permissions (see also VpeSetUserPassword() and VpeSetProtection()).

void VpeSetOwnerPassword(
  VpeHandle *hDoc*,
  LPCSTR *owner_password*
)

VpeHandle hDoc
  Document Handle

LPCSTR owner_password
  A string which may contain any characters. Up to 32 characters of the password are used for generating the encryption key (see VpeSetEncryptionKeyLength()).

Default:
  not set (i.e. empty)

27.29 VpeGetProtection

[VPE Professional Edition and above]

Returns the current document protection flags.

long VpeGetProtection(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

Returns:
    the current document protection flags

27.30 VpeSetProtection

[VPE Professional Edition and above]

Sets the document protection flags. Document protection is only in effect, if encryption (see VpeSetEncryption()) is activated.

void VpeSetProtection(
    VpeHandle *hDoc*,
    long *protection_flags*
)

VpeHandle hDoc
    Document Handle

long protection_flags
    possible values are:

Constant Name	Value	Comment
PDF_ALLOW_NONE	0	full protection, no flag is raised
PDF_ALLOW_PRINT	4	allow to print the document (see also the flag PDF_ALLOW_HIQ_PRINT)
PDF_ALLOW_MODIFY	8	allow to modify the contents of the document by operations other than those controlled by the flags PDF_ALLOW_TA_IFF, PDF_ALLOW_FILL_IFF and PDF_ALLOW_ASSEMBLE
PDF_ALLOW_COPY	16	PDF v1.3: allow to copy or otherwise extract text and graphics from the document, including extracting text and graphics (in support of accessibility to disabled users or for other purposes) PDF v1.4: allow to copy or otherwise extract text and graphics from thedocument by operations other than that controlled by PDF_ALLOW_EXTRACT
PDF_ALLOW_TA_IFF	32	allow to add or modify text annotations, fill in interactive form fields, and - if PDF_ALLOW_MODIFY is also used - create or modify interactive form fields (including signature fields)
PDF_ALLOW_FILL_IFF	256	PDF v1.4 only: allow to fill in existing interactive form fields (including signature fields), even if PDF_ALLOW_TA_IFF is not used
PDF_ALLOW_EXTRACT	512	PDF v1.4 only: allow to extract text and graphics (in support of accessibility to disabled users or for other purposes)
PDF_ALLOW_ASSEMBLE	1024	PDF v1.4 only: allow to assemble the document (insert, rotate, or delete pages and create bookmarks or thumbnail images), even if PDF_ALLOW_MODIFY is not used
PDF_ALLOW_HIQ_PRINT	2048	PDF v1.4 only: allow to print the document to a representation from which a faithful digital copy of the PDF content could be generated. When this flag is NOT specified (and PDF_ALLOW_PRINT is used), printing is limited to a lowlevel representation of the appearance, possibly of degraded quality. Acrobat viewers implement this limited mode of printing as “Print As Image”, except on UNIX systems, where this feature is not available.

PDF_ALLOW_ALL	3900	no protection, all flags are raised

Default:

PDF_ALLOW_NONE

27.31 VpeSetBookmarkDestination

[VPE Professional Edition and above]

Sets the destination for subsequently added bookmarks.

void VpeSetBookmarkDestination(
VpeHandle *hDoc*,
int *bkm_type*,
int *left*,
int *top*,
int *right*,
int *bottom*,
double *zoom_factor*
)

VpeHandle hDoc
Document Handle

int bkm_type
Bookmark Type, possible values are:

Constant Name	**Value**	**Comment**
VBOOKMARK_DEST_NONE	0	No Destination
VBOOKMARK_DEST_LTZ	1	Normal Destination (left, top, zoom are used) - this is the default Display the target page with the coordinates (*left, top*) positioned at the top-left corner of the window and the contents of the page magnified by the *zoom_factor*. A null value for any of the parameters *left, top*, or *zoom_factor* specifies that the current value of that parameter is to be retained unchanged.
VBOOKMARK_DEST_FIT	2	Fit the page in the window (no coordinates used) Display the target page with its contents magnified just enough to fit the entire page within the window both horizontally and vertically. If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the page within the window in the other dimension.
VBOOKMARK_DEST_FITH	3	Fit the page in the window horizontally (top coordinate used) Display the target page with the vertical coordinate *top* positioned at the top edge of the window and the contents of the page magnified just enough to fit the entire width of the page within the window.
VBOOKMARK_DEST_FITV	4	Fit the page in the specified rectangle (all coordinates used) Display the target page with its contents magnified just enough to fit the rectangle specified by the coordinates *left, bottom, right* and *top* entirely within the window both horizontally and vertically. If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the rectangle within the window in the other dimension.
VBOOKMARK_DEST_FITR	5	Fit the page in the specified rectangle (all coordinates used) Display the target page with its contents magnified just

		enough to fit the rectangle specified by the coordinates *left, bottom, right* and *top* entirely within the window both horizontally and vertically. If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the rectangle within the window in the other dimension.
VBOOKMARK_DEST_FITB	6	Fit the page's bounding rectangle in the window Display the target page with its contents magnified just enough to fit its bounding box entirely within the window both horizontally and vertically. If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the bounding box within the window in the other dimension.
VBOOKMARK_DEST_FITBH	7	Fit the page's bounding rectangle horizontally in the window (top coordinate used) Display the target page with the vertical coordinate *top* positioned at the top edge of the window and the contents of the page magnified just enough to fit the entire width of its bounding box within the window.
VBOOKMARK_DEST_FITBV	8	Fit the page's bounding rectangle vertically in the window (left coordinate used) Display the target page with the horizontal coordinate *left* positioned at the left edge of the window and the contents of the page magnified just enough to fit the entire height of its bounding box within the window.

int left, int top, int right, int bottom
position and dimensions

double zoom_factor
zoom factor, e.g.: 0.5 = 50%; 1.0 = 100%; 2.0 = 200%; etc.

Default:
VBOOKMARK_DEST_LTZ

27.32 VpeGetBookmarkStyle

[VPE Professional Edition and above]

Returns the current bookmark style.

int VpeGetBookmarkStyle(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

27.33 VpeSetBookmarkStyle

[VPE Professional Edition and above]

Sets the current bookmark style.

void VpeSetBookmarkStyle(
    VpeHandle *hDoc*,
    int *style*
)

VpeHandle hDoc
    Document Handle

int style
    possible values are any combination of the following flags, the values can be combined by adding there values:

Constant Name	Value	Comment
VBOOKMARK_STYLE_NONE	0	[default]
VBOOKMARK_STYLE_BOLD	1	PDF 1.4 or higher)
VBOOKMARK_STYLE_ITALIC	2	PDF 1.4 or higher
VBOOKMARK_STYLE_OPEN	4	

Default:
    VBOOKMARK_STYLE_NONE

Example:

```
// Bold and Italic styles will work only with PDF 1.4, activate it:
VpeSetPDFVersion(hDoc, VPE_PDF_ACROBAT_5);

// Change the bookmark style to bold and italic:
VpeSetBookmarkStyle(hDoc, VBOOKMARK_STYLE_BOLD + VBOOKMARK_STYLE_ITALIC)
```

27.34 VpeGetBookmarkColor

[VPE Professional Edition and above]

Returns the current bookmark color. Bookmark colors are supported by Acrobat Reader 5.0 (PDF v1.4) or higher.

COLORREF VpeGetBookmarkColor(
    VpeHandle *hDoc*,
)

VpeHandle hDoc
    Document Handle

27.35 VpeSetBookmarkColor

[VPE Professional Edition and above]

Sets the current bookmark color. Bookmark colors are supported by Acrobat Reader 5.0 (PDF v1.4) or higher.

void VpeSetBookmarkColor(
VpeHandle *hDoc*,
COLORREF *color*
)

VpeHandle hDoc
Document Handle

COLORREF color
one of the "COLOR_xyz" constants described in Programmer's or any RGB value

Default:
COLOR_BLACK

27.36 VpeAddBookmark

[VPE Professional Edition and above]

Adds a bookmark to the document: a PDF document may optionally display a document outline on the screen, allowing the user to navigate interactively from one part of the document to another. The outline consists of a tree-structured hierarchy of bookmark items, which serve as a "visual table of contents" to display the document’s structure to the user. Bookmarks are displayed in the left tree-view of Acrobat Reader.

The added bookmark will have the currently assigned destination, style and color (see VpeSetBookmarkDestination(), VpeSetBookmarkStyle(), VpeSetBookmarkColor()).

The target page for the destination is always the currently active page of the VPE document as you call this method.

In order to create a hierarchical tree structure, you can supply a parent bookmark as *parent*, i.e. the added bookmark will be a child of the supplied parent bookmark. To add a bookmark to the top-level of the hierarchy, specify a value of null for the *parent*.

The *VpeAddBookmark* method returns a handle to the newly created bookmark, which can be used as *parent* for subsequent calls.

VpeHandle VpeAddBookmark(
  VpeHandle *hDoc*,
  VpeHandle *parent*,
  LPCSTR *title*
)

VpeHandle hDoc
  Document Handle

VpeHandle parent
  the parent bookmark or null for a top-level bookmark

LPCSTR title
  the title of the bookmark

Returns:
  a handle to the newly created bookmark, which can be used as *parent* for subsequent calls

Remarks:
  Bookmarks are not shown within VPE documents. They are stored internally and are exported to PDF documents only.

  Bookmarks are not stored within VPE document files. If you write a VPE document that contains bookmarks to a file, read the file later into memory and export it to PDF, the bookmarks are omitted. For the same reason, bookmarks will be missing in exported PDF files, if the VPE source document is created stream-based by using VpeOpenDocFile().

Example:

```
// Bold and Italic styles will work only with PDF 1.4, activate it:
VpeSetPDFVersion(hDoc, VPE_PDF_ACROBAT_5);

// Output some text on the current page:
VpePrint(hDoc, 1, 1, "Introduction");

// Set the style for newly added bookmarks to bold, italic, open:
VpeSetBookmarkStyle(hDoc, VBOOKMARK_STYLE_BOLD + VBOOKMARK_STYLE_ITALIC +
                          VBOOKMARK_STYLE_OPEN);

// Set the destination type for newly added bookmarks:
VpeSetBookmarkDestination(hDoc, VBOOKMARK_DEST_FIT, 0, 0, 0, 0, 1);

// Add a new bookmark to the top-level of the hierachy:
VpeHandle ParentBookmark = VpeAddBookmark(hDoc, NULL, "Introduction");

// Append a new page to the VPE document:
VpePageBreak(hDoc);

// Output some text on the current page:
VpePrint(hDoc, 1, 1, "Bookmarks explained");

// Add a new bookmark as child of the previously inserted bookmark.
// It will have the currently adjusted style, color and destination:
VpeAddBookmark(hDoc, ParentBookmark, "Bookmarks explained");
```

27.37 VpeExtIntDA

Used internally for diagnostic tasks. **Do not use!**

LRESULT VpeExtIntDA(
    VpeHandle *hDoc*,
    LPARAM *i*,
    LPARAM *s*
)

28 HTML Export

28.1 HTML Export

This section describes the methods and properties related to HTML Export.

A VPE Document is exported to the HTML file format by calling the method VpeWriteDoc().

The following properties are written as (meta-)tags to generated HTML documents:

Title as <title> tag

Creator as "generator" meta-tag

Author as "author" meta-tag

Keywords as "keywords" meta-tag

Subject as "description" meta-tag

In addition, the settings for DocExportPictureQuality and DocExportPictureResolution are applied to converted bitmap images (i.e. where a source image is not in a browser-compatible format, for example TIFF and is converted to PNG).

Objects which are exported as bitmaps – like barcodes, charts, polygons, etc. – are using the properties DocExportPictureResolution, PictureExportColorDepth and PictureExportDither.

We recommend to set DocExportPictureResolution to 96 DPI for HTML export, which is the resolution for screens. This also creates smaller images (regarding their size in bytes).

For important details about HTML Export and the features provided by VPE, please see the Programmer's Manual, chapter "The HTML Export Module".

28.2 VpeSetHtmlScale

Sets the scaling for HTML output. It is recommended to set the scale below 1.0 so that the generated HTML documents are printable by browsers, which need page space for headers and footers they do insert.

void VpeSetHtmlScale(
  VpeHandle *hDoc*,
  double *scale*
)

VpeHandle hDoc
  Document Handle

double scale
  the scale of the generated HTML document, a scale below 1.0 shrinks the document, a scale above 1.0 expands the document

Default:
  0.75, which equals 75% of the original scale

28.3 VpeGetHtmlScale

Returns the current scaling for HTML output.

double VpeGetHtmlScale(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Returns:
the current scaling for HTML output

28.4 VpeSetHtmlWordSpacing

Sets the spacing between words for HTML output. If the specified HTML scale is too low, it can happen on some browsers that text will overflow on the right side of objects, especially on Firefox. In this case you can use this property to fix the problem.

void VpeSetHtmlWordSpacing(
VpeHandle *hDoc*,
double *word_spacing*
)

VpeHandle hDoc
Document Handle

double word_spacing
the word spacing of the generated HTML document, a spacing below 0 moves words closer together, a spacing above 0 expands the distance between words

Default:
0

28.5 VpeGetHtmlWordSpacing

Returns the current word spacing for HTML output.

double VpeGetHtmlWordSpacing(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current word spacing for HTML output

28.6 VpeSetHtmlRtfLineSpacing

When viewing exported documents which contain Rich Text in Mozilla browsers, the default line spacing is too large and text will overflow objects at the bottom. In this case set this property to true. This option improves appearance of RTF for Mozilla browsers, but worsens the result in Internet Explorer.

void VpeSetHtmlRtfLineSpacing(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	shrink the line spacing for viewing with Mozilla browsers
False	do not change the line spacing

Default:
False

28.7 VpeGetHtmlRtfLineSpacing

Returns the current RTF line spacing setting.

int VpeGetHtmlRtfLineSpacing(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current RTF line spacing setting

Value	Description
True	shrink the line spacing for viewing with Mozilla browsers
False	do not change the line spacing

28.8 VpeSetHtmlCopyImages

Specifies the mode for handling images of generated HTML documents.

If set to true, VPE copies / creates images in a destination directory named "<html file>.img". If an image is used multiple times, it is only stored once in the target directory. If the original images are of type PNG, JPG or GIF and the Rotation is zero degrees, the original image is copied. Otherwise a compressed PNG is created from the original.

If set to false, HTML links to the original locations of **all** images are written. In this case you may only use file types, which are supported by HTML browsers, i.e. PNG, JPG or GIF. Furthermore rotation is not applied. If you require rotated images, they should be stored on disk already in rotated form.

Windows platform: regardless of this property's value, VPE objects not supported by HTML are still exported as PNG files. If you don’t want this, do not use such objects.

void VpeSetHtmlCopyImages(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	copy / create images
False	write HTML links to the original image locations

Default:
True

Remarks:
This property should be set to false, if exporting HTML to a memory stream.

28.9 VpeGetHtmlCopyImages

Returns the current value of the image-copy mode.

int VpeGetHtmlCopyImages(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current value of the image-copy mode

Value	Description
True	copy / create images
False	write HTML links to the original image locations

28.10 VpeSetHtmlPageBorders

If true, borders are added at page boundaries, i.e. a frame is drawn at the page boundaries.

void VpeSetHtmlPageBorders(
VpeHandle *hDoc*,
int *yes_no*
)

VpeHandle hDoc
Document Handle

int yes_no

Value	Description
True	add borders at page boundaries
False	do not draw borders at page boundaries

Default:
False

28.11 VpeGetHtmlPageBorders

Returns the current value of the page borders property.

int VpeGetHtmlPageBorders(
    VpeHandle *hDoc*
)

VpeHandle hDoc
    Document Handle

Returns:
    the current value of the image-copy mode

Value	Description
True	add borders at page boundaries
False	do not draw borders at page boundaries

28.12 VpeSetHtmlPageSeparators

If true, pages are separated by horizontal rules.

void VpeSetHtmlPageSeparators(
  VpeHandle *hDoc*,
  int *yes_no*
)

VpeHandle hDoc
  Document Handle

int yes_no

Value	Description
True	separate pages by horizontal rules
False	do not separate pages by horizontal rules

Default:
  False

28.13 VpeGetHtmlPageSeparators

Returns the current value of the page separators property.

int VpeGetHtmlPageSeparators(
VpeHandle *hDoc*
)

VpeHandle hDoc
Document Handle

Returns:
the current value of the image-copy mode

Value	Description
True	separate pages by horizontal rules
False	do not separate pages by horizontal rules

29 Addendum

29.1 Names of the Registered Window Messages

The following is the complete list of Window messages registered by VPE. In order to avoid in any case conflicts with messages of other applications, all names start with the text ")/$%!" .

- ")/$%!VpeDestroyWindow"
- ")/$%!VpePrint"
- ")/$%!VpeAutoPageBreak"
- ")/$%!VpePrintNewPage"
- ")/$%!VpePrintDevData"
- ")/$%!VpeHelp"
- ")/$%!VpeCloseWindow"
- ")/$%!VpeObjectClicked"
- ")/$%!VpeUDOPaint"
- ")/$%!VpeBeforeMail"
- ")/$%!VpeAfterMail"
- ")/$%!VpeCanClose"
- ")/$%!VpeBeforeOpenFile"
- ")/$%!VpeAfterOpenFile"
- ")/$%!VpeBeforeSaveFile"
- ")/$%!VpeAfterSaveFile"

Additional messages of the VPE Interactive Edition:

- ")/$%!VpeCtrlAfterEnter"
- ")/$%!VpeCtrlCanExit"
- ")/$%!VpeCtrlAfterExit"
- ")/$%!VpeCtrlAfterChange"
- ")/$%!VpeFieldAfterChange"

29.2 How to handle VPE Events using the MFC

The events fired by VPE can be handled in two ways:

- either use VPE_FIXED_MESSAGES when calling VpeOpenDoc()
- or do the following:

Define the message map in your parent window of VPE as:

```
BEGIN_MESSAGE_MAP(CMyWnd)
     ON_REGISTERED_MESSAGE(uVpeDestroyWindow, OnVpeDestroyWindow)
     ON_REGISTERED_MESSAGE(uVpeHelp, OnVpeHelp)
     ON_REGISTERED_MESSAGE(uVpePrint, OnVpePrint)
     ON_REGISTERED_MESSAGE(uVpePrintNewPage, OnVpePrintNewPage)
     ON_REGISTERED_MESSAGE(uVpeCloseWindow, OnVpeCloseWindow)
     ON_REGISTERED_MESSAGE(uVpeObjectClicked, OnVpeObjectClicked)
     ON_REGISTERED_MESSAGE(uVpeUDOPaint, OnVpeUDOPaint)
     ON_REGISTERED_MESSAGE(uVpeAutoPageBreak, OnVpeAutoPageBreak)
     ON_REGISTERED_MESSAGE(uVpeBeforeMail, OnVpeBeforeMail)
     ON_REGISTERED_MESSAGE(uVpeAfterMail, OnVpeAfterMail)
     ON_REGISTERED_MESSAGE(uVpePrintDevData, OnVpePrintDevData)
END_MESSAGE_MAP()
```

and the constants itself as public in the CMyWnd.h class:

```
// Registered window messages
public:
     static const UINT uVpeDestroyWindow;
     static const UINT uVpePrint;
     static const UINT uVpeAutoPageBreak;
     static const UINT uVpePrintNewPage;
     static const UINT uVpeHelp;
     static const UINT uVpeCloseWindow;
     static const UINT uVpeObjectClicked;
     static const UINT uVpeUDOPaint;
     static const UINT uVpeBeforeMail;
     static const UINT uVpeAfterMail;
     static const UINT uVpePrintDevData;
```

In CMyWnd.cpp assign the according values to the constants in the global initialization block of the module, i.e. not within a method:

```
const UNT CMyWnd::uVpeDestroyWindow =
          RegisterWindowMessage (_T(")/$%!VpeDestroyWindow"));

const UINT CMyWnd::uVpePrint = RegisterWindowMessage (_T(")/$%!VpePrint"));

const UINT CMyWnd::uVpeAutoPageBreak =
          RegisterWindowMessage (_T(")/$%!VpeAutoPageBreak"));

const UINT CMyWnd::uVpePrintNewPage =
          RegisterWindowMessage (_T(")/$%!VpePrintNewPage"));

const UINT CMyWnd::uVpeHelp = RegisterWindowMessage (_T(")/$%!VpeHelp"));

const UINT CMyWnd::uVpeCloseWindow =
          RegisterWindowMessage (_T(")/$%!VpeCloseWindow"));

const UINT CMyWnd::uVpeObjectClicked =
          RegisterWindowMessage (_T(")/$%!VpeObjectClicked"));

const UINT CMyWnd::uVpeUDOPaint =
          RegisterWindowMessage (_T(")/$%!VpeUDOPaint"));

const UINT CMyWnd::uVpeBeforeMail =
          RegisterWindowMessage (_T(")/$%!VpeBeforeMail"));

const UINT CMyWnd::uVpeAfterMail =
          RegisterWindowMessage (_T(")/$%!VpeAfterMail"));

const UINT CMyWnd::uVpePrintDevData =
          RegisterWindowMessage (")/$%!VpePrintDevData");
```

30 Index