VPE Control Reference v7.40

USER MANUAL

This page is intentionally left blank.

This page is intentionally left blank.

How To Use the VPE Control

1 How To Use the VPE Control

The VPE Control is a wrapper around the VPE DLL (shared object / dylib).

It is available for the following programming languages:

- .NET
- Java
- PHP
- Python
- Ruby
- Delphi / C++ Builder VCL
- and as ActiveX

Because the events, properties and methods of the VPE Control are for all programming languages nearly 100% the same, we explain the VPE Control for all languages in this book.

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

A first introduction on using VPE is given in the "Programmer's Manual" chapter 2 "Getting Started".

The techniques on using VPE are explained in the "Programmer's Manual" chapter 4 "Programming Techniques".

The installation procedure for each control is also described in detail in the "Programmer's Manual" chapter 1 "Installation".

Please read the following sections careful, which describe the basic techniques on using the VPE Control with each supported programming language. Then turn over to the "Programmer's Manual" and read the whole chapter 4 "Programming Techniques" very carefully. This document is a reference for the VPE-Control, not for Virtual Print Engine itself.

1.1 VPE .NET Control

Installing the VPE .NET Control in Visual Studio

Open the path \vpe-install-dir\deploy in Windows Explorer.

```
Example: c:\program files\VPE_P64.740\deploy
```

Open the Toolbox in Visual Studio.

Drag and Drop from the Explorer window the files IDEALSoftware.VpeInteractive.dll and IDEALSoftware.VpeWebInteractive.dll into the Visual Studio Toolbox.

Note: Visual Studio versions prior 2022 are x86 applications, so you need to have VPE 32-bit installed in order to use VPE in Visual Studio for development.

Visual Studio 2022 is an x64 application, so you need to have VPE 64-bit installed in order to use VPE in Visual Studio 2022 and higher for development.

1.1.1 General Usage

The **Winforms .NET Control**, is based on the Winforms classes and should be used for GUI applications. It provides a preview window and related methods / properties.

VPE also includes a **.NET WebServerControl** (VpeWebControl) for use with ASP.NET. The control is a subset of the Winforms .NET Control. It does not have any events, properties or methods which are related to the graphical user interface (GUI) of VPE, since it is intended solely to be executed on servers within ASP.NET and therefore does not provide a GUI. All events, properties and methods not supported by the VpeWebControl are marked throughout this manual.

The common sequence of function calls is:

- Set the properties for behavior and appearance of VPE in the design mode or during runtime before calling the method OpenDoc()
- Open a document with the method "OpenDoc 189"
- Use all possible methods to insert VPE objects (like Write(), etc.)
- Use "PageBreak 363" to generate new pages
- Use "Preview 206" to show the preview to the user, this is optional
- Use "PrintDoc 311" to print the document, or "WriteDoc()" 225 to export it
- Close the document with "CloseDoc 193"

It is only possible to open one document per VPE-Control. If you want to open multiple documents simultaneously, you need to place the same number of VPE-Controls on the form.

Example for Visual Basic .NET, which can be easily translated to other programming languages:

The following example assumes that you use a VPE .NET Control object named "Report".

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
                       System.EventArgs) Handles MyBase.Load
     Report.OpenDoc()
     Report.Print(1, 1, "Hello World!")
     Report.Preview()
End Sub
```

This example is very simple. The document is opened, the text "Hello World!" is inserted at position (1, 1) and afterwards the preview is shown.

For a detailed explanation of the basic programming techniques, please continue reading in the "Programmer's Manual" chapter 4 "Programming Techniques".

1.1.2 Class Names and Namespaces

As mentioned earlier, the VPE.NET Control is included in two different versions:

The **Winforms .NET Control** and the (WebForms) **.NET WebServerControl**, which is a subset of the Winforms .NET Control. The .NET WebServerControl does not have any events, properties or methods which are related to the graphical user interface of VPE, since it is intended solely to be executed on servers within ASP.NET and therefore does not provide a GUI. All events, properties and methods not supported by the VPE .NET WebServerControl are marked throughout this manual.

The Winforms .NET VPE Control has the class name *VpeControl*.
It is derived from *System.Windows.Forms.UserControl*.

The VPE .NET WebServerControl has the class name *VpeWebControl*.
It is derived from *System.Web.UI.Control*.

Depending on the edition of VPE you are using, the namespace of the control is:

Edition	Winforms Namespace	WebServerControl Namespace
Community	IDEALSoftware.VpeCommunity	IDEALSoftware.VpeWebCommunity
Standard	IDEALSoftware.VpeStandard	IDEALSoftware.VpeWebStandard
Enhanced	IDEALSoftware.VpeEnhanced	IDEALSoftware.VpeWebEnhanced
Professional	IDEALSoftware.VpeProfessional	IDEALSoftware.VpeWebProfessional
Enterprise	IDEALSoftware.VpeEnterprise	IDEALSoftware.VpeWebEnterprise
Interactive	IDEALSoftware.VpeInteractive	IDEALSoftware.VpeWebInteractive

You will notice that the class name of the VPE Control within the Winforms category is the same for each edition, i.e. it is *VpeControl* for the Standard Edition, as well as for the

Enhanced Edition, the Professional Edition, etc. The same applies to the WebServerControl category, where the class name is *VpeWebControl* for all editions.

This makes it simple for you to migrate your source code from one edition of VPE to another, if you should decide later to upgrade to a higher edition. And since the VPE .NET WebServerControl constitutes a subset of the Winforms VPE Control, the migration of your source code from one platform to the other can be done within seconds.

NOTE: When using the VpeWebControl, always set the property EnableMultiThreading 199 = true.

The call-sequence must be:

```
OpenDoc()
EnableMultiThreading = true
```

1.1.3 Exceptions

In case of error conditions, the .NET Control might throw exceptions. Properties and methods which throw exceptions are marked in this reference. VPE throws only one kind of exception: InvalidOperationException(). The informational message of the exception object explains in details the cause for each exception.

1.1.4 Understanding the Data Types and Declarations

Because the properties and methods of the VPE Control (for .NET, Java, PHP, Python, Ruby, ActiveX and Delphi / C++ Builder VCL) are nearly 100% the same, we explain the API of the VPE control for all the different programming languages within this single book. We use a meta-description to describe the data types, properties, methods and parameters of the VPE API.

1.1.5 Visual Basic .NET

Throughout this manual, methods and parameters are always declared in the form <Data Type> <Name>, e.g. *boolean AutoDelet*e would be declared in VB as *Dim AutoDelete as Boolean*.

Data Types:

VpeCoord is Double

boolean is *Boolean*

integer is *Integer*

long is Integer

string is *String*

Color is *Color*

In addition there are defined many enumerations as data types for properties as well as for parameters of methods:

e.g. *property PenStyle [integer] PenStyle*
declares a property which has the enumeration type "PenStyle" and which is named "PenStyle".
Data types in square-brackets, like "[integer]" in the example above, are not of interest for .NET users. They describe the data type for the ActiveX and VCL controls.

Example on using the above property: *Doc.PenStyle = PenStyle.Solid*

i.e. the property named *PenStyle* is assigned the value *Solid* of a pre-defined enumeration, which has the type *PenStyle.*

Methods:

The keyword *method* declares a method, e.g. *method void CloseProgressBar()*.

A method is either a Sub or a Function, depending on how the return type of the method is declared. The keyword immediately following the word *method* declares the method's return type.

void is a *Sub* which does not return a value, e.g.
method void StorePos() is in Visual Basic: *Sub StorePos()*

all other declarations declare a function, e.g.
method boolean WriteDoc(string FileName) is in Visual Basic .NET
Function WriteDoc(FileName as String) as Boolean

Properties:

The keyword *property* describes a property. The line below the declaration of the property explains, how the property can be accessed (read-only, write-only or read / write together) and if it can be accessed during runtime only, or also during design time.

Example:

property boolean PageScrollerTracking

read / write; design- & runtime

means: the property is of type *Boolean*, its name is *PageScrollerTracking* and it can be accessed for *read* (you can read its value, e.g. *n = PageScrollerTracking*) and for *write* (you can assign it a value, e.g. *PageScrollerTracking = True*) and it is available during runtime (while your application is running) and design time (when you are developing your forms).

It would be declared in Visual Basic .NET as *Dim PageScrollerTracking as Boolean*

1.1.6 C#

Throughout this manual, methods and parameters are always declared in the form as it is done in C#. Only some data types are named differently.

Data Types:

VpeCoord is double

boolean is *bool*

integer is *int*

long is int

string is s*tring*

Color is *Color*

In addition there are defined many enumerations as data types for properties as well as for parameters of methods:

e.g. *property PenStyle [integer] PenStyle*
declares a property which has the enumeration type "PenStyle" and which is named "PenStyle".
Data types in square-brackets, like "[integer]" in the example above, are not of interest for .NET users. They describe the data type for the ActiveX and VCL controls.

Example on using the above property: *Doc.PenStyle = PenStyle.Solid*

i.e. the property named *PenStyle* is assigned the value *Solid* of a pre-defined enumeration, which has the type *PenStyle.*

Methods:

The keyword *method* declares a method, e.g. *method void CloseProgressBar()*.

Or *method boolean WriteDoc(string FileName)*

Properties:

The keyword *property* describes a property. The line below the declaration of the property explains, how the property can be accessed (read-only, write-only or read / write together) and if it can be accessed during runtime only, or also during design time.

Example:

property boolean PageScrollerTracking

read / write; design- & runtime

means: the property is of type *bool*, its name is *PageScrollerTracking* and it can be accessed for *read* (you can read its value, e.g. *n = PageScrollerTracking*) and for *write* (you can assign it a value, e.g. *PageScrollerTracking = True*) and it is available during runtime (while your application is running) and design time (when you are developing your forms).

It would be declared in C# as *public bool PageScrollerTracking*

1.2 VPE Java Control

The VPE Java Control is shipped in two different versions: the GUI and the Non-GUI version.

The GUI version is only available for Windows and offers an optional preview window.

The Non-GUI version has some special features on the Windows platform:

- Printing documents directly to printers
- Exporting pages as image files
- Mailing documents

The version on non-Windows platforms lacks the above features, but is in all other aspects 100% identical to the Windows-Version.

The GUI version can be found in the file
<vpe-installation-directory> / deploy / vpegui.jar

The Non-GUI version is available for all platforms and can be found in the file
<vpe-installation-directory> / deploy / vpe.jar

Make sure that any of the two VPE JAR files is in your classpath.

For the Non-GUI version the recommended way of using VPE is to export a created document to PDF and to use a PDF Reader (like Adobe Acrobat) for displaying and/or printing the document, or to send the PDF document via HTTP to a client browser.

For the use with server applications on the Windows platform, the Non-GUI version is recommended.

The common sequence of function calls is:

- Set the properties for behavior and appearance of VPE during runtime before calling the method openDoc()
- Open a document with the method "OpenDoc 189"
- Use all possible methods to insert VPE objects (like Write(), etc.)
- Use "PageBreak 363" to generate new pages
- Use "Preview 206" to show the preview to the user, this is optional and only supported in the GUI version for the Windows platform
- Use "PrintDoc 311" to print the document (only supported in the GUI version for the Windows platform), or "WriteDoc()" 225 to export it
- Close the document with "CloseDoc 193"

Example:

Create a file named “VpeTest.java” with the following code:

```
import com.idealSoftware.vpe.*;
import com.idealSoftware.vpe.events.*;

class VpeTest {

  public static void main(String[] args) {
       VpeControl doc = new VpeControl();
       doc.openDoc();
       doc.print(1, 1, "Hello World!");
       doc.writeDoc("hello world.pdf");
       doc.closeDoc();
  }
}
```

Compile with: `javac -classpath ..\deploy\vpe.jar VpeTest.java`

Execute with: `java -cp .;../deploy/vpe.jar VpeTest`

Make sure to adjust the classpath, so that it points to the VPE JAR file in your environment.

For the Windows platform, here is a version which shows a preview window:

Please note that the major difference to the previous version is one additional line of code with a call to "Doc.preview()" (and a method pause()).

Create a file named "VpeGuiTest.java" with the following code:

```
import com.idealSoftware.vpe.*;
import com.idealSoftware.vpe.events.*;
import java.io.*;

class VpeGuiTest {

  static void pause()
  {
       try
       {
            System.out.printf("\n\n  Press ENTER...");
            System.in.read();
            while (System.in.available() > 0)
                System.in.read();    // flush the buffer
       }
       catch (IOException e)
       {
            System.out.printf("Error\n");
       }
  }

  public static void main(String[] args) {
       VpeControl doc = new VpeControl();
       doc.openDoc();
       doc.print(1, 1, "Hello World!");
       doc.preview();
       doc.writeDoc("hello world.pdf");
       pause();
       doc.closeDoc();
  }
}
```

Compile with: `javac -classpath ..\deploy\vpegui.jar VpeGuiTest.java`

Execute with: `java -cp .;../deploy/vpegui.jar VpeGuiTest`

Make sure to adjust the classpath, so that it points to the VPE JAR file in your environment.

This example is very simple. A document is created, the text "Hello World!" is inserted at position (1, 1) and afterwards the document is exported to the PDF file named “hello word.pdf”.

For a detailed explanation of the basic programming techniques, please continue reading in the "Programmer's Manual" chapter 4 "Programming Techniques".

1.2.1 Exceptions

In case of error conditions, the Java Control might throw exceptions. Properties and methods which throw exceptions are marked in this reference. VPE throws only one kind of exception: *InvalidOperationException*. The informational message of the exception object explains in details the cause for each exception.

1.2.2 Understanding the Data Types and Declarations

Because the properties and methods of the VPE Control (for .NET, Java, PHP, Python, Ruby, ActiveX and Delphi / C++ Builder VCL) are nearly 100% the same, we explain the API of the VPE control for all the different programming languages within this single book. We use a meta-description to describe the data types, properties, methods and parameters of the VPE API.

Here is how the meta-description is converted to Java:

Throughout this manual, methods and parameters are always declared in the form as it is done in Java. Only some data types are named differently.

Data Types:

VpeCoord is double

boolean is *boolean*

integer is *int*

long is long

string is *String*

Color is *java.awt.Color*

In addition there are defined many enumerations as data types for properties as well as for parameters of methods:

e.g. *property PenStyle [integer] PenStyle*
declares a property which has the enumeration type "PenStyle" and which is named "PenStyle".
Data types in square-brackets, like "[integer]" in the example above, are not of interest for Java users. They describe the data type for the ActiveX and VCL controls.

Example on using the above property: *Doc.PenStyle = PenStyle.Solid*

i.e. the property named *PenStyle* is assigned the value *Solid* of a pre-defined enumeration, which has the type *PenStyle.*

Methods:

The keyword *method* declares a method, e.g. *method void CloseProgressBar()*.

Or *method boolean WriteDoc(string FileName)*

Please note that for methods, according to the Java naming conventions, the first letter is always lower-case. So for Java, *void CloseProgressBar()* translates into *void closeProgressBar().*

Properties:

The keyword *property* describes a property. In many programming languages a property is accessed like a class member variable, e.g. you can write `Vpe.AutoDelete = true`. But when this code is executed, in fact a setter-method is called behind the scenes, i.e. `Vpe.setAutoDelete(true)`. The same applies to reading the value of a property, behind the scenes a getter-method is called, e.g. for code like `status = Vpe.AutoDelete` the following getter is called: `status = Vpe.getAutoDelete()`.

Unfortunately, Java does not provide this elegant coding technique. Instead of properties, it does only provide getter- and setter-methods.

So when you encounter the keyword *property* in this manual, you need to translate it into getter- and setter-methods. The line below the declaration of the property specifies, how the property can be accessed. Each property is marked for read only (only a getter is present), write only (only a setter is present) or read / write (a getter and a setter is present).

It is also indicated, if the property can be accessed during runtime only, or also during design time. “Design Time” means: when the document is closed and “runtime” means: when the document is open, i.e. the method Vpe.openDoc() has been called. The term “Design Time” relates to programming tools with visual designers, where properties of graphical components can be modified by point-and-click during the design time of an application, for example with a Property Grid.

Example:

property boolean PageScrollerTracking

read / write; design- & runtime

means: the property is of type *boolean*, its name is *PageScrollerTracking* and it can be accessed for *read* (a getter is present) and for *write* (a setter is present) and it is available during runtime (when the document is open) and design time (when the document is closed).

For Java, this translates into the following getter and setter methods:

boolean getPageScrollerTracking ()

void setPageScrollerTracking (boolean value)

If the first letter of a property is lower-case, it must be converted to upper-case for getters and setters.

property VpeCoord nLeft

Translates into the following getter and setter methods:

double getNLeft ()

void setNLeft (double value)

The following members of the class *VpeControl* can be used as coordinates, if they are used read-only:

```
public static final double nFree = -2147483549;
public static final double nLeft = -2147483550;
public static final double nRight = -2147483551;
public static final double nLeftMargin = -2147483552;
public static final double nRightMargin = -2147483553;
public static final double nTop = -2147483554;
public static final double nBottom = -2147483555;
public static final double nTopMargin = -2147483556;
public static final double nBottomMargin = -2147483557;
```

Example:

```
doc.picture(1, 1, doc.nFree, doc.nFree, “sample.png”);
```

1.3 VPE Python

The VPE Control for Python is for version 3.x of Python only.

The class name is *VpeControl*.

The common sequence of function calls is:

- Set the properties for behavior and appearance of VPE before calling the method OpenDoc
- Open a document with the method "OpenDoc 189"
- Use all possible methods to insert VPE objects (like Write(), etc.)
- Use "PageBreak 363" to generate new pages
- Use "Preview 206" to show the preview to the user, this is optional
- Use "PrintDoc 311" to print the document, or "WriteDoc() 225" to export it
- Close the document with "CloseDoc 193"

It is only possible to open one document per VPE control. If you want to open multiple documents simultaneously, you need use the same number of VPE controls.

Create a file named “VpeTest.py” with the following code:

```
import os, sys
from VpeControl import *

doc = VpeControl()
doc.OpenDoc()
doc.Print(1, 1, "Hello World!")
doc.WriteDoc("hello world.pdf")
doc.CloseDoc()
```

Execute with: `python VpeTest.py`

This example is very simple. The document is opened, the text "Hello World!" is inserted at position (1, 1) and afterwards the document is written as PDF file.

For the Windows platform, here is a version which shows a preview window:

Create a file named “VpeGuiTest.py” with the following code:

```
import os, sys
from time import sleep
from VpeControl import *

doc = VpeControl()
doc.OpenDoc()
doc.Print(1, 1, "Hello World!")
doc.WriteDoc("hello world.pdf")

# Note that the DispatchAllMessages()-loop is required to pump
messages from the
# Windows operating system to the preview. This keeps the preview
alive.
doc.Preview()
abort = False
while not abort:
   abort = doc.DispatchAllMessages()
   sleep(0.01)
```

Because Python is windowless, you need to pump Windows messages to the preview as shown above.

For a detailed explanation of the basic programming techniques, please continue reading in the "Programmer's Manual" chapter 4 "Programming Techniques".

1.3.1 Multiple Previews

Because Python is windowless, you need to pump Windows messages to the preview:

```
# Note that the DispatchAllMessages()-loop is required to pump
messages from the
# Windows operating system to the preview. This keeps the preview
alive.
doc.Preview()
abort = False
while not abort:
   abort = doc.DispatchAllMessages()
   sleep(0.01)
```

If you want to show multiple previews at the same time, each document must be created within its own thread, ending in the above message pump. You can not put only the message pump into a thread, because the Windows operating system only dispatches messages for windows that have been created within the same thread in which the message pump is running. So as a result, the whole VPE document must be created within the same thread, including the message pump.

1.3.2 Exceptions

In case of error conditions, the Python VPE Control might throw exceptions. Properties and methods which throw exceptions are marked in this reference. The informational message of the exception object explains in details the cause for each exception.

1.3.3 Text Output

Internally, VPE processes ANSI strings, not Unicode. You provide UTF-8 encoded strings to VPE, but you must specify the charset that VPE shall use for string conversion, before using a non-western charset. See the property CharSet 481 in this help file.

Example:

```
import os, sys
from VpeControl import *

doc = VpeControl()
doc.OpenDoc()
doc.Print(1, 1, "Hello World!")
doc.CharSet = VCHARSET_WIN_CYRILLIC     # switch to Cyrillic charset
doc.Print(1, 2, 'Международную')
doc.WriteDoc("hello world.pdf")
```

1.4 VPE ActiveX

The common sequence of function calls is:

- Set the properties for behavior and appearance of VPE in the design mode or during runtime before calling the method OpenDoc
- Open a document with the method "OpenDoc 189"
- Use all possible methods to insert VPE objects (like Write(), etc.)
- Use "PageBreak 363" to generate new pages
- Use "Preview 206" to show the preview to the user, this is optional
- Use "PrintDoc 311" to print the document, or "WriteDoc() 225" to export it
- Close the document with "CloseDoc 193"

It is only possible to open one document per VPE-ActiveX. If you want to open multiple documents simultaneously, you need to place the same number of VPE-ActiveX's on the form.

Example for Visual Basic, which can be easily translated to other programming languages:

The following example assumes that you use a VPE-ActiveX object named "Report".

```
Private Sub Form_Load()
     Report.OpenDoc
     Report.VpePrint 1, 1, "Hello World!"
     Report.Preview
End Sub
```

This example is very simple. The document is opened, the text "Hello World!" is inserted at position (1, 1) and afterwards the preview is shown.

Informational: the VPE ActiveX is using the Apartment Threading Model.

For a detailed explanation of the basic programming techniques, please continue reading in the "Programmer's Manual" chapter 4 "Programming Techniques".

1.4.1 Where are the constants like 'VORIENT_PORTRAIT' defined?

The special ActiveX / VCL constants are automatically defined, when using the ActiveX or VCL. Both export these constants to your source code.

But some Containers (like for example Visual FoxPro) do not import the constants (like VORIENT_PORTRAIT, VFREE, ALIGN_LEFT, etc.).

For a few selected Containers we included such definition files for import. Please check the source code directories.

1.4.2 Important Note for Users of Visual Basic

Visual Basic has problems with the keywords "Print, Write, Line and Scale". VB doesn't recognize that these are methods and properties of an ActiveX. So we implemented them twice. In VB use "VpePrint, VpeWrite, VpeLine and VpeScale" instead.

1.4.3 Understanding the Data Types and Declarations (Visual Basic)

Because the properties and methods of the VPE Control (for .NET, Java, PHP, Python, Ruby, ActiveX and Delphi / C++ Builder VCL) are nearly 100% the same, we explain the API of the VPE control for all the different programming languages within this single book. We use a meta-description to describe the data types, properties, methods and parameters of the VPE API.

Here is how the meta-description is converted to Visual Basic:

Throughout this manual, methods and parameters are always declared in the form <Data Type> <Name>, e.g. *boolean AutoDelet*e would be declared in VB as *Dim AutoDelete as Boolean*.

The data type in square-brackets defines the data type for the ActiveX.

Example:

property PenStyle [integer] PenStyle

means that this property is of type "integer".
The leading "PenStyle" type declaration is for the .NET control only.

Data Types:

VpeCoord is Double

boolean is *Boolean*

integer is *Integer*

long is *Long*

string is *String*

Color is *Color* (in fact a 32-bit long integer)

Properties / Methods:

The keyword *method* declares a method, e.g. *method void CloseProgressBar()*.

A method is either a Sub or a Function, depending on how the return type of the method is declared. The keyword immediately following the word *method* declares the method.

void is a *Sub* which does not return a value, e.g.
method void StorePos() is in Visual Basic: *Sub StorePos()*

all other declarations declare a function, e.g.
method boolean WriteDoc(string FileName) is in Visual Basic
Function WriteDoc(FileName as String) as Boolean

The keyword *property* describes a property. The line below the declaration of the property explains, how the property can be accessed (read-only, write-only or read / write together) and if it can be accessed during runtime only, or also during design time.

Example:

property boolean PageScrollerTracking

read / write; design- & runtime

means: the property is of type *Boolean*, its name is *PageScrollerTracking* and it can be accessed for *read* (you can read its value, e.g. *n = PageScrollerTracking*) and for *write* (you can assign it a value, e.g. *PageScrollerTracking = True*) and it is available during runtime (while your application is running) and design time (when you are developing your forms).

It would be declared in Visual Basic as *Dim PageScrollerTracking as Boolean.*

1.4.4 Exceptions

In case of error conditions, the VPE ActiveX might raise errors. We call that "*throw exceptions*".

Properties and methods of the VPE ActiveX which raise errors are marked in this reference.

Visual Basic for example offers three statements to handle these errors:

- On Error GoTo *line*
- On Error Resume Next
- On Error GoTo 0

Visual Basic: please note that *Err.Number* needs to be compared with the VPE Exception Code + the constant value *vbOBJECTERROR*.

```
Select Case Err.Number
     case VPE_E_APIFAILED + vbOBJECTERROR    'Error caused by VPE API
          If VPE.LastError = VERR_TPL_AUTHENTICATION Then
               MsgBox("'sample2.dcd' has been modified.", "Error:",
MB_OK);
          ElseIf VPE.LastError = VERR_FILE_OPEN Then
               MsgBox("'sample2.dcd' not found or no access rights.",
                    "Error:", MB_OK);
          Else
               MsgBox("'sample2.dcd' could not be read.", "Error:",
MB_OK);
          End If
     case Else
          'Handle other situations here...
     End Select
End Sub
```

Please consult the documentation of your programming language / tool for further details on how to handle errors.

Possible VPE Exception Codes raised by the ActiveX are:

- VPE_E_NEEDOPENDOCUMENT 1024
  The VPE Document is not open. You need to call OpenDoc prior to this operation.
- VPE_E_ NEEDCLOSEDDOCUMENT 1025
  The VPE Document is open. The document needs to be closed prior to this operation.
- VPE_E_ ENGINEAPI 1026
  An underlaying API routine was not found in the VPE dynamic link library.
- VPE_E_ OUTOFRANGE 1027
  The argument is out of range.
- VPE_E_ NOTIMPL_EDITION 1028
  Sorry, this feature is not implemented in this edition of VPE.
- VPE_E_ NOWINDOW 1029
  Need a window to perform this operation. The object must have been in-place activated first.
- VPE_E_ APIFAILED 1030
  An underlaying VPE API routine has failed. Possible causes: Invalid arguments, conversion of data type impossible, Out of Memory, etc.
- VPE_E_ INVALIDOBJECTHANDLE 1031
  The handle to the object is invalid: it does not point to a valid VPE DLL object. Probably the method/property that returned the object has failed.
- VPE_E_ VPEGETFAILED 1032
  The value could not be retrieved. Possible causes: Field/Control not found, Data Type Conversion Impossible or Out of Memory.
- VPE_E_ VPESETFAILED 1033
  The value could not be set. Possible causes: Field/Control not found, Data Type Conversion Impossible or Out of Memory.

1.4.5 CreateObject Call

The ActiveX supports the CreateObject() call, i.e. it can be called from non-visual classes and objects. This is very useful if running in Active Server Pages (ASP).

Example for Visual Basic:

```
Dim x As Object
Set x = CreateObject("VpeControl.VpeControl.74")
```

The important parameter is "VpeControl.VpeControl.74" which is the class name of the VPE ActiveX Control.

1.4.6 Early Binding

The VPE ActiveX can be used with early binding without using a form, which has some performance advantages, for example for batch-processes on a server.

Example for Visual Basic:

IMPORTANT: do not insert VPE into your VB Project as Component, but as Reference!

```
Dim report As VPE
Private Sub Command1_Click()
     Set report = New VPE
     report.OpenDoc
     report.Preview
End Sub
```

1.4.7 Visual FoxPro Note

To make Visual FoxPro process all VPE events, and to prevent getting sporadic "OLE Exception Error" messages, you need to set in VFP the system variable "**_VFP.AutoYield = .F.**"

If you want that a form which contains the VPE ActiveX is closed when the Close Button in VPE's toolbar is clicked, do not call *thisform.release* when processing the event AfterDestroyWindow() 137, because VFOX will GPF for an unknown reason. Workaround: hide the Close Button in VPE's toolbar by setting tbClose 253 = .F. - since the VFP form already has a Close Button.

Otherwise, there is the following workaround: when receiving the AfterDestroyWindow() 137 event fired by VPE, start a timer with 0,1 second delay. After the timer has been counted down, you may call *thisform.release* safely.

1.4.8 MFC: Using the VPE ActiveX with the MFC without placing it in a Dialog Resource

If you didn’t place the VPE ActiveX as Resource into a Dialog, you need to implement the COM licensing mechanism of Microsoft by yourself (see the according documentation from Microsoft).

In short:

Call CWnd::CreateControl() - in the last parameter you must specify the license key **generated** by the control. In order to get the **generated** key, you once need to create during development a project with the VPE ActiveX and call the methods:

```
lpU = CWnd::GetControlUnknown()
lpU -> QueryInterface(IID_IClassFactory2, &pClassFactory2)
pClassFactory2->RequestLicKey(NULL, pBstrKey)
```

pBstrKey will contain the generated key, which is the coded License Key.
Use this key in all other calls to CWnd::CreateControl() in your projects.

If you place the VPE ActiveX in VC++ as resource in a dialog you will have no problems and you don't need to get the key, etc. because VC does everything for you.#

1.4.9 Using the VPE ActiveX in a CFormView derived class

Q: If I use the VPE ActiveX in a CFormView derived class, the control is not updated after a call to CFormView::Invalidate() or CDocument::UpdateAllViews().

A: Set the style of the dialog resource containing the VPE ActiveX to "Clip Children" (do not change the style in PreCreateWindow()). You can also call CVpeControl::UpdateWindow() in the overridden CFormView::OnUpdate() method.

1.5 VPE VCL

The common sequence of function calls is:

- Set the properties for behavior and appearance of VPE in the design mode, or during runtime before calling OpenDoc
- Open a virtual document with the method "OpenDoc 189"
- Use all possible methods to insert VPE objects (like Write(), etc.)
- Use "PageBreak 363" to generate new pages
- Use "Preview 206" to show the preview to the user, this is optional
- Use "PrintDoc 311" to print the document, or "WriteDoc()" 225 to export it
- Close the document with "CloseDoc 193"

It is only possible to open one document per VPE-VCL. If you want to open multiple documents simultaneously, you need to place the same number of VPE-VCL's on the form.

The following example assumes that you use a VPE-VCL object named "Report".

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Report.OpenDoc;
    Report.Print(1, 1, 'Hello World!');
    Report.Preview;
end;
```

This example is very simple. The document is opened, the text "Hello World!" is inserted at position (1, 1) and afterwards the preview is shown.

For a detailed explanation of the basic programming techniques, please continue reading in the "Programmer's Manual" chapter 4 "Programming Techniques".

If you are using the VPE Enterprise or Interactive Edition, there is also a very important note in the Programmer's Manual: Important Note for VPE-VCL Users!

1.5.1 Understanding the Data Types and Declarations (Delphi)

Because the properties and methods of the VPE Control (for .NET, Java, PHP, Python, Ruby, ActiveX and Delphi / C++ Builder VCL) are nearly 100% the same, we explain the API of the VPE control for all the different programming languages within this single book. We use a meta-description to describe the data types, properties, methods and parameters of the VPE API.

Here is how the meta-description is converted to Delphi:

Throughout this manual, methods and parameters are always declared in the form <Data Type> <Name>, e.g. *boolean AutoDelete* would be declared in Delphi as *AutoDelete: Boolean*.

The data type in square-brackets defines the data type for the VCL Component.

Example:

property PenStyle [integer] PenStyle

means that this property is of type "integer".
The leading "PenStyle" type declaration is for the .NET control only.

Data Types:

VpeCoord is Double

boolean is *Boolean*

integer is *Integer*

pointer is *Pointer*

long is *Long*

string is *String*

Color is *TColor* (in fact a 32-bit integer)

Properties / Methods:

The keyword *method* declares a method, e.g. *method void CloseProgressBar()*.

A method is either a Procedure or a Function, depending on how the return type of the method is declared. The keyword immediately following the word *method* declares the method.

void is a *Procedure* which does not return a value, e.g.
method void StorePos() is in Delphi: *procedure StorePos()*

all other declarations declare a function, e.g.
method boolean WriteDoc(string FileName) is in Delphi
function WriteDoc(FileName: String) : Boolean

The keyword *property* describes a property. The line below the declaration of the property explains, how the property can be accessed (read-only, write-only or read / write together) and if it can be accessed during runtime only, or also during design time.

Example:

property boolean PageScrollerTracking

read / write; design- & runtime

means: the property is of type *Boolean*, its name is *PageScrollerTracking* and it can be accessed for *read* (you can read its value, e.g. *n := PageScrollerTracking*) and for *write* (you can assign it a value, e.g. *PageScrollerTracking := True*) and it is available during runtime (while your application is running) and design time (when you are developing your forms).

It would be declared in Delphi as *PageScrollerTracking: Boolean*

1.5.2 Exceptions

In case of error conditions, the VPE-VCL might raise exceptions. We call that "*throw exceptions*".

Properties and methods of the VPE-VCL which throw exceptions are marked in this reference.

Please consult the Object Pascal and VCL documentation for further details on how to handle exceptions.

In order to throw and handle exceptions, the class *EVPEError* is defined by the VPE-VCL:

```
TVPEVCL_ERR =
(
     VPEVCL_ERR_DOC_NOT_OPEN,
     VPEVCL_ERR_DOC_IS_OPEN,
     VPEVCL_ERR_GET_VALUE,
     VPEVCL_ERR_SET_VALUE,
     VPEVCL_ERR_TPL_LOAD,
     VPEVCL_ERR_NULL_HANDLE
);

EVPEError = class(Exception)
protected
     ErrCode: TVPEVCL_ERR;

public
     constructor CreateByCode(code:TVPEVCL_ERR);

     property ErrorCode: TVPEVCL_ERR read ErrCode;
end;
```

If an exception of type *EVPEError* is raised, you can catch it and retrieve the thrown exception code with the property *EVPEError.ErrCode*.

Possible values for *ErrCode* are:

- VPEVCL_ERR_DOC_NOT_OPEN
  The VPE Document is not open. You need to call OpenDoc prior to this operation.

- VPEVCL_ERR_DOC_IS_OPEN
  The VPE Document is open. The document needs to be closed prior to this operation.
- VPEVCL_ERR_GET_VALUE
  The value could not be retrieved. Possible causes: Field/Control not found, Data Type Conversion Impossible or Out of Memory.
- VPEVCL_ERR_SET_VALUE
  The value could not be set. Possible causes: Field/Control not found, Data Type Conversion Impossible or Out of Memory.
- VPEVCL_ERR_TPL_LOAD
  The template file could not be loaded. Possible causes: file not found, wrong version or VPE Edition, or Out of Memory.
- VPEVCL_ERR_NULL_HANDLE
  The handle to the object is invalid. It does not point to a valid VPE DLL Object. Probably the method/property that returned the object has failed.

After catching the Exception Code, you can check for further error information with the property *VPE.LastError*.

Events Generated by the .NET Control

2 Events Generated by the .NET Control

The .NET Control fires several events to your application, so you have always total control about what's happening.

NOTE: You may not call CloseDoc() 193 while processing any event fired by VPE - except it is explicitly said in the decription of a particular event that it is allowed.

2.1 AfterDestroyWindow Event - .NET

[Not supported by VpeWebControl]

Is fired when the preview window was destroyed - for example closed by the user - and AutoDelete 266 is True (the default). **The document is also closed (removed from memory).**

```
EventHandler VPE.AfterDestroyWindow(
        object sender,
        System.EventArgs e
)
```

object sender
the VPE document object that fired the event

System.EventArgs e
event data, see .NET documentation

Remarks:

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

This event is fired, if AutoDelete 266 is True and the user closes the preview. Otherwise the event AfterCloseWindow 57 is fired.

2.2 Closing Event - .NET

[Not supported by VpeWebControl]

VPE requests confirmation from your application, if the preview can be closed.

CancelEventHandler VPE.Closing(
    object *sender,*
    System.ComponentModel.CancelEventArgs *e*
)

object sender
    the VPE document object that fired the event

System.ComponentModel.CancelEventArgs e
    event data, see .NET documentation
    You can set *e.Cancel* = *true* if you want to cancel the event, i.e. deny that the preview is closed

Example:

```
Protected Sub vpe_Closing(sender As Object, e As CancelEventArgs)
  If Not myDataIsSaved Then
       e.Cancel = True
       MessageBox.Show("You must save first.")
  Else
       e.Cancel = False
       MessageBox.Show("Goodbye.")
  End If
End Sub 'vpe_Closing
```

2.3 AfterCloseWindow Event - .NET

[Not supported by VpeWebControl]

Is fired, when the preview window was closed - for example by the user - and AutoDelete 266 is False (which means, that the document is not destroyed, if the preview is closed).

```
EventHandler VPE. AfterCloseWindow(
        object sender,
        System.EventArgs e
)
```

object sender
the VPE document object that fired the event

System.EventArgs e
event data, see .NET documentation

See also:
AfterDestroyWindow Event - .NET 55

2.4 BeforeOpenFile Event - .NET

[Not supported by VpeWebControl]

Is fired when the user clicked the Open File button in the toolbar (or pushed the corresponding key).

CancelEventHandler VPE. BeforeOpenFile(
object *sender,*
System.ComponentModel.CancelEventArgs *e*
)

object sender
the VPE document object that fired the event

System.ComponentModel.CancelEventArgs e
event data, see .NET documentation
You can set *e.Cancel* = *true* if you want to cancel the event, i.e. deny that the file open dialog will be shown. 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.

Example:

```
Protected Sub vpe_BeforeOpenFile (sender As Object, e As
CancelEventArgs)
   e.Cancel = True
End Sub
```

See also:
OpenFileName 223

2.5 AfterOpenFile Event - .NET

[Not supported by VpeWebControl]

Is fired after the file open operation has been completed.

```
EventHandler VPE. AfterOpenFile(
        object sender,
        System.EventArgs e
)
```

object sender
  the VPE document object that fired the event

System.EventArgs e
  event data, see .NET documentation

Remarks:
  When handling this event, your application can acquire the error status of the file open operation by reading the property LastError 201. For example, if the user clicked onto the Cancel-Button in the open file dialog, LastError will be ErrorCode.Cancelled.

See also:
  OpenFileName 223

2.6 BeforeSaveFile Event - .NET

[Not supported by VpeWebControl]

Is fired when the user clicked the Save File button in the toolbar (or pushed the corresponding key).

CancelEventHandler VPE. BeforeSaveFile(
    object *sender,*
    System.ComponentModel.CancelEventArgs *e*
)

object sender
the VPE document object that fired the event

System.ComponentModel.CancelEventArgs e
event data, see .NET documentation
You can set *e.Cancel* = *true* if you want to cancel the event, i.e. deny that the file save dialog will be shown. 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.

Example:

```
Protected Sub vpe_BeforeSaveFile (sender As Object, e As
CancelEventArgs)
   e.Cancel = True
End Sub
```

See also:
SaveFileName 224

2.7 AfterSaveFile Event - NET

[Not supported by VpeWebControl]

Is fired after the file save operation has been completed.

```
EventHandler VPE. AfterSaveFile(
        object sender,
        System.EventArgs e
)
```

object sender
 the VPE document object that fired the event

System.EventArgs e
 event data, see .NET documentation

Remarks:

When handling this event, your application can acquire the error status of the file open operation by reading the property LastError 201. For example, if the user clicked onto the Cancel-Button in the save file dialog, LastError will be ErrorCode.Cancelled.

See also:

SaveFileName 224

2.8 HelpRequested Event - .NET

[Not supported by VpeWebControl]

Is sent if the property RouteHelp 251 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 CloseDoc() 193 may be called while processing it.

```
HelpEventHandler VPE.HelpRequested(
        object sender,
        System.Windows.Forms.HelpEventArgs hlpevent
)
```

object sender
the VPE document object that fired the event

System.Windows.Forms.HelpEventArgs hlpevent
event data, see .NET documentation

2.9 AfterAutoPageBreak Event - .NET

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).

EventHandler VPE.AfterAutoPageBreak(
    object *sender,*
    System.EventArgs *e*
)

object sender
    the VPE document object that fired the event

System.EventArgs e
    event data, see .NET documentation

Remarks:

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 = Doc.CurrentPage   // remember the current page number
Doc.Print(1, 1, "... very long text ...")
if n <> Doc.CurrentPage
   catched auto break!
   The formula "Doc.CurrentPage - 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 364 = AUTO_BREAK_NO_LIMITS.

If you are modifiying any properties, as for example the AutoBreakMode or FontSize 478 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 StoreSet() 383 in the beginning of you event handler and by calling UseSet() 386 followed by RemoveSet() 387 at the end.**
Example: if a Print() 497 statement in your main code causes an AutoBreak and if your AutoBreak Handler sets the PenSize 443 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)() 499 or Write(Box)() 496 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.

2.10 RequestPrint Event - .NET

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

```
RequestPrintEventHandler VPE.RequestPrint(
        object sender,
        RequestPrintEventArgs e
)
```

object sender
the VPE document object that fired the event

RequestPrintEventArgs e
event data, derived from *EventArgs*, two additional members:

```
public class RequestPrintEventArgs : EventArgs
{
        public PrintAction Action;
        public PrintResultingAction PrintResultingAction;
}
```

PrintAction enumeration:

```
public enum PrintAction
{
        Abort,          // User aborted while printing
        Start,          // Print started
        End,            // Print ended
        SetupAbort,     // User aborted Setup-Dialog
        SetupStart,     // Setup-Dialog started
        SetupEnd,       // Setup-Dialog ended
}
```

PrintResultingAction enumeration:
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

```
public enum PrintResultingAction
{
        Ok,
        Abort,
}
```

Remarks:

Do not call CloseDoc() 193 while processing this event. You would terminate a module that is working.

When your application handles this event, the *PrintAction* member contains the current status of the printing progress. Your applications's event handler can return a value in the *PrintResultingAction* member.

Your application should return **Ok** if it processes this message, except for *PrintAction* = *SetupStart*, where your application may return in addition **Abort** to abort the print job. *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 *SetupStart* message is, to pre-initialize the Device Control Properties 314 at this stage, before the printer setup dialog will be shown to the user.

2.11 BeforePrintNewPage Event - .NET

Is fired only while printing exactly before printing a new page. The event is useful to change the Device Control Properties 314 on-the-fly during printing. You may change all properties, **except the device itself**.

```
PrintEventHandler VPE.BeforePrintNewPage(
        object sender,
        PrintEventArgs e
)
```

object sender
the VPE document object that fired the event

PrintEventArgs e
event data, derived from *EventArgs*, two additional members:

```
public class PrintEventArgs : EventArgs
{
    public int Page;
    public PrintResultingAction PrintResultingAction;
}
```

int Page
current page number that will be printed

PrintResultingAction enumeration:
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

```
public enum PrintResultingAction
{
    Ok,
    Change,
}
```

Remarks:

Your applications's event handler can return a value in the *ResultingAction* member of the *PrintEventArgs* parameter.

Your application should return **Ok**, if it processes this message without changing a printing device's properties.

If a *Device Control Property* was changed, your application must return **Change**.

- 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 RequestPrint Event – .NET 65: with PrintAction = PrintAction.Abort or PrintAction = PrintAction.End).
- Changing the properties (like DevPaperBin 339, DevOrientation 318, DevPaperFormat 319, 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 333" 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.*

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 *PrintResultingAction.Change* although no Device Control Property has been changed causes the printer to eject the page.

Returning *PrintResultingAction.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:

PrintDevData Event - .NET 70

2.12 PrintDevData Event - .NET

Is sent only while printing, exactly before printing a new page and immediately after BeforePrintNewPage() 67 has been sent. The only use for this event is to call DevSendData() 354 in response.

DevSendData() 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).

```
PrintEventHandler VPE.PrintDevData(
        object sender,
        PrintEventArgs e
)
```

object sender
the VPE document object that fired the event

PrintEventArgs e
event data, derived from *EventArgs*, two additional members:

```
public class PrintEventArgs : EventArgs
{
    public int Page;
    public PrintResultingAction PrintResultingAction;
}
```

int Page
current page number that will be printed

PrintResultingAction enumeration:
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

```
public enum PrintResultingAction
{
    Ok,
    Change,
}
```

Remarks:

When your application handles this event, the *PrintAction* member contains the current status of the printing progress. Your applications's event handler can return a value in the *PrintResultingAction* member.

Your application should return **Ok** if it processes this message without calling DevSendData().

If it called DevSendData(), your application must return **Change**.

See also:

BeforePrintNewPage Event 148

2.13 BeforeMail Event - .NET

[Not supported by VpeWebControl]

Is fired, 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.

```
CancelEventHandler VPE. BeforeMail(
        object sender,
        System.ComponentModel.CancelEventArgs e
)
```

object sender
the VPE document object that fired the event

System.ComponentModel.CancelEventArgs e
event data, see .NET documentation
You can set *e.Cancel* = *true* if you want to cancel the event, i.e. deny that the document will be mailed. Cancelling the operation allows you to display your own mail dialog and/or to execute your own mailing code.

Example:

```
Protected Sub vpe_BeforeMail (sender As Object, e As CancelEventArgs)
   e.Cancel = True
   // Here comes your own code to mail a document
End Sub
```

See also:
E-Mail Functions 592

2.14 AfterMail Event - .NET

[Not supported by VpeWebControl]

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.

EventHandler VPE.AfterMail (
    object *sender,*
    System.EventArgs *e*
)

object sender
    the VPE document object that fired the event

System.EventArgs e
    event data, see .NET documentation

Remarks:
    When handling this event, your application can acquire the error status of the mail by reading the property LastError 201.

See also:
    E-Mail Functions 592

2.15 ObjectClicked Event - .NET

[Not supported by VpeWebControl, Professional Edition and above]

A clickable object 640 with an assigned ObjectID 642 has been clicked with the mouse.

This is one of the few events where CloseDoc() 193 may be called while processing the event.

ObjectClickedEventHandler VPE.ObjectClicked(
    object *sender*,
    ObjectClickedEventArgs *e*
)

object sender
the VPE document object that fired the event

ObjectClickedEventArgs e
event data, derived from *EventArgs*, one additional member:

```
public class ObjectClickedEventArgs : EventArgs
{
    public int ObjectID;          // Object ID of the object that was clicked by the user
}
```

See also:
Clickable objects 640

2.16 UDOPaint Event - .NET

[Professional Edition and above]

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

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

```
EventHandler VPE.UDOPaint(
        object sender,
        System.EventArgs e
)
```

object sender
the VPE document object that fired the event

System.EventArgs e
event data, see .NET documentation

See also:
User Defined Object 646

2.17 AfterControlEnter - .NET

[Not supported by VpeWebControl, 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.

```
VPEObjectEventHandler VPE.AfterControlEnter(
        object sender,
        VPEObjectEventArgs e
)
```

object sender
the VPE document object that fired the event

VPEObjectEventArgs e
event data, derived from *EventArgs*, one additional member:

```
public class VPEObjectEventArgs : EventArgs
{
    public TVPEObject VpeObject;          // the VPE object that fired the event
}
```

See also:
Interactive Documents 904

2.18 RequestControlExit - .NET

[Not supported by VpeWebControl, 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.

```
ControlExitEventHandler VPE.RequestControlExit(
        object sender,
        ControlExitEventArgs e
)
```

object sender
the VPE document object, which fired the event

ControlExitEventArgs e
event data, derived from *EventArgs*, two additional members:

```
public class ControlExitEventArgs : EventArgs
{
    public TVPEObject VpeObject;            // the VPE object that fired the event
    public bool CanExit;
}
```

bool CanExit
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
True	yes, the control may loose the focus
False	no, the control may not loose the focus

See also:
Interactive Documents 904

2.19 AfterControlExit - .NET

[Not supported by VpeWebControl, 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() 916 or SetFocus() 213. 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.

VPEObjectEventHandler VPE.AfterControlExit(
object *sender,*
VPEObjectEventArgs *e*
)

object sender
the VPE document object that fired the event

VPEObjectEventArgs e
event data, derived from *EventArgs*, one additional member:

```
public class VPEObjectEventArgs : EventArgs
{
    public TVPEObject VpeObject;          // the VPE object that fired the event
}
```

See also:
Interactive Documents 904

2.20 AfterControlChange - .NET

[Not supported by VpeWebControl, 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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Control*** by code.

VPEObjectEventHandler VPE.AfterControlChange(
object *sender,*
VPEObjectEventArgs *e*
)

object sender
the VPE document object that fired the event

VPEObjectEventArgs e
event data, derived from *EventArgs*, one additional member:

```
public class VPEObjectEventArgs : EventArgs
{
    public TVPEObject VpeObject;            // the VPE object that fired the event
}
```

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 79 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.

See also:
Interactive Documents 904

2.21 AfterFieldChange - .NET

[Not supported by VpeWebControl, 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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Field*** by code.

```
VPEFieldEventHandler VPE.AfterFieldChange(
        object sender,
        VPEFieldEventArgs e
)
```

object sender
the VPE document object that fired the event

VPEFieldEventArgs e
event data, derived from *EventArgs*, one additional member:

```
public class VPEFieldEventArgs : EventArgs
{
    public TVPEField Field;          // the Field object 856 that fired the event
}
```

Remarks:
If you change the value of a Control by code, the AfterControlChange 78 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.

See also:
Interactive Documents 904

This page is intentionally left blank.

Events Generated by the Java Control

3 Events Generated by the Java Control

The Java Control fires several events to your application, so you have always total control about what's happening.

NOTE: You may not call CloseDoc() 193 while processing any event fired by VPE - except it is explicitly said in the decription of a particular event that it is allowed.

The event handling follows the Java standard. You need to implement an event listener class, which must be derived from *VpeEventAdapter*.

Depending on the edition of VPE, *VpeEventAdapter* is defined as follows:

```
// Community, Standard and Enhanced Edition
public class VpeEventAdapter implements VpeEventListener {
  public VpeEventAdapter() {}

  // GUI and Non-GUI
  public void AfterAutoPageBreak(EventObject e) {}

  // GUI only: The following methods are only present in the GUI version
  public void AfterDestroyWindow(EventObject e) {}
  public void AfterCloseWindow(EventObject e) {}
  public void AfterOpenFile(EventObject e) {}
  public void AfterSaveFile(EventObject e) {}
  public void AfterMail(EventObject e) {}
  public void Closing(CancelEvent e) {}
  public void BeforeOpenFile(CancelEvent e) {}
  public void BeforeSaveFile(CancelEvent e) {}
  public void BeforeMail(CancelEvent e) {}
  public void HelpRequested(EventObject e) {}
  public void RequestPrint(RequestPrintEvent e) {}
  public void BeforePrintNewPage(PrintEvent e) {}
  public void PrintDevData(PrintEvent e) {}
}


// Professional and Enterprise Edition
public class VpeEventAdapter implements VpeEventListener {
  public VpeEventAdapter() {}

  // GUI and Non-GUI
  public void AfterAutoPageBreak(EventObject e) {}

  // GUI only: The following methods are only present in the GUI version
  public void AfterDestroyWindow(EventObject e) {}
  public void AfterCloseWindow(EventObject e) {}
  public void AfterOpenFile(EventObject e) {}
  public void AfterSaveFile(EventObject e) {}
  public void AfterMail(EventObject e) {}
  public void Closing(CancelEvent e) {}
  public void BeforeOpenFile(CancelEvent e) {}
  public void BeforeSaveFile(CancelEvent e) {}
  public void BeforeMail(CancelEvent e) {}
  public void HelpRequested(EventObject e) {}
  public void RequestPrint(RequestPrintEvent e) {}
  public void BeforePrintNewPage(PrintEvent e) {}
  public void PrintDevData(PrintEvent e) {}

  // GUI only, Professional Edition
  public void ObjectClicked(ObjectClickedEvent e) {}
}


// Interactive Edition
public class VpeEventAdapter implements VpeEventListener {
  public VpeEventAdapter() {}

  // GUI and Non-GUI
  public void AfterAutoPageBreak(EventObject e) {}

  // GUI only: The following methods are only present in the GUI version
  public void AfterDestroyWindow(EventObject e) {}
  public void AfterCloseWindow(EventObject e) {}
  public void AfterOpenFile(EventObject e) {}
  public void AfterSaveFile(EventObject e) {}
  public void AfterMail(EventObject e) {}
  public void Closing(CancelEvent e) {}
  public void BeforeOpenFile(CancelEvent e) {}
  public void BeforeSaveFile(CancelEvent e) {}
  public void BeforeMail(CancelEvent e) {}
  public void HelpRequested(EventObject e) {}
  public void RequestPrint(RequestPrintEvent e) {}
  public void BeforePrintNewPage(PrintEvent e) {}
  public void PrintDevData(PrintEvent e) {}

  // GUI only, Professional Edition
  public void ObjectClicked(ObjectClickedEvent e) {}

  // GUI only, Interactive Edition
  public void AfterControlEnter(VPEObjectEvent e) {}
  public void AfterControlExit(VPEObjectEvent e) {}
  public void AfterControlChange(VPEObjectEvent e) {}
  public void RequestControlExit(ControlExitEvent e) {}
  public void AfterFieldChange(VPEFieldEvent e) {}
}
```

Each event is explained in detail throughout the following sections.

The event handling is shown in practice in the Java demo source codes, which are installed together with VPE. Basically, derive a class from *VpeEventAdapter* and overload the methods for which you want to receive events.

Assuming your application is a class named *MyApplication*, your event listener could be:

```
class MyApplication_Listener extends VpeEventAdapter {
  MyApplication adaptee;

  MyApplication_Listener(MyApplication adaptee) {
    this.adaptee = adaptee;
  }

  // In this example we are only interested in the
  // AfterDestroyWindow event
  public void AfterDestroyWindow(EventObject e) {
    super.AfterDestroyWindow(e);
    adaptee.AfterDestroyWindow(e);
  }
}
```

Next, implement for your class *MyApplication* the method:

```
  void AfterDestroyWindow(EventObject e)
```

which is your event handler.

After creating an instance of the VPE Control, you need to set the event listener (if an event listener is required):

```
   Doc = new VpeControl();
   Doc.setVpeListener(new MyApplication_Listener(this));
```

The VPE Control also implements an adapter class to test events:

```
       public class VpeEventTestAdapter implements VpeEventListener
```

This class dumps the names of occurring events to System.out. To use this class, implement your event listener as follows:

```
       class MyApplication_Listener extends VpeEventTestAdapter
```

For practical examples, please see the Java demo source codes, which are installed together with VPE.

3.1 AfterDestroyWindow Event - Java

[GUI version only]

Is fired when the preview window was destroyed - for example closed by the user - and AutoDelete 266 is True (the default). **The document is also closed (removed from memory).**

```
public void AfterDestroyWindow(
        EventObject e
)
```

EventObject e
the event object, see Java documentation

Remarks:

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

This event is fired, if AutoDelete 266 is True and the user closes the preview. Otherwise the event AfterCloseWindow 87 is fired.

3.2 Closing Event - Java

[GUI version only]

VPE requests confirmation from your application, if the preview can be closed.

public void Closing(
  CancelEvent *e*
)

CancelEvent e

The cancel event is defined as:

```
public class CancelEvent extends java.util.EventObject {
  private boolean cancel = false;

  public boolean getCancel() {
    return cancel;
  }

  public void setCancel(boolean cancel) {
    this.cancel = cancel;
  }

  public CancelEvent(VpeControl source) {
    super(source);
  }
}
```

You can call *e.setCancel(true)* if you want to cancel the event, i.e. deny that the preview is closed

Example:

```
public void Closing(CancelEvent e) {
   if (!myDataIsSaved) {
         e.setCancel(true);
         JOptionPane.showMessageDialog(frame, "You must save first.");
   }
   else {
         e.setCancel(false);
         JOptionPane.showMessageDialog(frame, "Goodbye.");
   }
}
```

3.3 AfterCloseWindow Event - Java

[GUI version only]

Is fired, when the preview window was closed - for example by the user - and AutoDelete 266 is False (which means, that the document is not destroyed, if the preview is closed).

```
public void AfterCloseWindow(
        EventObject e
)
```

EventObject e
the event object, see Java documentation

See also:
AfterDestroyWindow Event - Java 85

3.4 BeforeOpenFile Event - Java

[GUI version only]

Is fired when the user clicked the Open File button in the toolbar (or pushed the corresponding key).

public void BeforeOpenFile(
CancelEvent *e*
)

CancelEvent e

The cancel event is defined as:

```
public class CancelEvent extends java.util.EventObject {
  private boolean cancel = false;

  public boolean getCancel() {
    return cancel;
  }

  public void setCancel(boolean cancel) {
    this.cancel = cancel;
  }

  public CancelEvent(VpeControl source) {
    super(source);
  }
}
```

You can call *e.setCancel(true)* if you want to cancel the event, i.e. deny that the file open dialog will be shown. 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.

Example:

```
public void BeforeOpenFile(CancelEvent e) {
  e.setCancel(true);
}
```

See also:

OpenFileName 223

3.5 AfterOpenFile Event - Java

[GUI version only]

Is fired after the file open operation has been completed.

```
public void AfterOpenFile(
        EventObject e
)
```

EventObject e
the event object, see Java documentation

Remarks:
When handling this event, your application can acquire the error status of the file open operation by reading the property LastError 201. For example, if the user clicked onto the Cancel-Button in the open file dialog, LastError will be ErrorCode.Cancelled.

See also:
OpenFileName 223

3.6 BeforeSaveFile Event - Java

[GUI version only]

Is fired when the user clicked the Save File button in the toolbar (or pushed the corresponding key).

public void BeforeSaveFile(
CancelEvent *e*
)

CancelEvent e

The cancel event is defined as:

```
public class CancelEvent extends java.util.EventObject {
  private boolean cancel = false;

  public boolean getCancel() {
    return cancel;
  }

  public void setCancel(boolean cancel) {
    this.cancel = cancel;
  }

  public CancelEvent(VpeControl source) {
    super(source);
  }
}
```

You can call *e.setCancel(true)* if you want to cancel the event, i.e. deny that the file save dialog will be shown. 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.

Example:

```
public void BeforeSaveFile(CancelEvent e) {
  e.setCancel(true);
}
```

See also:

SaveFileName 224

3.7 AfterSaveFile Event - NET

[GUI version only]

Is fired after the file save operation has been completed.

```
public void AfterSaveFile(
        EventObject e
)
```

EventObject e
  the event object, see Java documentation

Remarks:
  When handling this event, your application can acquire the error status of the file open operation by reading the property LastError 201. For example, if the user clicked onto the Cancel-Button in the save file dialog, LastError will be ErrorCode.Cancelled.

See also:
  SaveFileName 224

3.8 HelpRequested Event - Java

[GUI version only]

Is sent if the property RouteHelp 251 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 CloseDoc() 193 may be called while processing it.

```
public void HelpRequested(
        EventObject e
)
```

EventObject e
the event object, see Java documentation

3.9 AfterAutoPageBreak Event - Java

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).

public void AfterAutoPageBreak(
    EventObject *e*
)

EventObject e
    the event object, see Java documentation

Remarks:

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 = Doc.CurrentPage   // remember the current page number
Doc.Print(1, 1, "... very long text ...")
if n <> Doc.CurrentPage
   catched auto break!
   The formula "Doc.CurrentPage - 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 364 = AUTO_BREAK_NO_LIMITS.

If you are modifiying any properties, as for example the AutoBreakMode or FontSize 478 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 StoreSet() 383 in the beginning of you event handler and by calling UseSet() 386 followed by RemoveSet() 387 at the end.**
Example: if a Print() 497 statement in your main code causes an AutoBreak and if your AutoBreak Handler sets the PenSize 443 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)() 499 or Write(Box)() 496 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.

3.10 RequestPrint Event - Java

[GUI version only]

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

public void RequestPrint(
RequestPrintEvent *e*
)

RequestPrintEvent e

The request print event is defined as:

```
public class RequestPrintEvent extends java.util.EventObject {
  private int printAction;
  private int printResultingAction = PrintResultingAction.Ok;

  public void setPrintResultingAction(int printResultingAction) {
    this.printResultingAction = printResultingAction;
  }

  public int getPrintResultingAction() {
    return printResultingAction;
  }

  public int getPrintAction() {
    return printAction;
  }

  public RequestPrintEvent(VpeControl source, int printAction) {
    super(source);
    this.printAction = printAction;
  }
}
```

printAction has one of the following values:

```
public class PrintAction
{
        public static final Abort = 0;          // User aborted while printing
        public static final Start = 1;          // Print started
        public static final End = 2;            // Print ended
        public static final SetupAbort = 3;     // User aborted Setup-Dialog
        public static final SetupStart = 4;     // Setup-Dialog started
        public static final SetupEnd = 5;       // Setup-Dialog ended
}
```

printResultingAction is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

```
public class PrintResultingAction
{
        public static final Ok = 0;
        public static final Abort = 1;
}
```

Remarks:

Do not call CloseDoc() 193 **while processing this event. You would terminate a module that is working.**

When your application handles this event, the *printAction* member contains the current status of the printing progress. Your applications's event handler can return a value in the *printResultingAction* member.

Your application should return **Ok** if it processes this message, except for *PrintAction* = *SetupStart*, where your application may return in addition **Abort** to abort the print job. *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 *SetupStart* message is, to pre-initialize the Device Control Properties 314 at this stage, before the printer setup dialog will be shown to the user.

3.11 BeforePrintNewPage Event - Java

[GUI version only]

Is fired only while printing exactly before printing a new page. The event is useful to change the Device Control Properties 314 on-the-fly during printing. You may change all properties, **except the device itself**.

public void BeforePrintNewPage(
PrintEvent *e*
)

PrintEvent e

The print event is defined as:

```
public class PrintEvent extends java.util.EventObject {
  private int page;
  private int printResultingAction=PrintResultingAction.Ok;

  public void setPrintResultingAction(int printResultingAction) {
    this.printResultingAction = printResultingAction;
  }

  public int getPrintResultingAction() {
    return printResultingAction;
  }

  public int getPage() {
    return page;
  }

  public PrintEvent(VpeControl source, int page) {
    super(source);
    this.page = page;
  }
}
```

int page
current page number that will be printed

printResultingAction is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

```
public class PrintResultingAction
{
	public static final Ok = 0;
	public static final Change = 1;
}
```

Remarks:

Your applications's event handler can return a value in the *printResultingAction* member of the *PrintEvent* parameter.

Your application should return **Ok**, if it processes this message without changing a printing device's properties.

If a *Device Control Property* was changed, your application must return **Change**.

- 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 RequestPrint Event – Java 95: with printAction = Abort or printAction = End).
- Changing the properties (like DevPaperBin 339, DevOrientation 318, DevPaperFormat 319, 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 333" 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.*

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 *PrintResultingAction.Change* although no Device Control Property has been changed causes the printer to eject the page.

Returning *PrintResultingAction.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:

PrintDevData Event - Java 100

3.12 PrintDevData Event - Java

[GUI version only]

Is sent only while printing, exactly before printing a new page and immediately after BeforePrintNewPage() 97 has been sent. The only use for this event is to call DevSendData() 354 in response.

DevSendData() 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).

public void PrintDevData(
PrintEvent *e*
)

PrintEvent e
The print event is defined as:

```
public class PrintEvent extends java.util.EventObject {
  private int page;
  private int printResultingAction=PrintResultingAction.Ok;

  public void setPrintResultingAction(int printResultingAction) {
    this.printResultingAction = printResultingAction;
  }

  public int getPrintResultingAction() {
    return printResultingAction;
  }

  public int getPage() {
    return page;
  }

  public PrintEvent(VpeControl source, int page) {
    super(source);
    this.page = page;
  }
}
```

int page
current page number that will be printed

printResultingAction is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

```
public class PrintResultingAction
{
	public static final Ok = 0;
	public static final Change = 1;
}
```

Remarks:
When your application handles this event, the *PrintAction* member contains the current status of the printing progress. Your applications's event handler can return a value in the *PrintResultingAction* member.

Your application should return **Ok** if it processes this message without calling DevSendData().

If it called DevSendData(), your application must return **Change**.

See also:

BeforePrintNewPage Event 148

3.13 BeforeMail Event - Java

[GUI version only]

Is fired, 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.

public void BeforeMail(
    CancelEvent *e*
)

CancelEvent e

The cancel event is defined as:

```
public class CancelEvent extends java.util.EventObject {
  private boolean cancel = false;

  public boolean getCancel() {
    return cancel;
  }

  public void setCancel(boolean cancel) {
    this.cancel = cancel;
  }

  public CancelEvent(VpeControl source) {
    super(source);
  }
}
```

You can call *e.setCancel(true)* if you want to cancel the event, i.e. deny that the document will be mailed. Cancelling the operation allows you to display your own mail dialog and/or to execute your own mailing code.

Example:

```
public void BeforeMail(CancelEvent e) {
   e.setCancel(true);
   // Here comes your own code to mail a document
}
```

See also:

E-Mail Functions 592

3.14 AfterMail Event - Java

[GUI version 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.

```
public void AfterMail(
        EventObject e
)
```

EventObject e
the event object, see Java documentation

Remarks:
When handling this event, your application can acquire the error status of the mail by reading the property LastError 201.

See also:
E-Mail Functions 592

3.15 ObjectClicked Event - Java

[GUI version only, Professional Edition and above]

A clickable object 640 with an assigned ObjectID 642 has been clicked with the mouse.

This is one of the few events where CloseDoc() 193 may be called while processing the event.

public void ObjectClicked(
        ObjectClickedEvent *e*
)

ObjectClickedEvent e

The object clicked event is defined as:

```
public class ObjectClickedEvent extends java.util.EventObject {
  private int objectID;

  public int getObjectID() {
    return objectID;
  }

  public ObjectClickedEvent(VpeControl source, int ObjectID) {
    super(source);
    this.objectID = objectID;
  }
}
```

objectID is the Object ID of the object that was clicked by the user

See also:

Clickable objects 640

3.16 AfterControlEnter - Java

[GUI version 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.

public void AfterControlEnter(
VPEObjectEvent *e*
)

VPEObjectEvent e

The VPE object event is defined as:

```
public class VPEObjectEvent extends java.util.EventObject{
  private TVPEObject VpeObject;

  public TVPEObject getVpeObject() {
    return VpeObject;
  }

  public VPEObjectEvent(VpeControl source, TVPEObject VpeObject)
{
    super(source);
    this.VpeObject = VpeObject;
  }
}
```

VpeObject is the VPE object that fired the event

See also:

Interactive Documents 904

3.17 RequestControlExit - Java

[GUI version 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.

public void RequestControlExit(
ControlExitEvent *e*
)

ControlExitEvent e
The control exit event is defined as:

```
public class ControlExitEvent extends VPEObjectEvent {
  private boolean canExit = true;

  public boolean getCanExit()
  {
    return canExit;
  }

  public void setCanExit(boolean canExit)
  {
    this.canExit = canExit;
  }

  public ControlExitEvent(VpeControl source, TVPEObject
VpeObject) {
    super(source, VpeObject);
  }
}
```

VpeObject is the VPE object that fired the event

boolean CanExit
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
true	yes, the control may loose the focus
false	no, the control may not loose the focus

See also:
Interactive Documents 904

3.18 AfterControlExit - Java

[GUI version 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() 916 or SetFocus() 213. 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.

public void AfterControlExit(
VPEObjectEvent *e*
)

VPEObjectEvent e
The VPE object event is defined as:

```
public class VPEObjectEvent extends java.util.EventObject{
  private TVPEObject VpeObject;

  public TVPEObject getVpeObject() {
    return VpeObject;
  }

  public VPEObjectEvent(VpeControl source, TVPEObject VpeObject)
{
    super(source);
    this.VpeObject = VpeObject;
  }
}
```

VpeObject is the VPE object that fired the event

See also:
Interactive Documents 904

3.19 AfterControlChange - Java

[GUI version 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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Control*** by code.

public void AfterControlChange(
VPEObjectEvent *e*
)

VPEObjectEvent e

The VPE object event is defined as:

```
public class VPEObjectEvent extends java.util.EventObject{
  private TVPEObject VpeObject;

  public TVPEObject getVpeObject() {
    return VpeObject;
  }

  public VPEObjectEvent(VpeControl source, TVPEObject VpeObject)
{
    super(source);
    this.VpeObject = VpeObject;
  }
}
```

VpeObject is the VPE object that fired the event

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 109 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.

See also:

Interactive Documents 904

3.20 AfterFieldChange - Java

[GUI version 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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Field*** by code.

public void AfterFieldChange(
VPEFieldEvent *e*
)

VPEFieldEvent e

The VPE field event is defined as:

```
public class VPEFieldEvent extends java.util.EventObject {
  private TVPEField VpeField;

  public TVPEField getVpeField() {
    return VpeField;
  }

  public VPEFieldEvent(VpeControl source, TVPEField VpeField) {
    super(source);
    this.VpeField = VpeField;
  }
}
```

VpeField is the Field object 856 that fired the event

Remarks:

If you change the value of a Control by code, the AfterControlChange 108 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.

See also:

Interactive Documents 904

This page is intentionally left blank.

Events Generated by the Python Control

4 Events Generated by the Python Control

The Python VPE Control fires several events to your application, so you have control about what's happening.

Event handlers are implemented as methods in the class VpeControl(). To override an event handler, derive your own class from VpeControl() and override the desired event handler method.

Example:

```
import os, sys
from time import sleep
from VpeControl import *

class MyVpeControl(VpeControl):
   def RequestClose(self):
         print("MyVpeControl: RequestClose() is called")
         return 0

doc = MyVpeControl()
doc.OpenDoc()
doc.Print(1, 1, "Hello World!")
doc.WriteDoc("hello world.pdf")

# Note that the DispatchAllMessages()-loop is required to pump
messages from the
# Windows operating system to the preview. This keeps the preview
alive.
doc.Preview()
abort = False
while not abort:
   abort = doc.DispatchAllMessages()
   sleep(0.01)
```

In the above example, the event handler `RequestClose()` is called, when you close the preview window.

On the following pages you will find a complete list of events generated by VPE.

4.1 AfterDestroyWindow Event - Python

Is fired when the preview window was destroyed - for example closed by the user - and AutoDelete 266 is True (the default). **The document is also closed (removed from memory).**

integer AfterDestroyWindow()

Remarks:

The event handler should return zero.

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

This event is fired, if AutoDelete 266 is True and the user closes the preview. Otherwise the event AfterCloseWindow 139 is fired.

4.2 RequestClose Event - Python

VPE requests confirmation from your application, if the preview can be closed.

integer RequestClose()

Remarks:

The event handler should return:

Value	Description
0	the preview may be closed
1	cancel operation, i.e. do not close the preview

4.3 AfterCloseWindow Event - Python

Is fired, when the preview window was closed - for example by the user - and AutoDelete 266 is False (which means, that the document is not destroyed if the preview is closed).

integer AfterCloseWindow()

Remarks:

The event handler should return zero.

See also:

AfterDestroyWindow Event 137

4.4 BeforeOpenFile Event - Python

Is fired when the user clicked the Open File button in the toolbar (or pushed the corresponding key).

integer BeforeOpenFile()

Remarks:

The event handler should return:

Value	Description
0	continue operation
1	cancel the event, i.e. deny that the file open dialog will be shown. 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:

OpenFileName 223

4.5 AfterOpenFile Event - Python

Is fired after the file open operation has been completed.

integer AfterOpenFile(
  Result as integer
)

Result
  the status of the operation, one of the VERR_xyz error codes. For example, if the user clicked onto the Cancel-Button in the open file dialog, *Result* will be VERR_CANCELLED.

Remarks:
  The event handler should return zero.

See also:
  OpenFileName 223

4.6 BeforeSaveFile Event - Python

Is fired when the user clicked the Save File button in the toolbar (or pushed the corresponding key).

integer BeforeSaveFile()

Remarks:

The event handler should return:

Value	Description
0	continue operation
1	cancel the event, i.e. deny that the file save dialog will be shown. 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:

SaveFileName 224

4.7 AfterSaveFile Event - Python

Is fired after the file save operation has been completed.

```
integer AfterSaveFile(
        Result as integer
)
```

Result

the status of the operation, one of the VERR_xyz error codes. For example, if the user clicked onto the Cancel-Button in the save file dialog, *Result* will be VERR_CANCELLED.

Remarks:

The event handler should return zero.

See also:

SaveFileName 224

4.8 DoHelp Event - Python

Is sent if the property RouteHelp 251 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 CloseDoc() 193 may be called while processing it.

integer DoHelp()

Remarks:

The event handler should return zero.

4.9 AfterAutoPageBreak Event - Python

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).

integer AfterAutoPageBreak()

Remarks:

The event handler should return zero.

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 = Doc.CurrentPage   // remember the current page number
Doc.Print(1, 1, "... very long text ...")
if n <> Doc.CurrentPage
   catched auto break!
   The formula "Doc.CurrentPage - 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 364 = AUTO_BREAK_NO_LIMITS.

If you are modifiying any properties, as for example the AutoBreakMode or FontSize 478 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 StoreSet() 383 in the beginning of you event handler and by calling UseSet() 386 followed by RemoveSet() 387 at the end.**
Example: if a Print() 497 statement in your main code causes an AutoBreak and if your AutoBreak Handler sets the PenSize 443 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)() 499 or Write(Box)() 496 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.

4.10 RequestPrint Event - Python

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

integer RequestPrint(
Action as integer
)

Action

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

Remarks:

The event handler should return:

Resulting Action	Value
PRINT_ACTION_OK	0
PRINT_ACTION_ABORT	1

Do not call CloseDoc() 193 while 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 Action = 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 314 at this stage, before the printer setup dialog will be shown to the user.

4.11 BeforePrintNewPage Event - Python

Is fired only while printing exactly before printing a new page. The event is useful to change the Device Control Properties 314 on-the-fly during printing. You may change all properties, **except the device itself**.

integer BeforePrintNewPage(
    PageNumber as integer
)

PageNumber
current page number that will be printed

Remarks:
The event handler should return:

Resulting Action	Value
PRINT_ACTION_OK	0
PRINT_ACTION_CHANGE	1

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 RequestPrint Event – ActiveX 147: with Action = PRINT_MSG_ABORT or Action = PRINT_MSG_END).
- Changing the properties (like DevPaperBin 339, DevOrientation 318, DevPaperFormat 319, 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 333" 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:

DoPrintDevData Event 150

4.12 DoPrintDevData Event - Python

Is sent only while printing, exactly before printing a new page and immediately after BeforePrintNewPage() 148 has been sent. The only use for this event is to call DevSendData() 354 in response.

DevSendData 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).

integer DoPrintDevData(
        PageNumber as integer
)

PageNumber
    current page number that will be printed

Remarks:
    The event handler should return:

Resulting Action	Value
PRINT_ACTION_OK	0
PRINT_ACTION_CHANGE	1

Your application should return **PRINT_ACTION_OK** (zero) if it processes this message without calling DevSendData().

If it called DevSendData(), your application must return **PRINT_ACTION_CHANGE** (= 1).

See also:
    BeforePrintNewPage Event 148

4.13 BeforeMail Event - Python

Is fired, 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.

integer BeforeMail()

Remarks:

The event handler should return:

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

If you set this parameter to True, the operation is cancelled, i.e. VPE will 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 592

4.14 AfterMail Event - Python

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.

```
integer AfterMail(
        Result as integer
)
```

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

Remarks:
The event handler should return zero.

See also:
E-Mail Functions 592

4.15 DoObjectClicked Event - Python

[Professional Edition and above]

A clickable object 640 with an assigned ObjectID 642 has been clicked with the mouse.

This is one of the few events where CloseDoc() 193 may be called while processing the event.

```
integer DoObjectClicked(
        ObjectID as integer
)
```

ObjectID
the Object ID of the object that was clicked by the user

Remarks:
The event handler should return zero.

See also:
Clickable Objects 640

4.16 AfterControlEnter - Python

[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.

integer AfterControlEnter(
    Obj as TVPEObject
)

Obj
    the object, which fired the event

Remarks:
    The event handler should return zero.

See also:
    Interactive Documents 904

4.17 RequestControlExit - Python

[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.

```
integer RequestControlExit(
        Obj as TVPEObject
)
```

Obj
the object, which fired the event

Remarks:
The event handler 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 904

4.18 AfterControlExit - Python

[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() 916 or SetFocus() 213. 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.

```
integer AfterControlExit(
        Obj as TVPEObject
)
```

Obj
the object, which fired the event

Remarks:
The event handler should return zero.

See also:
Interactive Documents 904

4.19 AfterControlChange - Python

[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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Control*** by code.

```
integer AfterControlChange(
        Obj as TVPEObject
)
```

Obj
the object, which fired the event

Remarks:
The event handler should return zero.

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 159 is fired. Vice versa, if you change the value of a Field by code, the AfterFieldChange 159 event is not fired, but any Controls associated with the Field will fire the event AfterControlChange.

See also:
Interactive Documents 904

4.20 AfterFieldChange - Python

[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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Field*** by code.

integer AfterFieldChange(
    Obj as TVPEField
)

Obj
    the Field object 856, which fired the event

Remarks:
The event handler should return zero.

If you change the value of a Control by code, the AfterControlChange 158 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 158 .

See also:
Interactive Documents 904

Events Generated by the ActiveX

5 Events Generated by the ActiveX

The VPE-ActiveX fires several events to your application, so you have always total control about what's happening.

NOTE: You may not call CloseDoc() 193 while processing any event fired by VPE - except it is explicitly said in the decription of a particular event that it is allowed.

5.1 AfterDestroyWindow Event

Is fired when the preview window was destroyed - for example closed by the user - and AutoDelete 266 is True (the default). **The document is also closed (removed from memory).**

Remarks:

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

This event is fired, if AutoDelete 266 is True and the user closes the preview. Otherwise the event AfterCloseWindow 139 is fired.

5.2 RequestClose Event

VPE requests confirmation from your application, if the preview can be closed.

OnRequestClose(
        CanClose as Boolean
)

CanClose
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
False	no, the preview may not be closed
True	Otherwise

5.3 AfterCloseWindow Event

Is fired, when the preview window was closed - for example by the user - and AutoDelete 266 is False (which means, that the document is not destroyed if the preview is closed).

See also:

AfterDestroyWindow Event 137

5.4 BeforeOpenFile Event

Is fired when the user clicked the Open File button in the toolbar (or pushed the corresponding key).

BeforeOpenFile(
    Cancel as Boolean
)

Cancel
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
False	continue operation
True	cancel the event, i.e. deny that the file open dialog will be shown. 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.

Example:

```
Protected Sub vpe_BeforeOpenFile (Cancel as Boolean)
   Cancel = True
End Sub
```

See also:
OpenFileName 223

5.5 AfterOpenFile Event

Is fired after the file open operation has been completed.

```
AfterOpenFile(
        ByVal Result As Long
)
```

Result
the status of the operation, one of the VERR_xyz error codes. For example, if the user clicked onto the Cancel-Button in the open file dialog, *Result* will be VERR_CANCELLED.

See also:
OpenFileName 223

5.6 BeforeSaveFile Event

Is fired when the user clicked the Save File button in the toolbar (or pushed the corresponding key).

BeforeSaveFile(
Cancel as Boolean
)

Cancel
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
False	continue operation
True	cancel the event, i.e. deny that the file save dialog will be shown. 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.

Example:

```
Protected Sub vpe_BeforeSaveFile (Cancel as Boolean)
   Cancel = True
End Sub
```

See also:
SaveFileName 224

5.7 AfterSaveFile Event

Is fired after the file save operation has been completed.

```
AfterSaveFile(
        ByVal Result As Long
)
```

Result
the status of the operation, one of the VERR_xyz error codes. For example, if the user clicked onto the Cancel-Button in the save file dialog, *Result* will be VERR_CANCELLED.

See also:
SaveFileName 224

5.8 DoHelp Event

Is sent if the property RouteHelp 251 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 CloseDoc() 193 may be called while processing it.

5.9 AfterAutoPageBreak Event

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).

Remarks:

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 = Doc.CurrentPage  // remember the current page number
Doc.Print(1, 1, "... very long text ...")
if n <> Doc.CurrentPage
   catched auto break!
   The formula "Doc.CurrentPage - 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 364 = AUTO_BREAK_NO_LIMITS.

If you are modifiying any properties, as for example the AutoBreakMode or FontSize 478 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 StoreSet() 383 in the beginning of you event handler and by calling UseSet() 386 followed by RemoveSet() 387 at the end.**
Example: if a Print() 497 statement in your main code causes an AutoBreak and if your AutoBreak Handler sets the PenSize 443 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)() 499 or Write(Box)() 496 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.

5.10 RequestPrint Event - ActiveX

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

```
RequestPrint(
        ByVal Action As Long,
        ResultingAction As Long
)
```

Action

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

ResultingAction

is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Resulting Action	Value
PRINT_ACTION_OK	0
PRINT_ACTION_ABORT	1

Remarks:

Do not call CloseDoc() 193 while 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 Action = 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 314 at this stage, before the printer setup dialog will be shown to the user.

5.11 BeforePrintNewPage Event

Is fired only while printing exactly before printing a new page. The event is useful to change the Device Control Properties 314 on-the-fly during printing. You may change all properties, **except the device itself**.

```
BeforePrintNewPage(
        ByVal Page As Long,
        long ResultingAction
)
```

Page
current page number that will be printed

ResultingAction
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Resulting Action	Value
PRINT_ACTION_OK	0
PRINT_ACTION_CHANGE	1

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 RequestPrint Event – ActiveX 147: with Action = PRINT_MSG_ABORT or Action = PRINT_MSG_END).
- Changing the properties (like DevPaperBin 339, DevOrientation 318, DevPaperFormat 319, 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 333" 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:

DoPrintDevData Event 150

5.12 DoPrintDevData Event

Is sent only while printing, exactly before printing a new page and immediately after BeforePrintNewPage() 148 has been sent. The only use for this event is to call DevSendData() 354 in response.

DevSendData 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).

DoPrintDevData(
  ByVal *Page* As Long,
  long *ResultingAction*
)

Page
  current page number that will be printed

ResultingAction
  is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Resulting Action	Value
PRINT_ACTION_OK	0
PRINT_ACTION_CHANGE	1

Remarks:
  Your application should return **PRINT_ACTION_OK** (zero) if it processes this message without calling DevSendData().

  If it called DevSendData(), your application must return **PRINT_ACTION_CHANGE** (= 1).

See also:
  BeforePrintNewPage Event 148

5.13 BeforeMail Event

Is fired, 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.

```
BeforeMail(
        Cancel As Boolean
)
```

Cancel

is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
True	cancel operation i.e. do not mail the document
False	continue operation

If you set this parameter to True, the operation is cancelled, i.e. VPE will 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 592

5.14 AfterMail Event

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.

```
AfterMail(
        ByVal Result As Long
)
```

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

See also:
E-Mail Functions 592

5.15 DoObjectClicked Event

[Professional Edition and above]

A clickable object 640 with an assigned ObjectID 642 has been clicked with the mouse.

This is one of the few events where CloseDoc() 193 may be called while processing the event.

```
DoObjectClicked(
        ByVal ObjectID As Long
)
```

ObjectID
the Object ID of the object that was clicked by the user

See also:
Clickable Objects 640

5.16 DoUDOPaint Event

[Professional Edition and above]

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

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

```
DoUDOPaint(
)
```

See also:

User Defined Objects 646

5.17 AfterControlEnter

[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.

```
AfterControlEnter(
        ByVal Obj As TVPEObject
)
```

Obj
  the object, which fired the event

See also:
  Interactive Documents 904

5.18 RequestControlExit

[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.

RequestControlExit(
    ByVal *Obj* As TVPEObject,
    CanExit As Boolean
)

Obj
the object, which fired the event

CanExit
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
True	yes, the control may loose the focus
False	no, the control may not loose the focus

See also:
Interactive Documents 904

5.19 AfterControlExit

[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() 916 or SetFocus() 213. 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.

```
AfterControlExit(
        ByVal Obj As TVPEObject
)
```

Obj
the object, which fired the event

See also:
Interactive Documents 904

5.20 AfterControlChange

[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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Control*** by code.

```
AfterControlChange(
        ByVal Obj As TVPEObject
)
```

Obj
the object, which fired the event

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 159 is fired.
Vice versa, if you change the value of a Field by code, the AfterFieldChange 159 event is not fired, but any Controls associated with the Field will fire the event AfterControlChange.

See also:
Interactive Documents 904

5.21 AfterFieldChange

[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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Field*** by code.

```
AfterFieldChange(
        ByVal Obj As TVPEField
)
```

Obj
the Field object 856, which fired the event

Remarks:
If you change the value of a Control by code, the AfterControlChange 158 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 158
.

See also:
Interactive Documents 904

This page is intentionally left blank.

Events Generated by the VCL

6 Events Generated by the VCL

The VPE-VCL (TVPEngine) fires several events to your application, so you have always total control about what's happening.

NOTE: You may not call CloseDoc() 193 while processing any event fired by VPE - except it is explicitly said in the decription of a particular event that it is allowed.

6.1 OnDestroyWindow Event

Is fired when the preview window was destroyed - for example closed by the user - and AutoDelete 266 is True (the default). **The document is also closed (removed from memory).**

```
OnDestroyWindow(
        Sender: TVPEngine
)
```

Sender
the VPE object that fired the event

Remarks:

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

This event is fired, if AutoDelete 266 is True and the user closes the preview. Otherwise the event OnCloseWindow 165 is fired.

6.2 OnRequestClose Event

VPE requests confirmation from your application, if the preview can be closed.

```
OnRequestClose(
        Sender: TVPEngine;
        var CanClose: Boolean
)
```

Sender
the VPE object that fired the event

CanClose
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
False	no, the preview may not be closed
True	Otherwise

6.3 OnCloseWindow Event

Is fired, when the preview window was closed - for example by the user - and AutoDelete 266 is False (which means, that the document is not destroyed if the preview is closed).

```
OnCloseWindow(
        Sender: TVPEngine
)
```

Sender
the VPE object that fired the event

See also:
OnDestroyWindow Event 163

6.4 OnBeforeOpenFile Event

Is fired when the user clicked the Open File button in the toolbar (or pushed the corresponding key).

```
OnBeforeOpenFile(
        Sender: TVPEngine;
        var Cancel: Boolean
)
```

Sender
the VPE object that fired the event

Cancel
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
False	continue operation
True	cancel the event, i.e. deny that the file open dialog will be shown. 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:
OpenFileName 223

6.5 OnAfterOpenFile Event

Is fired after the file open operation has been completed.

```
OnAfterOpenFile(
        Sender: TVPEngine;
        Result: Integer
)
```

Sender
the VPE object that fired the event

Result
the status of the operation, one of the VERR_xyz error codes. For example, if the user clicked onto the Cancel-Button in the open file dialog, *Result* will be VERR_CANCELLED.

See also:
OpenFileName 223

6.6 OnBeforeSaveFile Event

Is fired when the user clicked the Save File button in the toolbar (or pushed the corresponding key).

```
OnBeforeSaveFile(
        Sender: TVPEngine;
        var Cancel: Boolean
)
```

Sender
the VPE object that fired the event

Cancel
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
False	continue operation
True	cancel the event, i.e. deny that the file save dialog will be shown. 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:
SaveFileName 224

6.7 OnAfterSaveFile Event

Is fired after the file save operation has been completed.

```
OnAfterSaveFile(
        Sender: TVPEngine;
        Result: Integer
)
```

Sender
the VPE object that fired the event

Result
the status of the operation, one of the VERR_xyz error codes. For example, if the user clicked onto the Cancel-Button in the save file dialog, *Result* will be VERR_CANCELLED.

See also:
SaveFileName 224

6.8 OnHelp Event

Is sent if the property RouteHelp 251 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 CloseDoc() 193 may be called while processing it.

```
OnHelp(
        Sender: TVPEngine
)
```

Sender
the VPE object that fired the event

6.9 OnAutoPageBreak Event

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).

OnAutoPageBreak(
    Sender: TVPEngine
)

Sender
the VPE object that fired the event

Remarks:

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 := Doc.CurrentPage; // remember the current page number
Doc.Print(1, 1, '... very long text ...');
if n <> Doc.CurrentPage then
begin
   catched auto break!
   The formula "Doc.CurrentPage - n" returns the number of
   automatically generated pages.
end;
```

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 364 = AUTO_BREAK_NO_LIMITS.

If you are modifiying any properties, as for example the AutoBreakMode or FontSize 478 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 StoreSet() 383 in the beginning of you event handler and by calling UseSet() 386 followed by RemoveSet() 387 at the end.**
Example: if a Print() 497 statement in your main code causes an AutoBreak and if your AutoBreak Handler sets the 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)() 499 or Write(Box)() 496 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.

6.10 OnRequestPrint Event - VCL

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

OnRequestPrint(
Sender: TVPEngine;
Action: Integer;
var *ResultingAction:* Integer
)

Sender
the VPE object that fired the event

Action

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

ResultingAction
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Resulting Action	Value
PRINT_ACTION_OK	0
PRINT_ACTION_ABORT	1

Remarks:

Do not call CloseDoc() 193 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 Action = 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 314 at this stage, before the printer setup dialog will be shown to the user.

6.11 OnPrintNewPage Event

Is fired only while printing exactly before printing a new page. The event is useful to change the Device Control Properties 314 on-the-fly during printing. You may change all properties, **except the device itself**.

```
OnPrintNewPage(
        Sender: TVPEngine;
        Page: Integer;
        var ResultingAction: Integer
)
```

Sender
the VPE object that fired the event

Page
current page number that will be printed

ResultingAction
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Resulting Action	Value
PRINT_ACTION_OK	0
PRINT_ACTION_ CHANGE	1

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 OnRequestPrint Event - VCL 173: with Action = PRINT_MSG_ABORT or Action = PRINT_MSG_END).
- Changing the properties (like DevPaperBin 339, DevOrientation 318, DevPaperFormat 319, 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 333" 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:

OnPrintDevData Event 176

6.12 OnPrintDevData Event

Is sent only while printing, exactly before printing a new page and immediately after OnPrintNewPage() 174 has been sent. The only use for this event is to call DevSendData() 354 in response.

DevSendData() 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).

```
OnPrintDevData(
        Sender: TVPEngine;
        Page: Integer;
        var ResultingAction: Integer
)
```

Sender
the VPE object that fired the event

Page
current page number that will be printed

ResultingAction
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Resulting Action	Value
PRINT_ACTION_OK	0
PRINT_ACTION_ CHANGE	1

Remarks:

Your application should return **PRINT_ACTION_OK** (zero) if it processes this message without calling DevSendData().

If it called DevSendData(), your application must return **PRINT_ACTION_CHANGE** (= 1).

See also:

OnPrintNewPage() 174

6.13 OnBeforeMail Event

Is fired, 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.

OnBeforeMail(
    Sender: TVPEngine;
    var Cancel: Boolean
)

Sender
the VPE object that fired the event

Cancel
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
True	cancel operation i.e. do not mail the document
False	continue operation

If you set this parameter to True, the operation is cancelled, i.e. VPE will 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 592

6.14 OnAfterMail Event

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.

```
OnAfterMail(
        Sender: TVPEngine;
        Result: Integer
)
```

Sender
the VPE object that fired the event

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

See also:
E-Mail Functions 592

6.15 OnObjectClicked Event

[Professional Edition and above]

A clickable object 640 with an assigned ObjectID 642 has been clicked with the mouse.

This is one of the few events where CloseDoc() 193 may be called while processing it.

```
OnObjectClicked(
        Sender: TVPEngine;
        ObjectID: LongInt
)
```

Sender
the VPE object that fired the event

ObjectID
the Object ID of the object that was clicked by the user

See also:

Clickable objects 640

6.16 OnUDOPaint Event

[Professional Edition and above]

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

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

```
OnUDOPaint(
        Sender: TVPEngine
)
```

Sender
the VPE object that fired the event

See also:
User Defined Objects 646

6.17 OnAfterControlEnter

[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.

```
OnAfterControlEnter(
        Sender: TVPEngine;
        Obj: TVPEObject
)
```

Sender
  the VPE object that fired the event

Obj
  the object, which fired the event

See also:
  Interactive Documents 904

6.18 OnRequestControlExit

[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.

```
OnRequestControlExit(
        Sender: TVPEngine;
        Obj: TVPEObject;
        var CanExit: Boolean
)
```

Sender
the VPE object that fired the event

Obj
the object, which fired the event

CanExit
is a return-parameter, i.e. you can assign it one of the following values to control the resulting action of VPE:

Value	Description
True	yes, the control may loose the focus
False	no, the control may not loose the focus

See also:
Interactive Documents 904

6.19 OnAfterControlExit

[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() 916 or SetFocus() 213. 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.

OnAfterControlExit(
    Sender: TVPEngine;
    Obj: TVPEObject
)

Sender
    the VPE object that fired the event

Obj
    the object, which fired the event

See also:
    Interactive Documents 904

6.20 OnAfterControlChange

[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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Control*** by code.

```
OnAfterControlChange(
        Sender: TVPEngine;
        Obj: TVPEObject
)
```

Sender
  the VPE object that fired the event

Obj
  the object, which fired the event

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 185 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.

See also:
  Interactive Documents 904

6.21 OnAfterFieldChange

[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() 193 may be called while processing the event.
The event is not fired, if you set the value of a ***Field*** by code.

OnAfterFieldChange(
Sender: TVPEngine;
Obj: TVPEField
)

Sender
the VPE object that fired the event

Obj
the Field object 856, which fired the event

Remarks:
If you change the value of a Control by code, the OnAfterControlChange 184 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 OnAfterControlChange.

See also:
Interactive Documents 904

This page is intentionally left blank.

Management

7 Management

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

7.1 OpenDoc

Creates a new document with one initial blank page, reflecting the current state of all Design Time Properties. You always need to open the document before you may access any methods or runtime properties.

method void VPE.OpenDoc( )

Remarks:

In case of an Error, LastError 201 is set.

You may create an unlimited number of pages per document and an unlimited number of documents simultaneously, but both is limited by available memory. How much memory is needed, depends on the number of objects you insert and what type of objects you insert (a bitmap for example needs much more memory than a single line). So we can't give clear guidelines about memory usage. In case of doubt, use a monitoring tool to view how much memory is needed for your specific kind of document(s). For example, one page in the "Speed + Tables" demo needs about 10 KB of memory. This is really low, but 100 pages need about 1 MB of memory. If the memory usage is too high, we recommend to use File Swapping 195.

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 set AutoDelete 266 = false, the document is not closed when the preview is closed.

On non-Windows platforms you can call CloseDoc() 193 to remove a document from memory, or destroy the component itself.

To prevent the runtime error of opening a document that is already open, you have two possible options:

1. You check the property "IsOpen 192":

```
Private Sub ButtonReport_Click()
   If Report.IsOpen Then Exit Sub
   Call Report.OpenDoc
   Call Report.VpePrint(1, 1, "Hello World!")
   Call Report.Preview
End Sub
```

or,

2. You disable the control (button, menu-entry or whatsoever), which will cause opening the document, until the document is closed:

```
Private Sub ButtonReport_Click()
   ButtonReport.Enabled = False
   Call Report.OpenDoc
   Call Report.VpePrint(1, 1, "Hello World!")
   Call Report.Preview
End Sub
```

The closing of the document fires the event "DestroyWindow", so we enable the button here again:

```
Private Sub Report_DestroyWindow()
  ButtonReport.Enabled = True
End Sub
```

7.2 ParentWindow

[Python only]

If you are already using a window library, you can specify the parent window of the VPE preview using this property. It is of type HWND window handle from the Windows operating system.

property HWND VPE.ParentWindow

read / write; design & runtime, closed document required

Remarks:

In VPE for Python the HWND type is encapsulated in a *ctypes.c_void_p* type.

7.3 IsOpen

Returns the status of the document.

property boolean VPE.IsOpen

read; runtime only

Returns:

Value	Description
True	the document is open (OpenDoc has been called)
False	the document is not open

Remarks:

This property does not return the status of the Preview. To check if the Preview is open use the property IsPreviewVisible 212.

7.4 CloseDoc

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

method int VPE.CloseDoc( )

Returns:

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

Remarks:

You may call this method even if the document is already closed. In this case the method will return True.

Note: when a form which contains the VPE Control is being closed, it is **very important** that you call CloseDoc. You need to call CloseDoc exactly at the time when receiving the event which asks for confirmation if the form can be closed. (For C# and Visual Basic .NET it is the ***Form_Closing()*** event, for Visual Basic it is the ***Form_QueryUnload()*** event, for Delphi and C++ Builder it is the ***TForm.OnClose()*** event.)

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

Example for Visual Basic .NET:

```
Private Sub MenuClose_Click(ByVal sender As System.Object,
                            ByVal e As System.EventArgs)
                            Handles MenuClose.Click
   ' Do NOT call Application.Exit(), because Form1_Closing() will not be
   ' executed then!
   Close()
End Sub

Private Sub Form1_Closing(ByVal sender As Object,
ByVal e As System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
   If Not Usage.CloseDoc Or Not Enhanced.CloseDoc Then
         MsgBox("The application cannot terminate until all jobs have
                finished printing.")
         e.Cancel = True
   End If
End Sub
```

Example for C#:

```
private void menuClose_Click(object sender, System.EventArgs e)
{
   // Do NOT call Application.Exit(), because Form1_Closing() will not be
   // executed then!
   Close();
}

private void Form1_Closing(object sender,
                           System.ComponentModel.CancelEventArgs e)
{
   if (!Usage.CloseDoc() || !Enhanced.CloseDoc())
   {
      MessageBox.Show("The application cannot terminate until all
                       jobs have finished printing.");
      e.Cancel = true;
   }
}
```

Example for Visual Basic:

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
   If Not VPE1.CloseDoc Or Not VPE2.CloseDoc Then
      MsgBox ("The application cannot terminate until all jobs have
               finished printing.")
      Cancel = True
   End If
End Sub
```

Example for Delphi:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
   if not VPE1.CloseDoc or not VPE2.CloseDoc then
   begin
      ShowMessage('The application cannot terminate until all jobs
                   have finished printing.');
      Action := caNone;
   end;
end;
```

In the above examples the form contains two VPE Controls named "VPE1" and "VPE2". For both controls it is checked whether their previews can be closed or not.

7.5 SwapFileName

If this property is set, VPE will work with a Swap File when OpenDoc() 189 is called.

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 PageBreak() 363
- Moving to a different page by modifying the property CurrentPage 366

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

property string VPE.SwapFileName

read / write; design & runtime, closed document required

Possible Values:
swap file name (= document file name)

Default:
empty = not set

Remarks:
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.

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 LastError 201 after calling OpenDoc and each time after calling PageBreak 363.

When creating a new document with a swap file, compression 220 is always activated.

Example:

```
Doc.SwapFileName = "c:\reports\report.vpe"
Doc.OpenDoc
```

Instructs VPE to use the file "c:\reports\report.vpe" as Swap File.

```
Doc.SwapFileName = ""
```

Instructs VPE to use no Swap File (the default).

Example:

```
long count
Doc.SwapFileName = "c:\docs\report1.vpe"
Doc.OpenDoc
count = Doc.PageCount
Doc.PageBreak
Doc.Print(1, 1, "Added a new page.")
Doc.Preview
```

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 property DocFileReadOnly: 233

```
long count;
Doc.SwapFileName = "c:\docs\report1.vpe"
Doc.DocFileReadOnly = True
Doc.OpenDoc
if Doc.LastError <> 0 then
   exit function
end if
count = Doc.PageCount
Doc.VisualPage = count
Doc.Preview
```

If the VPE document file "c:\docs\report1.vpe" is not existing, the property LastError 201 will return VERR_FILE_OPEN. Otherwise the file will be opened in read-only mode 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. The preview will show the last page contained in the document.

When using the SwapFileName property, DevJobName 353 is set automatically to the file name.

See Also:

WriteDoc 225 and ReadDoc 229

7.6 EditProtection

[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 using SwapFileName 195 or if you call WriteDoc() 225 - the created VPE document file will be protected.

Interactive Edition only: if the EditProtection is enabled, the interactive objects 904 stored in VPE Document files are not editable within VPE, nor *VPEView*.

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

property EditProtection [integer] VPE.EditProtection

read / write; runtime only

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

.NET:
possible values are:

```
enum EditProtection
{
      Unprotected,
      Protected
}
```

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

Remarks:
Once you have assigned a value to this property, 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:

ActiveX / VCL:

```
VPE.EditProtection = 1
VPE.WriteDoc("my_file.vpe")
```

.NET:

```
VPE.EditProtection = EditProtection.Protected
VPE.WriteDoc("my_file.vpe")
```

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

7.7 License

[.NET, Java, PHP, Python, Ruby, VCL Only]

Licences VPE or an add-on module for the given document, so the demo banners disappear.

```
method void VPE.License(
        string Serial1,
        string Serial2
)
```

string Serial1, Serial2
the two serial strings provided by IDEAL Software when you acquire a license

Example:

If the license key has the following form:

VPE-A1234-123456
ABCD-EFGH

The method is called with:

```
VPE.License("VPE-A1234-123456", "ABCD-EFGH")
```

Remarks:

If you are using the VPE ActiveX, this method is not available.
The licensing for the ActiveX is done automatically by the COM Licensing Mechanism as defined by Microsoft. For a detailed explanation of the COM Licensing Mechanism, please see "Installing the VPE ActiveX - The Demo Banners Are Still Shown" in the Programmer's Manual.

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 199 = *true*.

7.8 EnableMultiThreading

[Professional Edition or higher]

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.

property boolean VPE.EnableMultiThreading

write; runtime

Possible Values:

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() 189, 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 allow to activate the thread-safe code for testing purposes.

The following does not apply to the .NET and ActiveX components, since both perform the licensing internally:
If you activated multi-threading and call later the License() 198 method, the licensing will fail. You must call the License() method **before** you activate the thread-safe code.

7.9 EnableURLs

[ActiveX only]

Instructs the ActiveX to redirect all file accesses to URL's, e.g. via the internet. This enables you do load documents, images, and Rich Text (RTF) from servers and internet sites. It is therefore ideal to use the ActiveX with **VBScript** or **JavaScript** within an Internet Browser. Even images INSIDE of a VPE document file - that reference a URL - are loaded via the network.

property boolean VPE.EnableURLs

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	Enabled
False	Disabled

Default:

False (do not interpret file names as URL's)

Example:

```
Doc.EnableURLs = True
Doc.ReadDoc("http://www.my-server.com/report-187-3.vpe")
Doc.Picture(1, 1, VFREE, VFREE, "ftp://ftp.my-server.com/image1.gif")
```

If the VPE ActiveX is embedded within a Browser-HTML page, you can also use relative paths. For example if the HTML file with the embedded VPE ActiveX was loaded from:

```
"http://www.my-server.com/usa/reports/active.html"
```

and the embedded VPE ActiveX shall load the image:

```
"http://www.my-server.com/usa/reports/images/img1.jpg"
```

the following call will do when **creating** the document itself:

```
VpeControl.Picture(1, 1, VFREE, VFREE, "images/img1.jpg")
```

To read a file from the local hard drive with EnableURLs = True, use the following:

```
Doc.ReadDoc("file://c:\docs\report.vpe")
Doc.Picture(1, 1, VFREE, VFREE, "file://c:\pictures\img1.jpg")
```

In the directory "Internet" - located in the VPE installation directory - is a complete HTML source code example. It demonstrates the use of VPE embedded within a Browser and JavaScript. The HTML source code example contains many helpful comments, which should be studied carefully.

7.10 LastError

Returns the error state of the last VPE function call.

property ErrorCode [long] VPE.LastError

read; runtime only

Possible Values:

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 property.

Error Codes:

Constant Name	Valu e	Enum	Description
VERR_OK	0	Ok	no error
VERR_COMMON	1	Common	common error
VERR_CANCELLED	2	Cancelled	the user cancelled an operation, for example the Open or Save file dialog
VERR_MEMORY	100	Memory	out of memory
VERR_FILE_OPEN	200	FileOpen	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	FileDocVersion	Document file has the wrong (higher) version and can not be opened / read
VERR_FILE_CREATE	202	FileCreate	error creating file; occurs when calling functions like WriteDoc(), OpenDoc() with SwapFileName set, WritePrinterSetup(), SetupPrinter() (only during write), etc.

			Meaning: illegal path or file name, file locked, disk full, no permissions
VERR_FILE_ACCESS	203	FileAccess	Access denied (no permission); see also DocFileReadOnly
VERR_FILE_READ	204	FileRead	error during file read operation
VERR_FILE_WRITE	205	FileWrite	error during file write operation
VERR_PRINT_SETUP_ABORT	225	PrintSetupAbort	Printer setup was aborted by user
VERR_PRINT_SETUP_INIT	226	PrintSetupInit	Printer setup failed: initialization of printer
VERR_PRINT_SYS	227	PrintSys	Failure in the Windows printing subsystem Possible causes might include a network printer that has been renamed, corrupt printer drivers, or corrupt print subsystem files. Solution or Workaround In an effort to solve the problem, use the steps below to remove and reinstall the printer. If the problem persists, it may suggest that there are further problems with the operating system or printer itself. Contact your system administrator for further assistance. - Go to Start > Settings > Printers. Remove the printer by selecting it and choose Delete from the File menu - Choose File > Server Properties to open up the Server Properties dialog. - Click on the Drivers tab. - Find the printer's name in the list and remove it. There may be multiple entries for the same printer; be sure to remove all instances of the printer driver. - Reinstall the printer normally by returning to the Start > Settings > Printers and clicking on click the Add Printer icon
VERR_PRINT_COMMON	228	PrintCommon	Common error during printing

VERR_PIC_IMPORT	300	PicImport	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	PicNoLicense	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	PicDXFCoord	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	PicExport	image could not be exported Meaning: File could not be created, or not enough memory
VERR_MOD_GRAPH_IMP	400	ModGraphImp	Error loading Graphics Import Library
VERR_MOD_GRAPH_PROC	401	ModGraphProc	Error loading Graphics Processing Library
VERR_MOD_BARCODE	402	ModBarcode	Error loading Barcode Library
VERR_MOD_CHART	403	ModChart	Error loading Chart Library
VERR_MOD_ZLIB	404	ModZlib	Error loading ZLIB Library
VERR_MOD_VPDF	405	ModPDF	Error loading PDF Export Library
VERR_MOD_VBAR2D	406	ModBarcode2D	Error loading 2D Barcode Library
VERR_MAIL_LOAD_MAPI	450	MailLoadMapi	Could not load MAPI
VERR_MAIL_CREATE	451	MailCreate	Could not create temporary file
VERR_MAIL_USER_ABORT	452	MailUserAbort	
VERR_MAIL_FAILURE	453	MailFailure	
VERR_MAIL_LOGON_FAILURE	454	MailLogonFailure	
VERR_MAIL_DISK_FULL	455	MailDiskFull	
VERR_MAIL_INSUFFICIENT_MEMORY	456	MailInsufficientMemory	

VERR_MAIL_ACCESS_DENIED	457	MailAccessDenied	
VERR_MAIL_RESERVED	458	MailReserved	
VERR_MAIL_TOO_MANY_SESSIONS	459	MailTooManySessions	
VERR_MAIL_TOO_MANY_FILES	460	MailTooManyFiles	
VERR_MAIL_TOO_MANY_RECIPIENTS	461	MailTooManyRecipients	
VERR_MAIL_ATTACHMENT_NOT_FOUND	462	MailAttachmentNotFound	
VERR_MAIL_ATTACHMENT_OPEN_FAILURE	463	MailAttachmentOpenFailure	
VERR_MAIL_ATTACHMENT_WRITE_FAILURE	464	MailAttachmentWriteFailure	
VERR_MAIL_UNKNOWN_RECIPIENT	465	MailUnknownRecipient	
VERR_MAIL_BAD_RECIPTYPE	466	MailBadRecipType	
VERR_MAIL_NO_MESSAGES	467	MailNoMessages	
VERR_MAIL_INVALID_MESSAGE	468	MailInvalidMessage	
VERR_MAIL_TEXT_TOO_LARGE	469	MailTextTooLarge	
VERR_MAIL_INVALID_SESSION	470	MailInvalidSession	
VERR_MAIL_TYPE_NOT_SUPPORTED	471	MailTypeNotSupported	
VERR_MAIL_AMBIGUOUS_RECIPIENT	472	MailAmbiguousRecipient	
VERR_MAIL_MESSAGE_IN_USE	473	MailMessageInUse	
VERR_MAIL_NETWORK_FAILURE	474	MailNetworkFailure	
VERR_MAIL_INVALID_EDITFIELDS	475	MailInvalidEditFields	
VERR_MAIL_INVALID_RECIPS	476	MailInvalidRecips	
VERR_MAIL_NOT_SUPPORTED	477	MailNotSupported	
VERR_ZLIB_STREAM	500	ZlibStream	Stream Inconsistent
VERR_ZLIB_DATA	501	ZlibData	Data Corrupt
VERR_ZLIB_BUFFER	502	ZlibBuffer	Internal Buffer Error
VERR_ZLIB_VERSION	503	ZlibVersion	Wrong Version of ZLIB Library
VERR_VBAR2D_FORMAT_OUT_OF_RANGE	550	Vbar2DFormatOutOfRange	
VERR_VBAR2D_UNDEFINED_ID	551	Vbar2DUndefinedId	
VERR_VBAR2D_FORMAT_TOO_LONG	552	Vbar2DFormatTooLong	
VERR_VBAR2D_FORMAT_OUT_OF_MEMORY	553	Vbar2DFormatOutOfMemory	
VERR_VBAR2D_FORMAT_DATA_INVALID	554	Vbar2DFormatDataInvalid	
VERR_VBAR2D_FORMAT_NOT_ALLOWED	555	Vbar2DFormatNotAllowed	
VERR_VBAR2D_DATA_WRONG_LENGTH	556	Vbar2DDataWrongLength	
VERR_VBAR2D_DATA_ZERO_LENGTH	557	Vbar2DDataZeroLength	
VERR_VBAR2D_DATA_TOO_SHORT	558	Vbar2DDataTooShort	
VERR_VBAR2D_DATA_TOO_LONG	559	Vbar2DDataTooLong	
VERR_VBAR2D_INVALID_DATA	560	Vbar2DInvalidData	
VERR_VBAR2D_SQUARE_EDGE_TOO_SMALL	561	Vbar2DSquareEdgeTooSmall	
VERR_VBAR2D_SQUARE_TOO_LARGE	562	Vbar2DSquareTooLarge	
VERR_VBAR2D_EDGE_OVER_FORCED	563	Vbar2DEdgeOverForced	

VERR_VBAR2D_SQUARE_ASPECT_SMALL	564	Vbar2DSquareAspectSmall	
VERR_VBAR2D_SQUARE_ASPECT_LARGE	565	Vbar2DSquareAspectLarge	
VERR_VBAR2D_SQUARE_EVEN_ODD_MATCH	566	Vbar2DSquareEvenOddMatch	
VERR_VBAR2D_INVALID_EDGE	567	Vbar2DInvalidEdge	
VERR_VBAR2D_SQUARE_EDGE_TOO_LARGE	568	Vbar2DSquareEdgeTooLarge	
VERR_VBAR2D_INVALID_ECC	569	Vbar2DInvalidEcc	
VERR_VBAR2D_INVALID_BORDER	570	Vbar2DInvalidBorder	
VERR_VBAR2D_SELF_TEST_FAILED	571	Vbar2DSelfTestFailed	
VERR_RTF_BRACES	1000	RtfBraces	RTF: unbalanced braces "{}"
VERR_RTF_OVERFLOW	1001	RtfOverflow	RTF: only 16-bit version; generated internal structure > 64 KB
VERR_RTF_FONTTBL	1002	RtfFontTable	RTF: error parsing font table
VERR_RTF_COLORTBL	1003	RtfColorTable	RTF: error parsing color table
VERR_TPL_OWNERSHIP	2000	TplOwnership	Template: tried to dump the template to a foreign VPE document (where the template was not loaded into).
VERR_TPL_PAGE_ALREADY_DUMPED	2001	TplPageAlreadyDumped	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	TplAuthentication	Template: the template has an authentication key and it has not been validated successfully

7.11 Preview

[GUI Control Only]

Opens the preview window. The preview shows the page specified by the property VisualPage 216 (by default, this is the first page). You can embed the preview into your own window / form, see the property External Window 234.

```
method void VPE.Preview(
)
```

See also:

PreviewDoc 207

7.12 PreviewDoc

[GUI Control Only]

The same as Preview() 206. Opens the preview window on a defined position. The preview shows the page specified by VisualPage 216 (by default, this is the first page). You can embed the preview into your own window / form, see the property External Window 234.

method void VPE.PreviewDoc(
long *Left*,
long *Top*,
long *Right*,
long *Bottom*,
integer *ShowHide*
)

long Left, Top, Right, Bottom
position of the preview window in pixels, for "Left" = -1 all coordinates are ignored.

integer ShowHide
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

7.13 CenterPreview

[GUI Control Only]

The preview-window is centered in the middle of the parent window (= parent form) or desktop.

```
method void VPE.CenterPreview(
        long Width,
        long Height,
        boolean OnScreen
)
```

long 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)

boolean OnScreen

Value	Description
True	the preview is centered on the desktop (screen)
False	the preview is centered relative to the parent window

Remarks:

Java / Python: The parameter *OnScreen* has no effect. The preview is always centered on the desktop.

7.14 BringPreviewToTop

[GUI Control Only]

Brings the VPE preview-window to the foreground and sets the focus to it. This only works, if the preview window is not embedded, and if another window of the calling process has currently the focus.

method void VPE.BringPreviewToTop()

7.15 PreviewCtrl

[GUI Control Only]

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

property PreviewCtrl [integer] VPE.PreviewCtrl

read / write; design & runtime (read access is only possible, if the document is closed)

Possible Values:

ActiveX / VCL	Value	Enum	Comment
PREVIEW_STAY	0	Stay	vertical position unaffected
PREVIEW_JUMPTOP	1	JumpTop	vertical position on top of page (default)

Default:

PREVIEW_JUMPTOP

Example:

ActiveX / VCL:

```
Doc.PreviewCtrl = PREVIEW_STAY
```

.NET:

```
Doc.PreviewCtrl = PreviewCtrl.Stay
```

7.16 ClosePreview

[GUI Control Only]

Closes the preview. The document is **NOT** closed (= removed from memory), regardless of the setting of AutoDelete 266. So you can re-open the preview with the method Preview() 206 after a call to this method.

```
method void VPE.ClosePreview(
)
```

7.17 IsPreviewVisible

[GUI Control Only]

Returns the visibility state of the Preview.

property boolean VPE.IsPreviewVisible

read; runtime only

Returns:

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

7.18 SetFocus

[ActiveX / VCL only – see also the method Focus 215]

If the Preview is not embedded, a call to this method sets the focus to the Preview.

```
method void VPE.SetFocus(
)
```

Remarks:

For Visual Basic use the method VpeSetFocus 214, because Visual Basic does not understand that a method of an ActiveX control is called and mismatches it with a call to the VB internal method.

7.19 VpeSetFocus

[ActiveX only]

If the Preview is not embedded, a call to this method sets the focus to the Preview.

```
method void VPE.VpeSetFocus(
)
```

7.20 Focus

[GUI Control Only, .NET and Java only]

Sets input focus to the control. If the Preview is not embedded, the external preview window will receive the focus.

```
method bool VPE.Focus(
)
```

Returns:

true if the input focus request was successful; otherwise **false**.

7.21 VisualPage

[GUI Control Only]

Retrieve or set 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 CurrentPage 366).

If you set this property, the preview display 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 / retrieve the position of the visual page.

property long VPE.VisualPage

read / write; runtime only

Possible Values:
The number of the currently visible page.

Default:
1 = the first page, after a call to OpenDoc() 189

Remarks:
In case of an error, LastError 201 is set.

If 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 = Doc.VisualPage

// move the preview to the next page:
Doc.VisualPage = n + 1
```

7.22 DispatchAllMessages

[GUI Control 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.

```
method boolean VPE.DispatchAllMessages(
)
```

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 DispatchAllMessages() while processing a VPE Event!

7.23 Refresh

[GUI Control 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.

```
method void VPE.Refresh(
)
```

7.24 DocExportType

The value of this property controls, what type of document is created by the method WriteDoc() 225.

property DocExportType [integer] VPE.DocExportType

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VPE_DOC_TYPE_AUTO	0	Auto	WriteDoc() will create VPE, PDF, HTML, XML or ODT files, depending on the file suffix of the supplied file name
VPE_DOC_TYPE_VPE	1	VPE	WriteDoc() will create VPE files, regardless of the file suffix
VPE_DOC_TYPE_PDF	2	PDF	WriteDoc() will create PDF files, regardless of the file suffix
VPE_DOC_TYPE_HTML	3	HTML	WriteDoc() will create HTML files, regardless of the file suffix
VPE_DOC_TYPE_XML	4	XML	WriteDoc() will create XML files, regardless of the file suffix
VPE_DOC_TYPE_ODT	5	ODT	WriteDoc() will create ODT files, regardless of the file suffix

Default:
VPE_DOC_TYPE_AUTO

7.25 Compression

[Not supported by the Community Edition]

Returns / sets the current mode for file compression. This affects writing native VPE document files 225 as well as exporting PDF files.

property Compression [long] VPE.Compression

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
DOC_COMPRESS_NONE	0	None	no compression
DOC_COMPRESS_FLATE	1	Flate	flate (ZLIB) compression

Default:

DOC_COMPRESS_FLATE

Remarks:

When creating a VPE document file using SwapFileName 195, 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.

7.26 OpenFileDialog

[GUI Control Only, not supported by the Community Edition]

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

method void VPE.OpenFileDialog(
)

Remarks:

You can preset a file name for the file-open dialog with the property OpenFileName 223.

7.27 SaveFileDialog

[GUI Control 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.

```
method void VPE.SaveFileDialog(
)
```

Remarks:

You can preset a file name for the file-open dialog with the property SaveFileName 224.

7.28 OpenFileName

[GUI Control Only, not supported by the Community Edition]

Returns / sets the default path and file name, which will be shown in the Open File Dialog 221.

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

property string VPE.OpenFileName

read / write; runtime only

Possible Values:
the file name

Default:
empty string

7.29 SaveFileName

[GUI Control Only, not supported by the Community Edition]

Returns / sets the default path and file name, which will be shown in the Save File Dialog 222.

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

property string VPE.SaveFileName

read / write; runtime only

Possible Values:
the file name

Default:
empty string

7.30 WriteDoc

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)
- .xml – XML file format (requires Professional Edition or higher)
- .odt – Open Document Text 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 HTML files, please see the "Programmer's Manual", chapter "The HTML Export Module".

```
method boolean VPE.WriteDoc(
        string FileName
)
```

string FileName
the path and filename

Returns:

Value	Description
True	Success
False	Failure

Remarks:

In case of an error, LastError 201 is set.

Depending on the setting of the property DocExportType 219, 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 220 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*.
VPE includes VPE View (see "VPE View - the Document Viewer" in the Programmer's Manual) a royalty-free Document Viewer for VPE Documents.
If you want to protect your files from being edited, use the property EditProtection 197.

7.31 WriteDocPageRange

[Professional Edition and above]

Identical to WriteDoc() 225, but writes only the given range of pages to a document file.

```
method boolean VPE.WriteDocPageRange(
        string FileName,
        long FromPage,
        long ToPage
)
```

string FileName
the path and filename

long FromPage
starting page of the page range

long ToPage
ending page of the page range

Returns:

Value	Description
True	Success
False	Failure

7.32 WriteDocStream

[Professional Edition and above]

Identical to WriteDoc() 225, but writes the document to a stream. Currently the only type of stream offered by VPE is a memory stream.

```
method boolean VPE.WriteDocStream(
        TVPEStream stream
)
```

TVPEStream stream
The stream-object where the document is written to. The stream must have been created before by calling CreateMemoryStream() 694.

Returns:

Value	Description
True	Success
False	Failure

7.33 WriteDocStreamPageRange

[Professional Edition and above]

Identical to WriteDocStream() 227, but writes only the given range of pages to a stream.

```
method boolean VPE.WriteDocStreamPageRange(
        TVPEStream stream,
        long FromPage,
        long ToPage
)
```

TVPEStream stream
The stream-object where the document is written to. The stream must have been created before by calling CreateMemoryStream() 694.

long FromPage
starting page of the page range

long ToPage
ending page of the page range

Returns:

Value	Description
True	Success
False	Failure

7.34 ReadDoc

Reads a VPE or VPE-XML document from the file named <FileName> and appends it to the current document (this even works, if the current document was opened using a Swap File 195, e.g. it is itself a file!)

The document file may have been created before with OpenDoc() 189 using a Swap File or WriteDoc() 225.

```
method boolean VPE.ReadDoc(
        string FileName
)
```

string FileName
the path and filename

Returns:

Value	Description
True	Success
False	Failure

Remarks:

In case of an error, LastError 201 is set.

Tip: If you want to preview or print an already created VPE-Document file, DON'T USE ReadDoc(). Use OpenDoc() 189 with a Swap File 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 and VPE-XML document files.

7.35 ReadDocPageRange

[Professional Edition and above]

Identical to ReadDoc() 229, but reads only the given range of pages from a document file.

```
method boolean VPE.ReadDocPageRange(
        string FileName,
        long FromPage,
        long ToPage
)
```

string FileName
the path and filename

long FromPage
starting page of the page range

long ToPage
ending page of the page range

Returns:

Value	Description
True	Success
False	Failure

Remarks:
In case of an error, LastError 201 is set.

7.36 ReadDocStream

[Professional Edition and above]

Identical to ReadDoc() 229, but reads the document from a stream. Currently the only type of stream offered by VPE is a memory stream.

method boolean VPE.ReadDocStream(
    TVPEStream *stream*
)

TVPEStream stream
    The stream-object where the document is written to. The stream must have been created before by calling CreateMemoryStream() 694 and of course it must hold a valid VPE document.

Returns:

Value	Description
True	Success
False	Failure

7.37 ReadDocStreamPageRange

[Professional Edition and above]

Identical to ReadDocStream() 231, but reads only the given range of pages from a stream.

```
method boolean VPE.ReadDocStreamPageRange(
        TVPEStream stream,
        long  FromPage,
        long ToPage
)
```

TVPEStream stream
The stream-object where the document is written to. The stream must have been created before by calling CreateMemoryStream() 694 and of course it must hold a valid VPE document.

long FromPage
starting page of the page range

long ToPage
ending page of the page range

Returns:

Value	Description
True	Success
False	Failure

7.38 DocFileReadOnly

If you use set this property to True, a 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.

If you read an existing document file with ReadDoc() 229 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.

property boolean VPE.DocFileReadOnly

read / write; design & runtime (read access is only possible, if the document is closed)

Possible Values:

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

Default:
False

7.39 ExternalWindow

[GUI Control Only, not supported by the Community Edition]

Controls, whether the VPE Preview Window is embedded in the parent form it is contained in, or if it is an external window managed by VPE itself.

property boolean VPE.ExternalWindow

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	Preview Window is external
False	Preview Window is embedded

Default:

True

Remarks:

ActiveX: If ExternalWindow is True, it might be necessary to set the Property "Visible" to False, this depends on the container application.

Java: The preview can not be embedded, it is only available as external window.

For the Community Edition the preview can not be embedded, it is only available as external window.

7.40 BorderStyle

[.NET only, GUI Control Only]

If embedded 234, the style of the border drawn around the control. Use 'None', if fully docked.

property BorderStyle VPE.BorderStyle

read / write; design- and runtime

Possible Values:

Value	Description
Fixed3D	A three-dimensional border
FixedSingle	A single-line border
None	No border

Default:
None

7.41 Caption

[ActiveX / VCL only, see also the property Text 893]

If the preview window is external 234, this will be its caption (i.e. the title of the Preview Window). The caption is also used by VPE, to compose the default JobName 353 of the print job.

property string VPE.Caption

read / write; design & runtime, closed document required

Default:
"VPE Preview"

Example:

```
Doc.Caption = "Monthly Report"
```

7.42 Text

[.NET, Java, PHP, Python, Ruby Only, GUI Control Only]

If the preview window is external, this will be its caption 236 (i.e. the title of the Preview Window). The caption is also used by VPE, to compose the default JobName 353 of the print job.

property string VPE.Text

read / write; design & runtime, closed document required

Default:
the default caption is generated by the designer, e.g. Visual Studio .Net

Example:

```
Doc.Text = "Monthly Report"
```

7.43 ToolBar

[GUI Control Only, not supported by the Community Edition]

Controls, whether the ToolBar is shown or not.

property boolean VPE.ToolBar

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the ToolBar
False	do not show the ToolBar

Default:

True

7.44 tbOpen

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Open-File Button in the ToolBar is shown or not.

property boolean VPE.tbOpen

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the Open-File Button
False	do not show the Open-File Button

Default:

True

7.45 tbSave

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Save-File Button in the ToolBar is shown or not.

property boolean VPE.tbSave

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the Save-File Button
False	do not show the Save-File Button

Default:

True

7.46 tbPrint

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Print Button in the ToolBar is shown or not.

property boolean VPE.tbPrint

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the Print Button
False	do not show the Print Button

Default:
True

7.47 tbPrintSetup

[GUI Control 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.

property boolean VPE.tbPrintSetup

read / write; design & runtime (read access is only possible, if the document is closed)

Possible Values:

Value	Description
True	Enabled
False	Disabled

Default:

enabled (Print Setup Dialog will appear)
For the VpeWebControl, the default is disabled, so no Print Setup Dialog will appear. It is not very meaningful that printer setup dialogs are shown on a running server.

7.48 tbMail

[GUI Control Only, not supported by the Community Edition]

Controls, whether the E-Mail Button in the ToolBar is shown or not.

property boolean VPE.tbMail

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the E-Mail Button
False	do not show the E-Mail Button

Default:

True

7.49 EnableMailButton

[GUI Control Only, not supported by the Community Edition]

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

property boolean VPE.EnableMailButton

write; runtime

Possible Values:

Value	Description
True	Enabled
False	Disabled

Default:

enabled

7.50 tbScale

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Scale Button Group in the ToolBar is shown or not.

property boolean VPE.tbScale

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the Scale Button Group
False	do not show the Scale Button Group

Default:

True

7.51 tbGrid

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Grid Button in the ToolBar is shown or not.

property boolean VPE.tbGrid

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the Grid Button
False	do not show the Grid Button

Default:

False

7.52 GridMode

[GUI Control 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.

property GridMode [integer] VPE.GridMode

read / write; design- & runtime (read access is only possible, if the document is closed)

Possible Values:

ActiveX / VCL	Enum	Comment
True	InForeground	draw the grid in the foreground
False	InBackground	draw the grid in the background

Default:
True (InForeground)

7.53 GridVisible

[GUI Control Only, not supported by the Community Edition]

Specifies, if the Layout Grid is drawn.

property boolean VPE.GridVisible

read / write; design- & runtime (read access is only possible, if the document is closed)

Possible Values:

Value	Description
True	is drawn
False	is not drawn

Default:

False

7.54 tbNavigate

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Navigation Button Group in the ToolBar is shown or not.

property boolean VPE.tbNavigate

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the Navigation Button Group
False	do not show the Navigation Button Group

Default:
True

7.55 tbHelp

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Help-Button in the ToolBar is shown or not.

property boolean VPE.tbHelp

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the Help Button
False	do not show the Help Button

Default:
True

7.56 RouteHelp

[GUI Control 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 DoHelp() 144 (VCL: OnHelp() 170, .NET: HelpRequested() 62) 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.

property boolean VPE.RouteHelp

read / write; design- & runtime (read access is only possible, if the document is closed)

Possible Values:

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

Default:

False

7.57 tbAbout

[GUI Control Only, not supported by the Community Edition]

Controls, whether the About Button in the ToolBar is shown or not.

property boolean VPE.tbAbout

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the About Button
False	not show the About Button

Default:

True

7.58 tbClose

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Close Button in the ToolBar is shown or not.

If the preview is external (ExternalWindow 234 = True), the close controls of the window (including the system-menu) are also disabled.

property boolean VPE.tbClose

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the Close Button
False	not show the Close Button

Default:

True

7.59 EnableCloseButton

[GUI Control 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 external (ExternalWindow 234 = True), the close controls of the window (including the system-menu) are also disabled.

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

property boolean VPE.EnableCloseButton

write; runtime

Possible Values:

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 tbClose 253).

7.60 StatusBar

[GUI Control Only, not supported by the Community Edition]

Controls, whether the StatusBar is shown or not.

property boolean VPE.StatusBar

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the StatusBar
False	do not show the StatusBar

Default:

True

7.61 PageScroller

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Page Scroller in the Status Bar is shown or not.

property boolean VPE.PageScroller

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the Page Scroller
False	do not show the Page Scroller

Default:

True

7.62 PageScrollerTracking

[GUI Control Only, not supported by the Community Edition]

When activated (set to true), moving the page scroller in the statusbar 255 will immediately update the preview. Otherwise the preview will be updated if the scrolling has finished.

property boolean VPE.PageScrollerTracking

read / write; design- & runtime (read access is only possible, if the document is closed)

Possible Values:

Value	Description
True	enabled
False	disabled

Default:

True

7.63 StatusSegment

[GUI Control Only, not supported by the Community Edition]

Controls, whether the Status Segment (where you can show messages and the Progress Bars) in the Status Bar is shown or not.

property boolean VPE.StatusSegment

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	show the StatusSegment
False	do not show the StatusSegment

Default:
True

7.64 StatusText

[GUI Control Only, not supported by the Community Edition]

Writes the given text into the Statusbar 255. If text is empty (i.e. ""), the default "Ready" in the current GUI-Language will be displayed.

property string VPE.StatusText

write; runtime only

Possible Values:

The text that shall be written into the Statusbar.

Remarks:

The Progress Bar 260 overlays the text segment in the Statusbar, therefore text will not be visible until the Progress Bar is closed.

7.65 OpenProgressBar

[GUI Control Only, not supported by the Community Edition]

Creates a Progress Bar in the Statusbar 255 with the initial value of zero (0%).

method void VPE.OpenProgressBar(
)

Remarks:

The Progress Bar hides the text segment in the Statusbar, therefore text displayed there with StatusText 259 will not be visible until the Progress Bar is closed.

Example:

```
Doc.OpenProgressBar
for n = 0 to 100
   Doc.StatusProgress = n
next n
Doc.CloseProgressBar
```

7.66 StatusProgress

[GUI Control Only, not supported by the Community Edition]

Sets the Progress Bar to the given percentage value.

property integer VPE.StatusProgress

write; runtime only

Possible Values:

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

Example:

```
Doc.OpenProgressBar
for n = 0 to 100
  Doc.StatusProgress = n
next n
Doc.CloseProgressBar
```

7.67 CloseProgressBar

[GUI Control Only, not supported by the Community Edition]

Closes the Progress Bar.

method void VPE.CloseProgressBar(
)

Example:

```
Doc.OpenProgressBar
for n = 0 to 100
  Doc.StatusProgress = n
next n
Doc.CloseProgressBar
```

7.68 BusyProgressBar

[Enhanced Edition and above; GUI Control Only]

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 255.

If you are also showing the private Progress Bar (see OpenProgressBar 260), **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.

property boolean VPE.BusyProgressBar

write; runtime only

Possible Values:

Value	Description
True	Enabled
False	Disabled

Default:
True

7.69 Rulers

[GUI Control Only, not supported by the Community Edition]

Controls, whether rulers are shown in the VPE Preview Window or not.

property boolean VPE.Rulers

read / write; design & runtime, closed document required

Possible Values:

Value	Description
True	Rulers are shown
False	Rulers are not shown

Default:

True

7.70 RulersMeasure

[GUI Control 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 UnitTransformation 361.

property RulersMeasure [long] VPE.RulersMeasure

read / write; design- & runtime

Possible Values:

ActiveX / VCL	Value	Enum	Comment
CM	0	Centimeter	rulers are shown in metric units
INCH	1	Inch	rulers are shown in inch units

Default:

CM (Centimeter)

7.71 AutoDelete

[GUI Control Only, not supported by the Community Edition]

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

property boolean VPE.AutoDelete

write; runtime

Possible Values:

Value	Description
True	yes, close the document also
False	No

Default:

True (= the document is deleted if the user closes the preview)

7.72 PreviewWithScrollers

[GUI Control 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.

property boolean VPE.PreviewWithScrollers

write; runtime only

Possible Values:

Value	Description
True	scrollbars are visible
False	scrollbars are invisible

Default:

True

7.73 PaperView

[GUI Control 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.

property boolean VPE.PaperView

read / write; design- & runtime (read access is only possible, if the document is closed)

Possible Values:

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

Default:
True

7.74 Scale

[GUI Control Only, not .NET (see PreviewScale 270) not supported by the Community Edition]

Returns / sets the scale of the preview (not for printing - see PrintScale 310).

For .NET Controls the name of this property is reserved.

property double VPE.Scale

read / write; runtime

Possible Values:
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.

Remarks:
Not available for .NET, see PreviewScale 270
For Visual Basic 6 and earlier versions, see VpeScale 271

See also:
PreviewScale 270
VpeScale 271

7.75 PreviewScale

[GUI Control Only; not supported by the Community Edition]

The same as Scale 269. Returns / sets the scale of the preview (not for printing - see PrintScale 310).

property double VPE.PreviewScale

read / write; runtime

Possible Values:

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.

See also:

VpeScale 271

7.76 VpeScale

[ActiveX only; not supported the Community Edition]

The same as Scale 269. This method is for use with Visual Basic. Visual Basic has problems with the keywords "Print, Write, Line and Scale". VB doesn't recognize, that these are methods and properties of an ActiveX.

Returns / sets the scale of the preview (not for printing - see PrintScale 310).

property double VPE.VpeScale

read / write; runtime

Possible Values:

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.

7.77 ScalePercent

[GUI Control Only, not supported by the Community Edition]

Returns / sets the scale for the preview in percent (not for printing - see PrintScale 310).

property integer VPE.ScalePercent

read / write; runtime

Possible Values:

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.

7.78 MinScale

[GUI Control 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 %).

property double VPE.MinScale

write; runtime only

Possible Values:
for example: 1.0 = 1:1   0.25 = 1:4

Default:
0.25

7.79 MinScalePercent

[GUI Control 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 %)

property integer VPE.MinScalePercent

write; runtime only

Possible Values:

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

Default:

25

7.80 MaxScale

[GUI Control 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 %).

property double VPE.MaxScale

write; runtime only

Possible Values:

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

Default:

32.0 (3200 %)

7.81 MaxScalePercent

[GUI Control 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 %).

property integer VPE.MaxScalePercent

write; runtime only

Possible Values:

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

Default:

3200 (3200 %)

7.82 ScaleMode

[GUI Control Only, not supported by the Community Edition]

Sets or gets the scale mode of the preview.

property ScaleMode [int] VPE.ScaleMode

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VSCALE_MODE_FULL_PAGE	0	FullPage	show full page
VSCALE_MODE_PAGE_WIDTH	1	PageWidth	fit page width
VSCALE_MODE_ZOOM_TOOL	2	ZoomTool	zoom tool activated
VSCALE_MODE_FREE	3	Free	any scale allowed

Default:

VSCALE_MODE_PAGE_WIDTH

7.83 ZoomPreview

[GUI Control Only, not supported by the Community Edition]

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

```
method void VPE.ZoomPreview(
        long Left,
        long Top,
        long Right,
        long Bottom
)
```

long 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.

7.84 ZoomIn

[GUI Control Only, not supported by the Community Edition]

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

```
method void VPE.ZoomIn(
)
```

Remarks:

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

7.85 ZoomOut

[GUI Control Only, not supported by the Community Edition]

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

```
method void VPE.ZoomOut(
)
```

Remarks:

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

7.86 SetPreviewPosition

[GUI Control Only, not supported by the Community Edition]

Scrolls the preview into the given position.

```
method void VPE.SetPreviewPosition(
        long Left,
        long Top
)
```

long 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.

7.87 DefineKey

[GUI Control Only, not supported by the Community Edition]

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

method void VPE.DefineKey(
    KeyFunction [long] *Function*,
    Keys [long] *KeyCode*,
    Keys [long] *AddKeyCode1*,
    Keys [long] *AddKeyCode2*
)

KeyFunction [long] Function
All available keyboard functions are enumerated in the VKEY_xyz constants (see below).

Keys [long] KeyCode, AddKeyCode1, AddKeyCode2
Virtual keycode constants (defined in the Windows SDK) for the keys that need to be pressed at once for the specified function. Add-on keys like Ctrl, Alt and Shift may not be used in the parameter *KeyCode*, use these key codes in the parameters *AddKeyCode1* or *AddKeyCode2*.

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

Example:

Active X / VCL:
```
Doc.DefineKey(VKEY_CLOSE, VK_ESCAPE, 0, 0)
```
Defines the key <ESC> to close the preview when pressed.

```
Doc.DefineKey(VKEY_CLOSE, VK_ESCAPE, VK_SHIFT, 0)
```
Defines the key pair <SHIFT> + <ESC> to close the preview when pressed.

```
Doc.DefineKey(VKEY_PRINT, 0, 0, 0)
```
There is no key associated with the keyboard function VKEY_PRINT.

.NET:
```
Doc.DefineKey(KeyFunction.Close, Keys.Escape, 0, 0)
```
Defines the key <ESC> to close the preview when pressed.

```
Doc.DefineKey(KeyFunction.Close, Keys.Escape, Keys.ShiftKey, 0)
```
Defines the key pair <SHIFT> + <ESC> to close the preview when pressed.

```
Doc.DefineKey(KeyFunction.Print, 0, 0, 0)
```
There is no key associated with the keyboard function "Print".

Possible values for the parameter "Function" are:

Constant	Value	Enum	Default Key(s)
VKEY_SCROLL_LEFT	0	ScrollLeft	Arrow Left
VKEY_SCROLL_PAGE_LEFT	1	ScrollPageLeft	Ctrl-Arrow Left
VKEY_SCROLL_RIGHT	2	ScrollRight	Arrow Right
VKEY_SCROLL_PAGE_RIGHT	3	ScrollPageRight	Ctrl-Arrow Right
VKEY_SCROLL_UP	4	ScrollUp	Arrow Up
VKEY_SCROLL_PAGE_UP	5	ScrollPageUp	Ctrl-Arrow Up
VKEY_SCROLL_DOWN	6	ScrollDown	Arrow Down
VKEY_SCROLL_PAGE_DOWN	7	ScrollPageDown	Ctrl-Arrow Down
VKEY_SCROLL_TOP	8	ScrollTop	Pos1
VKEY_SCROLL_BOTTOM	9	ScrollBottom	End
VKEY_PRINT	10	Print	F2
VKEY_MAIL	11	Mail	F3
VKEY_FULL_PAGE	12	FullPage	Ctrl-Ins
VKEY_PAGE_WIDTH	13	PageWidth	Ctrl-Del
VKEY_ZOOM_IN	14	ZoomIn	Ins
VKEY_ZOOM_OUT	15	ZoomOut	Del
VKEY_GRID	16	Grid	"G"
VKEY_PAGE_FIRST	17	PageFirst	Ctrl-Page Up
VKEY_PAGE_LEFT	18	PageLeft	Page Up
VKEY_PAGE_RIGHT	19	PageRight	Page Down
VKEY_PAGE_LAST	20	PageLast	Ctrl-Page Down
VKEY_HELP	21	Help	F1
VKEY_INFO	22	Info	"I"
VKEY_CLOSE	23	Close	<not defined>
VKEY_GOTO_PAGE	24	GotoPage	ENTER
VKEY_OPEN	25	Open	Ctrl-O
VKEY_SAVE	26	Save	Ctrl-S
VKEY_ESCAPE	27	Escape	Esc

See also:

SendKey 284

7.88 SendKey

[GUI Control Only, not supported by the Community Edition]

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

method void VPE.SendKey(
    KeyFunction [long] *Vkey*
)

KeyFunction [long] VKey
    All available keyboard functions are enumerated in the VKEY_xyz constants (see below).

Example:

ActiveX / VCL:
```
Doc.SendKey(VKEY_SCROLL_DOWN)
```

.NET:
```
Doc.SendKey(KeyFunction.ScrollDown)
```
Scrolls the preview down by one cm.

Possible values for the parameter "VKey" are:

Constant	Value	Enum
VKEY_SCROLL_LEFT	0	ScrollLeft
VKEY_SCROLL_PAGE_LEFT	1	ScrollPageLeft
VKEY_SCROLL_RIGHT	2	ScrollRight
VKEY_SCROLL_PAGE_RIGHT	3	ScrollPageRight
VKEY_SCROLL_UP	4	ScrollUp
VKEY_SCROLL_PAGE_UP	5	ScrollPageUp
VKEY_SCROLL_DOWN	6	ScrollDown
VKEY_SCROLL_PAGE_DOWN	7	ScrollPageDown
VKEY_SCROLL_TOP	8	ScrollTop
VKEY_SCROLL_BOTTOM	9	ScrollBottom
VKEY_PRINT	10	Print
VKEY_MAIL	11	Mail
VKEY_FULL_PAGE	12	FullPage
VKEY_PAGE_WIDTH	13	PageWidth
VKEY_ZOOM_IN	14	ZoomIn

VKEY_ZOOM_OUT	15	ZoomOut
VKEY_GRID	16	Grid
VKEY_PAGE_FIRST	17	PageFirst
VKEY_PAGE_LEFT	18	PageLeft
VKEY_PAGE_RIGHT	19	PageRight
VKEY_PAGE_LAST	20	PageLast
VKEY_HELP	21	Help
VKEY_INFO	22	Info
VKEY_CLOSE	23	Close
VKEY_GOTO_PAGE	24	GotoPage
VKEY_OPEN	25	Open
VKEY_SAVE	26	Save
VKEY_ESCAPE	27	Escape

See also:

DefineKey 282

7.89 GUITheme

[GUI Control Only, not supported by the Community Edition]

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

property GUITheme [integer] VPE.GUITheme

read / write; runtime only

Possible Values:

available themes are:

ActiveX / VCL	Value	Enum	Comment
VGUI_THEME_OFFICE2000	0	Office2000	
VGUI_THEME_OFFICE2003	1	Office2003	
VGUI_THEME_WHIDBEY	2	Whidbey	

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.

7.90 GUILanguage

[GUI Control 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.

property GUILanguage [integer] VPE.GUILanguage

write; runtime only

Possible Values:

available languages are:

ActiveX / VCL	Value	Enum	Comment
VGUI_LANGUAGE_USER_DEFINED	-1	UserDefined	
VGUI_LANGUAGE_ENGLISH	0	English	
VGUI_LANGUAGE_GERMAN	1	German	
VGUI_LANGUAGE_FRENCH	2	French	
VGUI_LANGUAGE_DUTCH	3	Dutch	
VGUI_LANGUAGE_SPANISH	4	Spanish	
VGUI_LANGUAGE_DANISH	5	Danish	
VGUI_LANGUAGE_SWEDISH	6	Swedish	
VGUI_LANGUAGE_FINNISH	7	Finnish	
VGUI_LANGUAGE_ITALIAN	8	Italian	
VGUI_LANGUAGE_NORWEGIAN	9	Norwegian	
VGUI_LANGUAGE_PORTUGUESE	10	Portuguese	

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 by calling SetResourceString 288.

Example:

ActiveX / VCL:

```
Doc.GUILanguage = VGUI_LANGUAGE_SPANISH
```

.NET:

```
Doc.GUILanguage = GUILanguage.Spanish
```

forces VPE to use spanish as language for the GUI

7.91 SetResourceString

[GUI Control Only, not supported by the Community Edition]

If you set the GUILanguage 287 to *UserDefined*, you can specify freely any text in any language for each element of the GUI.

method void VPE.SetResourceString(
ResourceId [int] *Id*,
string *Text*
)

ResourceId [int] Id
possible values are:

ActiveX / VCL	Value	Enum	Comment
VRSCID_COPY_OF	0	CopyOf	"Copy %u of %u"
VRSCID_PAGE_OF	1	PageOf	"Page %u of %u"
VRSCID_BAND	2	Band	"Band %u" (i.e. "stripe" = "band")
VRSCID_ERROR	3	Error	"Error"
VRSCID_WARNING	4	Warning	"Warning"
VRSCID_PRINTING	5	Printing	"Printing"
VRSCID_CANCEL	6	Cancel	"Cancel"
VRSCID_OK	7	Ok	"Ok"
Printer Setup Dialog:			
VRSCID_PD_TITLE	8	PdTitle	dialog title "Print"
VRSCID_PD_PRINTER	9	PdPrinter	group box "Printer"
VRSCID_PD_PROPERTIES	10	PdProperties	button "&Properties"
VRSCID_PD_NAME	11	PdName	printer name combo box "&Name:"
VRSCID_PD_RANGE	12	PdRange	group box "Print range"
VRSCID_PD_ALL	13	PdAll	radio button "&All"
VRSCID_PD_PAGES	14	PdPages	radio button "Pa&ges"
VRSCID_PD_FROM	15	PdFrom	static text "&from:"
VRSCID_PD_TO	16	PdTo	static text "&to:"
VRSCID_PD_COPIES	17	PdCopies	group box "Copies"
VRSCID_PD_NUM_COPIES	18	PdNumCopies	static text "Number of &copies:"
VRSCID_PD_COLLATE	19	PdCollate	static text "C&ollate"

Tooltips:			
VRSCID_PRINT_DOC	20	PrintDoc	"Print the Document"
VRSCID_FULL_PAGE	21	FullPage	"Show Full Page"
VRSCID_PAGE_WIDTH	22	PageWidth	"Fit Page Width"
VRSCID_ZOOM_IN	23	ZoomIn	"Zoom in"
VRSCID_ZOOM_OUT	24	ZoomOut	"Zoom out"
VRSCID_FIRST_PAGE	25	FirstPage	"Go to First Page"
VRSCID_PREV_PAGE	26	PrevPage	"Go to Previous Page"
VRSCID_NEXT_PAGE	27	NextPage	"Go to Next Page"
VRSCID_LAST_PAGE	28	LastPage	"Go to Last Page"
VRSCID_HELP	29	Help	"Help on Control"
VRSCID_INFORMATION	30	Information	"Program Information"
VRSCID_CLOSE_PREVIEW	31	ClosePreview	"Close Preview"
VRSCID_GRID	32	Grid	"Show / Hide Grid"
VRSCID_ENTER_PAGENO	33	EnterPageno	"enter page# with mouse-click"
VRSCID_ZOOM_FACTOR	34	ZoomFactor	"Zoom Factor"
VRSCID_STATUS	35	Status	"Status"
VRSCID_READY	36	Ready	"Ready"
VRSCID_EMAIL	37	Email	"eMail"
VRSCID_OPEN	38	Open	"Open"
VRSCID_SAVE	39	Save	"Save"
Other:			
VRSCID_ERROR_OPEN	40	ErrorOpen	"Could not open file!"
VRSCID_ERROR_WRITE	41	ErrorWrite	"Could not write file!"
Usage Help Dialog:			
VRSCID_USAGE	42	Usage	"Usage" (caption of the help-dialog)
VRSCID_MOUSE	43	Mouse	"Mouse"
VRSCID_USAGE_MOUSE	44	UsageMouse	<all mouse usage text, lines separated by carriage return or linefeed>
VRSCID_KEYBOARD	45	Keyboard	"Keyboard"
VRSCID_USAGE_KEYBOARD	46	UsageKeyboard	<all keyboard usage text, lines separated by carriage return or linefeed >

string Text
The text that shall be used for the specified resource.

7.92 hWndPreview

[GUI Control Only, .NET, ActiceX, VCL 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.

property IntPtr [long] VPE.hWndPreview

read; runtime only

Returns:

the window handle of the preview

7.93 PreviewWindow

[.NET only, GUI Control Only]

Returns the IWin32Window of the Preview. It can be used directly as parameter for many .NET methods, like for example MessageBox.Show().

property PreviewHandle VPE.PreviewWindow

read; runtime only

Returns:

the IWin32Window of the preview

PreviewHandle is defined as:

```
public class PreviewHandle : IWin32Window
```

7.94 ControlVersion

Returns the current version number of the ActiveX / VCL / .NET component.

property double VPE.ControlVersion

read; runtime only

Returns:
The current version number.

7.95 VpeVersion

Returns the current version number of the loaded VPE-DLL.

property double VPE.VpeVersion

read; runtime only

Returns:
The current version number.

7.96 Edition

Returns the Edition of the loaded Engine.

property Edition [long] VPE.Edition

read; runtime only

Returns:

ActiveX / VCL	Value	Enum	Comment
VEDITION_COMMUNITY	500	Community	
VEDITION_STANDARD	1000	Standard	
VEDITION_ENHANCED	2000	Enhanced	
VEDITION_PROFESSIONAL	3000	Professional	
VEDITION_ENTERPRISE	4000	Enterprise	
VEDITION_INTERACTIVE	5000	Interactive	

This page is intentionally left blank.

Printing Functions

8 Printing Functions

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

8.1 SetupPrinter

[Windows platform only; not supported by PHP and 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 DialogControl = 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() 189 and then SetupPrinter("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!

```
method integer VPE.SetupPrinter(
        string FileName,
        PrintDialogControl [integer] DialogControl
)
```

string FileName
(path and) file name of the file where to restore or store the setup or ""

PrintDialogControl [integer] DialogControl
possible values are:

ActiveX / VCL	Value	Enum	Comment

PRINTDLG_NEVER	0	Never	never show setup-dialog (if file_name is "", the last setting or the setting of the default-printer will be taken)
PRINTDLG_ONFAIL	1	OnFail	show setup-dialog only, if file-read fails
PRINTDLG_ALWAYS	2	Always	show setup-dialog always
PRINTDLG_FULL	4	Full	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 ReadPrinterSetup() 357 and you can write them using WritePrinterSetup() 356.

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 314 like DevPaperFormat 319 and DevOrientation 318, etc. to the current VPE Document (see PageFormat 367 and PageOrientation 378). 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 PageOrientation 378 (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 PageFormat 367 (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 choose 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 303.

Example:

ActiveX / VCL:

```
VPE.OpenDoc
// use PRINT_NO_AUTO_PAGE_DIMS, in order to be able to retrieve the
// values for the page dimensions from the PRS file:
VPE.PrintOptions = PRINT_ALL + PRINT_NO_AUTO_PAGE_DIMS
VPE.SetupPrinter "personal settings.prs", PRINTDLG_ONFAIL
VPE.PageWidth = VPE.DevPaperWidth
VPE.PageHeight = VPE.DevPaperHeight
VPE.PageOrientation = VPE.DevOrientation

// switch PrintOptions back and let VPE control the printer's page
// dimensions, i.e. clear the flag PRINT_NO_AUTO_PAGE_DIMS
VPE.PrintOptions = PRINT_ALL
```

.NET:

```
VPE.OpenDoc
// use NoAutoPageDims, in order to be able to retrieve the values
// for the page dimensions from the PRS file:
VB: VPE.PrintOptions = PrintOptions.PrintAll +
PrintOptions.NoAutoPageDims
C#: VPE.PrintOptions = PrintOptions.PrintAll |
PrintOptions.NoAutoPageDims
VPE.SetupPrinter "personal settings.prs", PrintDialogControl.OnFail
VPE.PageWidth = VPE.DevPaperWidth
VPE.PageHeight = VPE.DevPaperHeight
VPE.PageOrientation = VPE.DevOrientation

// switch PrintOptions back and let VPE control the printer's page
// dimensions, i.e. clear the flag PRINT_NO_AUTO_PAGE_DIMS
VPE.PrintOptions = PrintOptions.PrintAll
```

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 "VPE.PageFormat 367 = VPE.DevPaperFormat 319" **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 367 or PageWidth 375 and PageHeight 377 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.

8.2 PrintOptions

[Windows platform only; not supported by PHP and 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 SetupPrinter() 299 is called.

property PrintOptions [long] VPE.PrintOptions

write; runtime only

Possible Values:

a combination of the following values:

ActiveX / VCL	Value	Enum	Comment
PRINT_ALL	0	PrintAll	print all pages
PRINT_EVEN	1	PrintEven	print only even pages
PRINT_ODD	2	PrintOdd	print only odd pages
PRINT_NOABORTDLG	4	NoAbortDialog	show no abort / progress dialog during printing
PRINT_NO_AUTO_PAGE_DIMS	16	NoAutoPageDims	VPE shall not instruct the printer to use the page dimensions of the currently printed page

Default:

PRINT_ALL

for the VpeWebControl it is: PrintAll + NoAbortDialog, because it is not meaningful to show a progress / abort dialog on a server.

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:

ActiveX / VCL:

```
Doc.PrintOptions = PRINT_EVEN
```

print only even numbered pages

```
Doc.PrintOptions = PRINT_ODD + PRINT_NOABORTDLG
```

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

```
Doc.PrintOptions = 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

```
Doc.PrintOptions = PRINT_ALL
```

reset to default

.NET:
```
Doc.PrintOptions = PrintOptions.PrintEven
```
print only even numbered pages

```
Doc.PrintOptions = PrintOptions.PrintOdd + PrintOptions.NoAbortDialog
```
print only odd numbered pages and do not show the print-progress dialog

```
Doc.PrintOptions = PrintOptions.PrintAll + PrintOptions.NoAbortDialog
                   + PrintOptions.NoAutoPageDims
```
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

```
Doc.PrintOptions = PrintOptions.PrintAll
```
reset to default

C#:
```
Doc.PrintOptions = PrintOptions.PrintEven
```
print only even numbered pages

```
Doc.PrintOptions = PrintOptions.PrintOdd | PrintOptions.NoAbortDialog
```
print only odd numbered pages and do not show the print-progress dialog

```
Doc.PrintOptions = PrintOptions.PrintAll | PrintOptions.NoAbortDialog
                   | PrintOptions.NoAutoPageDims
```
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

```
Doc.PrintOptions = PrintOptions.PrintAll
```
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 choose 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() 299 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:

```
VPE.Device = "Epson LQ-550"
VPE.PageWidth = 760
VPE.PageHeight = 1280
```

That's it.

8.3 PrintPosMode

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

Specifies, how objects are positioned on the printer.

property PrintPosMode [integer] VPE.PrintPosMode

write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
PRINTPOS_ABSOLUTE	0	Absolute	absolute to the paper-edges (default)
PRINTPOS_RELATIVE	1	Relative	relative to the first printable position of the printer

Default:
PRINTPOS_ABSOLUTE

See also:
"Positioning On the Printer" in the Programmer's Manual
SetPrintOffset 307

8.4 SetPrintOffset

[Windows platform only; not supported by PHP and 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.

```
method void VPE.SetPrintOffset(
        VpeCoord OffsetX,
        VpeCoord OffsetY
)
```

VpeCoord *OffsetX, OffsetY*
the offset in metric or inch units, values can be positive or negative

Default:
OffsetX = 0 (no offset)
OffsetY = 0 (no offset)

See also:
"Positioning On the Printer" in the Programmer's Manual
PrintPosMode 306

8.5 PrintOffsetX

[Windows platform only; not supported by PHP and 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.

property VpeCoord VPE.PrintOffsetX

read / write; runtime only

Possible Values:
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
PrintPosMode 306

8.6 PrintOffsetY

[Windows platform only; not supported by PHP and 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.

property VpeCoord VPE.PrintOffsetY

read / write; runtime only

Possible Values:
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
PrintPosMode 306

8.7 PrintScale

[Windows platform only, Professional Edition and higher only, not supported by PHP]

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 308 - and PrintOffsetY 309 - 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.

property double VPE.PrintScale

read / write; runtime only

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:

```
VPE.PrintScale = 0.5
```

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

```
VPE.PrintScale = 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

8.8 PrintDoc

[Windows platform only; not supported by PHP and 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 PrintDoc() 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 OpenDoc() 189. 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.

method void VPE.PrintDoc(
    boolean *WithSetup*
)

boolean WithSetup

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

Remarks:

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

```
Call Report.PrintDoc(True or False)
Call Report.CloseDoc
```

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

8.9 IsPrinting

[Windows platform only; not supported by PHP]

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.

property boolean VPE.IsPrinting

read; runtime only

Possible Values:

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

Device Control Properties

9 Device Control Properties

[Windows platform only; not supported by PHP and 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() 189.

Remarks:

- Changing the properties (like DevBin, DevOrientation 318, DevPaperFormat 319, 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 333" 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.

9.1 DevEnum

[Windows platform only; not supported by PHP and 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 GetDevEntry() 316.

method long VPE.DevEnum(
)

Returns:
the number of printing devices installed in the system

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

Example:
see GetDevEntry() 316

9.2 GetDevEntry

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

After calling DevEnum() 315, 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.

method string VPE.GetDevEntry(
long *Index*
)

long Index
must be in the range between 0 and the value returned DevEnum() 315 - 1

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 DevEnum() 315 before.

Example:

```
long i, count
string s
count = DevEnum() - 1// initialize enumeration
for i = 0 to count
  s = DevEntry(i)
  MessageBox(s) // show each available device in a separate message
box
next i
```

Note:
Do not call DevEnum() 315 repeatedly, because each time VPE will query all printers from the system.

Wrong Example:

```
long i
string s
for i = 0 to DevEnum() - 1
  s = DevEntry(i)
  MessageBox(s) // show each available device in a separate message
box
next i
```

9.3 Device

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

Returns / sets the current output device. After a new output device is successfully selected, **all device settings (like DevPaperFormat 319, etc.) are lost!** You are also able to select the default output device by setting device to an empty string ("").

property string VPE.Device

read / write; runtime only

Possible Values:

the name of the output device

Remarks:

In case of an error, LastError 201 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 DevEnum() 315.

Example:

```
Doc.Device = "Office Printer"
```

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

```
Doc.Device = ""
```

Selects the default device of the system.

9.4 DevOrientation

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

Returns / 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 PageOrientation 378).

property PageOrientation [integer] VPE.DevOrientation

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VORIENT_PORTRAIT	1	Portrait	
VORIENT_LANDSCAPE	2	Landscape	

Remarks:

The DevOrientation property is available for total control. It is only useful if called while processing the BeforePrintNewPage() 148 event (VCL: OnPrintNewPage() 174, .NET: BeforePrintNewPage() 67). Otherwise the DevOrientation property is always overridden by the PageOrientation property. **You should always use the PageOrientation property to change the orientation for the preview and the printer.**

In case of an error, LastError 201 is set to VERR_COMMON. An error may occur if the device does not support changing the orientation.

Example:

ActiveX / VCL

```
Doc.DevOrientation = VORIENT_LANDSCAPE
```

.NET

```
Doc.DevOrientation = PageOrientation.Landscape
```

9.5 DevPaperFormat

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

Returns / 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 PageFormat 367).

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.

property integer VPE.DevPaperFormat

read / write; runtime only

Possible Values:

one of the predefined VPAPER_xyz constants (see below)

Remarks:

This property is available for total control. It is only useful if called while processing the BeforePrintNewPage() 148 event (VCL: OnPrintNewPage() 174, .NET: BeforePrintNewPage() 67). Otherwise the DevPaperFomat property is always overridden by the PageFormat property. **You should always use PageFormat 367 property to change the page format for the preview and the printer.**

In case of an error, LastError 201 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.

Example:

ActiveX / VCL

```
Doc.DevPaperFormat = VPAPER_CSHEET  // C Sheet, 17- by 22-inches
```

.NET

```
Doc.DevPaperFormat = PageFormat.Csheet  // C Sheet, 17- by 22-inches
```

ActiveX / VCL: *DevPaperFormat* 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

.NET, Java, PHP, Python, Ruby, etc. : *DevPaperFormat* can be one of the following values

```
public enum PageFormat
{
        A4,                         A4 Sheet, 210- by 297-millimeters
        Letter,                     US Letter, 8 1/2- by 11-inches
        Legal,                      Legal, 8 1/2- by 14-inches
        CSheet,                     C Sheet, 17- by 22-inches
        DSheet,                     D Sheet, 22- by 34-inches
        ESheet,                     E Sheet, 34- by 44-inches
        LetterSmall,                Letter Small, 8 1/2- by 11-inches
        Tabloid,                    Tabloid, 11- by 17-inches
        Ledger,                     Ledger, 17- by 11-inches
        Statement,                  Statement, 5 1/2- by 8 1/2-inches
        Executive,                  Executive, 7 1/4- by 10 1/2-inches
        A3,                         A3 sheet, 297- by 420-millimeters
        A4Small,                    A4 small sheet, 210- by 297-millimeters
        A5,                         A5 sheet, 148- by 210-millimeters
        B4,                         B4 sheet, 250- by 354-millimeters
        B5,                         B5 sheet, 182- by 257-millimeter paper
        Folio,                      Folio, 8 1/2- by 13-inch paper
        Quarto,                     Quarto, 215- by 275-millimeter paper
        Standard10x14,              10- by 14-inch sheet
        Standard11x17,              11- by 17-inch sheet
        Note,                       Note, 8 1/2- by 11-inches
        Envelope9,                  #9 Envelope, 3 7/8- by 8 7/8-inches
        Envelope10,                 #10 Envelope, 4 1/8- by 9 1/2-inches
        Envelope11,                 #11 Envelope, 4 1/2- by 10 3/8-inches
        Envelope12,                 #12 Envelope, 4 3/4- by 11-inches
        Envelope14,                 #14 Envelope, 5- by 11 1/2-inches
        EnvelopeDL,                 DL Envelope, 110- by 220-millimeters
        EnvelopeC5,                 C5 Envelope, 162- by 229-millimeters
        EnvelopeC3,                 C3 Envelope,  324- by 458-millimeters
        EnvelopeC4,                 C4 Envelope,  229- by 324-millimeters
        EnvelopeC6,                 C6 Envelope,  114- by 162-millimeters
        EnvelopeC65,                C65 Envelope, 114- by 229-millimeters
        EnvelopeB4,                 B4 Envelope,  250- by 353-millimeters
        EnvelopeB5,                 B5 Envelope,  176- by 250-millimeters
        EnvelopeB6,                 B6 Envelope,  176- by 125-millimeters
        EnvelopeItaly,              Italy Envelope, 110- by 230-millimeters
        EnvelopeMonarch,            Monarch Envelope, 3 7/8- by 7 1/2-inches
        EnvelopePersonal,           6 3/4 Envelope, 3 5/8- by 6 1/2-inches
        FanfoldUS,                  US Std Fanfold, 14 7/8- by 11-inches
        FanfoldStdGerman,           German Std Fanfold, 8 1/2- by 12-inches
        FanfoldLglGerman,           German Legal Fanfold, 8 1/2- by 13-inches
        UserDefined,                User Defined
        IsoB4,                      B4 (ISO) 250 x 353 mm
```

JapanesePostcard,	Japanese Postcard 100 x 148 mm
Standard9x11,	9 x 11 in
Standard10x11,	10 x 11 in
Standard15x11,	15 x 11 in
EnvelopeInvite,	Envelope Invite 220 x 220 mm
Reserved48,	RESERVED--DO NOT USE
Reserved49,	RESERVED--DO NOT USE
LetterExtra,	Letter Extra 9 \275 x 12 in
LegalExtra,	Legal Extra 9 \275 x 15 in
TabloidExtra,	Tabloid Extra 11.69 x 18 in
A4Extra,	A4 Extra 9.27 x 12.69 in
LetterTransverse,	Letter Transverse 8 \275 x 11 in
A4Transverse,	A4 Transverse 210 x 297 mm
LetterExtraTransverse,	Letter Extra Transverse 9\275 x 12 in
APlus,	SuperA/SuperA/A4 227 x 356 mm
BPlus,	SuperB/SuperB/A3 305 x 487 mm
LetterPlus,	Letter Plus 8.5 x 12.69 in
A4Plus,	A4 Plus 210 x 330 mm
A5Transverse,	A5 Transverse 148 x 210 mm
B5Transverse,	B5 (JIS) Transverse 182 x 257 mm
A3Extra,	A3 Extra 322 x 445 mm
A5Extra,	A5 Extra 174 x 235 mm
B5Extra,	B5 (ISO) Extra 201 x 276 mm
A2,	A2 420 x 594 mm
A3Transverse,	A3 Transverse 297 x 420 mm
A3ExtraTransverse,	A3 Extra Transverse 322 x 445 mm

Windows 2000 or Higher:

DblJapanesePostcard,	Japanese Double Postcard 200 x 148 mm
A6,	A6 105 x 148 mm
JapaneseEnvelopeKaku2,	Japanese Envelope Kaku #2
JapaneseEnvelopeKaku3,	Japanese Envelope Kaku #3
JapaneseEnvelopeChou3,	Japanese Envelope Chou #3
JapaneseEnvelopeChou4,	Japanese Envelope Chou #4
LetterRotated,	Letter Rotated 11 x 8 1/2 in
A3Rotated,	A3 Rotated 420 x 297 mm
A4Rotated,	A4 Rotated 297 x 210 mm
A5Rotated,	A5 Rotated 210 x 148 mm
B4JisRotated,	B4 (JIS) Rotated 364 x 257 mm
B5JisRotated,	B5 (JIS) Rotated 257 x 182 mm
JapanesePostcardRotated,	Japanese Postcard Rotated 148 x 100 mm
DblJapanesePostcardRotated,	Double Japanese Postcard Rotated 148 x 200 mm
A6Rotated,	A6 Rotated 148 x 105 mm
JapaneseEnvelopeKaku2Rotated,	Japanese Envelope Kaku #2 Rotated
JapaneseEnvelopeKaku3Rotated,	Japanese Envelope Kaku #3 Rotated
JapaneseEnvelopeChou3Rotated,	Japanese Envelope Chou #3 Rotated

JapaneseEnvelopeChou4Rotated,	Japanese Envelope Chou #4 Rotated
B6Jis,	B6 (JIS) 128 x 182 mm
B6JisRotated,	B6 (JIS) Rotated 182 x 128 mm
Standard12x11,	12 x 11 in
JapaneseEnvelopeYou4,	Japanese Envelope You #4
JapaneseEnvelopeYou4Rotated,	Japanese Envelope You #4 Rotated
Prc16K,	PRC 16K 146 x 215 mm
Prc32K,	PRC 32K 97 x 151 mm
Prc32KBig,	PRC 32K(Big) 97 x 151 mm
PrcEnvelope1,	PRC Envelope #1 102 x 165 mm
PrcEnvelope2,	PRC Envelope #2 102 x 176 mm
PrcEnvelope3,	PRC Envelope #3 125 x 176 mm
PrcEnvelope4,	PRC Envelope #4 110 x 208 mm
PrcEnvelope5,	PRC Envelope #5 110 x 220 mm
PrcEnvelope6,	PRC Envelope #6 120 x 230 mm
PrcEnvelope7,	PRC Envelope #7 160 x 230 mm
PrcEnvelope8,	PRC Envelope #8 120 x 309 mm
PrcEnvelope9,	PRC Envelope #9 229 x 324 mm
PrcEnvelope10,	PRC Envelope #10 324 x 458 m
Prc16KRotated,	PRC 16K Rotated
Prc32KRotated,	PRC 32K Rotated
Prc32KBigRotated,	PRC 32K(Big) Rotated
PrcEnvelope1Rotated,	PRC Envelope #1 Rotated 165 x 102 mm
PrcEnvelope2Rotated,	PRC Envelope #2 Rotated 176 x 102 mm
PrcEnvelope3Rotated,	PRC Envelope #3 Rotated 176 x 125 mm
PrcEnvelope4Rotated,	PRC Envelope #4 Rotated 208 x 110 mm
PrcEnvelope5Rotated,	PRC Envelope #5 Rotated 220 x 110 mm
PrcEnvelope6Rotated,	PRC Envelope #6 Rotated 230 x 120 mm
PrcEnvelope7Rotated,	PRC Envelope #7 Rotated 230 x 160 mm
PrcEnvelope8Rotated,	PRC Envelope #8 Rotated 309 x 120 mm
PrcEnvelope9Rotated,	PRC Envelope #9 Rotated 324 x 229 mm
PrcEnvelope10Rotated,	PRC Envelope #10 Rotated 458 x 324 mm

}

9.6 DevPaperWidth

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

Returns / 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, DevPaperFormat 319 returns VPAPER_USER_DEFINED (=0).

property VpeCoord VPE.DevPaperWidth

read / write; runtime only

Possible Values:

the paper width

Remarks:

In case of an error, LastError 201 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 PRINT_NEWPAGE event. Otherwise this property is always overridden by the PageFormat 367 or PageWidth 375 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:

```
Doc.DevPaperWidth = 10.5  // 10.5 cm / inch
```

9.7 DevPaperHeight

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

Returns / 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, DevPaperFormat 319 returns VPAPER_USER_DEFINED (=0). Setting this property makes sense for tractor printers with endless paper feeders.

property long VPE.DevPaperHeight

read / write; runtime only

Possible Values:

the paper height

Remarks:

In case of an error, LastError 201 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 PRINT_NEWPAGE event. Otherwise this property is always overridden by the PageFormat 367 or PageHeight 377 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 319 to a predefined paper format.

Example:

```
Doc.DevPaperHeight = 5.8  // 5.8 cm / inch
```

9.8 DevScalePercent

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

Returns / 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.

property integer VPE.DevScalePercent

read / write; runtime only

Possible Values:

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

Remarks:

In case of an error, LastError 201 is set to VERR_COMMON. An error may occur if the device does not support changing the scale.

9.9 DevPrintQuality

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

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

property DevPrintQuality [integer] VPE.DevPrintQuality

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VRES_DRAFT	-1	Draft	Draft Quality
VRES_LOW	-2	Low	Low Quality
VRES_MEDIUM	-3	Medium	Medium Quality
VRES_HIGH	-4	High	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 330 without error, this property should specify the x-resolution in DPI.

Remarks:

In case of an error, LastError 201 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 201 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:

ActiveX / VCL:

```
Doc.DevPrintQuality = VRES_DRAFT    // Draft Mode
Doc.DevPrintQuality = 300           // 300 DPI
```

.NET:

```
Doc.DevPrintQuality = DevPrintQuality.Draft   // Draft Mode
Doc.DevPrintQuality = (DevPrintQuality)300    // 300 DPI
```

9.10 DevYResolution

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

Returns / sets the y-resolution in dots per inch for the currently selected output device. If the output device initializes this property, the DevPrintQuality 329 property specifies the x-resolution, in dots per inch (DPI), of the printer.

property integer VPE.DevYResolution

read / write; runtime only

Possible Values:

the y-resolution in DPI

Remarks:

In case of an error, LastError 201 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 201 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:

```
Doc.DevPrintQuality = 300 // 300 DPI X-Resolution
Doc.DevYResolution = 300  // 300 DPI Y-Resolution
```

9.11 DevColor

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

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

property DevColor [integer] VPE.DevColor

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VCOLOR_MONOCHROME	1	Monochrome	
VCOLOR_COLOR	2	Color	

Remarks:

In case of an error, LastError 201 is set to VERR_COMMON. An error may occur if the device does not support setting the color mode.

Example:

ActiveX / VCL

```
Doc.DevColor = VCOLOR_MONOCHROME
```

.NET

```
Doc.DevColor = DevColor.Monochrome
```

9.12 DevDuplex

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

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

property DevDuplex [integer] VPE.DevDuplex

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VDUP_SIMPLEX	1	Simplex	
VDUP_VERTICAL	2	Vertical	
VDUP_HORIZONTAL	3	Horizontal	

Remarks:

In case of an error, LastError 201 is set to VERR_COMMON. An error may occur if the device does not support setting the duplex mode.

Example:

ActiveX / VCL

```
Doc.DevDuplex = VDUP_VERTICAL
```

.NET

```
Doc.DevDuplex = DevDuplex.Vertical
```

9.13 DevTTOption

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

Returns / specifies how TrueType® fonts should be printed on the currently selected output device.

property DevTTOption [integer] VPE.DevTTOption

read / write; runtime only

Possible Values:

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

Remarks:

In case of an error, LastError 201 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:

ActiveX / VCL

```
Doc.DevTTOption = VTT_BITMAP
```

.NET

```
Doc.DevTTOption = DevTTOption.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.

9.14 DevEnumPaperBins

[Windows platform only; not supported by PHP and 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 GetDevPaperBinName() 336.

method long VPE.DevEnumPaperBins(
)

Returns:

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

Remarks:

In case of an error, LastError 201 is set to VERR_COMMON. An error may occur if the device does not support multiple paper bins.

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

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

Example:

see GetDevPaperBinName() 336

9.15 GetDevPaperBinName

[Windows platform only; not supported by PHP and 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 DevEnumPaperBins() 335 before.

method string VPE.GetDevPaperBinName(
    long *Index*
)

long Index
  must be in the range between 0 and the value returned by DevEnumPaperBins() - 1

Returns:
  the Paper Bin name

Remarks:
  In case of an error, LastError 201 is set to VERR_COMMON. This method only works correctly if you called DevEnumPaperBins() before.

Example:

```
long count, i
string s
count = Doc.DevEnumPaperBins() - 1 // initialize enumeration
for i = 0 to count
   s = Doc.GetDevPaperBinName(i)
   MessageBox(s)    // show each available bin in a separate message box
next i
```

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

WRONG EXAMPLE:

```
long i
string s
for i = 0 to Doc.DevEnumPaperBins() - 1
    s = Doc.GetDevPaperBinName(i)
    MessageBox(s)    // show each available bin in a separate message
box
next i
```

9.16 GetDevPaperBinID

[Windows platform only; not supported by PHP and 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 DevPaperBin 339) 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 DevEnumPaperBins() 335 before.

method PaperBin [long] VPE.GetDevPaperBinID(
    long *Index*
)

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

Returns:
the Bin-ID, possible values are:

ActiveX / VCL	Value	Enum	Comment
VBIN_UNTOUCHED	0	Untouched	
VBIN_UPPER	1	Upper	
VBIN_ONLYONE	1	OnlyOne	
VBIN_LOWER	2	Lower	
VBIN_MIDDLE	3	Middle	
VBIN_MANUAL	4	Manual	
VBIN_ENVELOPE	5	Envelope	
VBIN_ENVMANUAL	6	EnvelopeManual	
VBIN_AUTO	7	Auto	
VBIN_TRACTOR	8	Tractor	
VBIN_SMALLFMT	9	SmallFormat	
VBIN_LARGEFMT	10	LargeFormat	
VBIN_LARGECAPACITY	11	LargeCapacity	
VBIN_CASSETTE	14	Cassette	

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 documentation for more specific descriptions of these options.

Remarks:
In case of an error, LastError 201 is set to VERR_COMMON. This method works only correctly, if you called DevEnumPaperBins() before.

Example:

see DevPaperBin 339

9.17 DevPaperBin

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

Selects a paper bin for the currently selected output device.

property PaperBin [long] VPE.DevPaperBin

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VBIN_UPPER	1	Upper	
VBIN_ONLYONE	1	OnlyOne	
VBIN_LOWER	2	Lower	
VBIN_MIDDLE	3	Middle	
VBIN_MANUAL	4	Manual	
VBIN_ENVELOPE	5	Envelope	
VBIN_ENVMANUAL	6	EnvelopeManual	
VBIN_AUTO	7	Auto	
VBIN_TRACTOR	8	Tractor	
VBIN_SMALLFMT	9	SmallFormat	
VBIN_LARGEFMT	10	LargeFormat	
VBIN_LARGECAPACITY	11	LargeCapacity	
VBIN_CASSETTE	14	Cassette	

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 documentation for more specific descriptions of these options.

Remarks:

In case of an error, LastError 201 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 DevEnumPaperBins() 335 and retrieve each Bin-ID associated with a Bin-Entry with GetDevPaperBinID() 337.

The *DevPaperBin* property is available for total control. It is only useful if called while processing the BeforePrintNewPage() 148 event (VCL: OnPrintNewPage() 174, .NET:

BeforePrintNewPage() 67). Otherwise the *DevPaperBin* property is always overridden by the PaperBin 380 property, which is specified separately for each page. **You should always use the PaperBin 380 property 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:

```
long count, i
string s
count = Doc.DevEnumPaperBins() - 1 // initialize enumeration
for i = 0 to count
  // retrieve the name of the i-th paper bin
  s = Doc.GetDevPaperBinName(i)

  // show each available paper bin name in a separate message box
  MessageBox(s)

  // select the bin
  Doc.DevPaperBin = Doc.GetDevPaperBinID(i)
next i
```

or, to select a bin directly:

ActiveX / VCL

```
Doc.DevPaperBin = VBIN_LOWER
```

.NET

```
Doc.DevPaperBin = PaperBin.Lower
```

9.18 DevPrinterOffsetX

[Windows platform only; not supported by PHP and 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.

property VpeCoord VPE.DevPrinterOffsetX

read; runtime only

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

Remarks:
In case of an error, LastError 201 is set to VERR_COMMON.

9.19 DevPrinterOffsetY

[Windows platform only; not supported by PHP and 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.

property VpeCoord VPE.DevPrinterOffsetY

read; runtime only

Returns:

the topmost y-position where the printer starts printing

Remarks:

In case of an error, LastError 201 is set to VERR_COMMON.

9.20 DevPrintableWidth

[Windows platform only; not supported by PHP and 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.

property VpeCoord VPE.DevPrintableWidth

read; runtime only

Returns:

the printable width

Remarks:

In case of an error, LastError 201 is set to VERR_COMMON.

9.21 DevPrintableHeight

[Windows platform only; not supported by PHP and 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.

property VpeCoord VPE.DevPrintableHeight

read; runtime only

Returns:
the printable height

Remarks:
In case of an error, LastError 201 is set to VERR_COMMON.

9.22 DevPhysPageWidth

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

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

property VpeCoord VPE.DevPhysPageWidth

read; runtime only

Returns:

the total page width of the currently selected output device

Remarks:

In case of an error, LastError 201 is set to VERR_COMMON.

9.23 DevPhysPageHeight

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

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

property VpeCoord VPE.DevPhysPageHeight

read; runtime only

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

Remarks:
In case of an error, LastError 201 is set to VERR_COMMON.

9.24 DevCopies

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

Returns / sets the number of copies for the currently selected output device.

property long VPE.DevCopies

read / write; runtime only

Possible Values:

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 OpenDoc() 189 or when you set explicitly the property Device 317).

9.25 DevCollate

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

Returns / specifies, if printed copies shall be collated on the currently selected output device.

property boolean VPE.DevCollate

read / write; runtime only

Possible Values:

Value	Description
True	collate
False	do not collate

9.26 DevFromPage

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

Returns / sets the starting page that shall be printed onto the currently selected output device.

property long VPE.DevFromPage

read / write; runtime only

Possible Values:

the starting page

Remarks:

This property is not written to a VPE setup file.

9.27 DevToPage

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

Returns / sets the ending page that shall be printed onto the currently selected output device.

property long VPE.DevToPage

read / write; runtime only

Possible Values:

the ending page

Remarks:

This property is not written to a VPE setup file.

9.28 DevToFile

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

Returns / specifies for the currently selected output device if the document shall be printed into a file.

property boolean VPE.DevToFile

read / write; runtime only

Possible Values:

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

Remarks:

This property is not written to a VPE setup file.

9.29 DevFileName

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

Returns / sets for the currently selected output device the file name that is used, if the document is printed into a file (see: DevToFile 351). If this property is not set or reset to an empty string (""), 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.

property string VPE.DevFileName

read / write; runtime only

Possible Values:
the (path- and) filename the document is printed to

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

9.30 DevJobName

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

Returns / 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 SwapFileName 195 is set).

property string VPE.DevJobName

read / write; runtime only

Possible Values:
job name

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

9.31 DevSendData

[Windows platform only; not supported by PHP and 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).

method long VPE.DevSendData(
string *Data*
)

string Data
the string with the data

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError 201 is set to VERR_COMMON. An error may occur if the device does not support sending binary data.

ActiveX: You may only call this function while processing the event DoPrintDevData() 150.

VCL: You may only call this function while processing the event OnPrintDevData() 176.

.NET: You may only call this function while processing the event PrintDevData() 70.
In C# the values true and false are not compatible with the integer datatype.
Use 1 for true and 0 for false.

If you call DevSendData() while not processing the event, the method does nothing.

Example:

Example in Visual Basic (while processing the DoPrintDevData() event):

ActiveX:

```
Const RevLandScape = "&l3O"   ' PCL command to change the Paper
                              ' orientation to Reverse Landscape.
PCL_Escape$ = Chr$(27) + RevLandScape
Doc.DevSendData(PCL_Escape$)
ResultingAction = PRINT_ACTION_CHANGE
```

.NET:

```
const string RevLandScape = "&l3O"  // PCL command to change the Paper
                                    // orientation to Reverse
Landscape.
PCL_Escape = Chr(27) + RevLandScape
Doc.DevSendData(PCL_Escape)
event.PrintResultingAction = PrintResultingAction.Change
```

The characters in the string are: "&" + the letter "l" + "3" + the letter "O"

9.32 WritePrinterSetup

[Windows platform only; not supported by PHP and 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 SetupPrinter() 299.

method boolean VPE.WritePrinterSetup(
string *FileName*
)

string FileName
the name of the file, the setup is written to

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError 201 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 201 = 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 SetupPrinter() 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.

9.33 ReadPrinterSetup

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

Reads a printer setup file that was previously written by SetupPrinter() 299 or WritePrinterSetup() 356 and uses the settings contained in the file.

method boolean VPE.ReadPrinterSetup(
string *FileName*
)

string FileName
the name of the file, the setup is read from

Returns:

Value	Description
True	success
False	failure

Remarks:
In case of an error, LastError 201 is set to VERR_COMMON.

see WritePrinterSetup() 356

This page is intentionally left blank.

Layout Functions

10 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: "Dynamic Positioning" in the Programmer's Manual.

10.1 UnitTransformation

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, as well as the old (prior to v4.0) 0.1mm units.

Internally, VPE computes object positions and dimensions with a precision of 1/10.000 mm.

property double VPE. UnitTransformation

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Description
VUNIT_FACTOR_CM	100000	Centimeter	coordinates are handled in cm units by the VPE API
VUNIT_FACTOR_INCH	254000	Inch	coordinates are handled in inch units by the VPE API
VUNIT_FACTOR_MM10	1000	Mm10	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 RulersMeasure 265.

Examples:

ActiveX / VCL:

```
Doc.UnitTransformation = VUNIT_FACTOR_MM10
```

.NET Visual Basic:

```
Doc.UnitTransformation = UnitFactor.Mm10
```

.NET C#:

```
Doc.UnitTransformation = (double)UnitFactor.Mm10
```

Sets the unit transformation to the old 0.1 mm units.

10.2 EngineRenderMode

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).

property EngineRenderMode [integer] VPE. EngineRenderMode

read / write; runtime only

Possible Values:

the renderer mode

ActiveX / VCL	Value	Enum	Description
VENGINE_RENDER_MODE_VER3	0	Ver3	VPE Renders Text compatible to Version 3.xx
VENGINE_RENDER_MODE_VER4	1	Ver4	VPE Renders Text compatible to Version 4.00 and higher

Remarks:

Reading an old v3.xx VPE document file will turn VPE into VER3 Mode for the whole document!

10.3 PageBreak

Adds a new blank page to the end of the document. Then VPE sets internally the CurrentPage 366 to this page, so that all further output-calls will draw onto this new page.

method boolean VPE.PageBreak(
)

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError 201 is set. An error can occur, if the document is SwapFile based - i.e. you have set the property SwapFileName 195 - and there was an error writing to the SwapFile, because for example the harddisk is full.

10.4 AutoBreakMode

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

property AutoBreakMode [integer] VPE.AutoBreakMode

read / write; runtime only

Possible Values:

ActiveX / VCL	Enum	Comment
AUTO_BREAK_ON	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	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	NoLimits	Remaining text is NOT broken onto the next page(s), it can be placed anywhere on the paper with no limits.
AUTO_BREAK_FULL	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

10.5 PageCount

Retrieve the numbers of pages of the document

property long VPE.PageCount

read; runtime only

Returns:

The numbers of pages in the document

10.6 CurrentPage

Moves to the current active page, or returns the number of the current active page. This is the page where you can insert objects, all further output-calls will insert objects onto this page. It is independently from the Visual Page 216 shown in the preview. (see "Multipage Documents" in the Programmer's Manual)

You can move at any time to any page to insert objects.

property long VPE.CurrentPage

read / write; runtime only

Possible Values:

The number of the current active page.

Remarks:

In case of an error, LastError 201 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.

Examples:

```
Doc.CurrentPage = 5     // moves to page 5
n = Doc.CurrentPage     // returns the value 5
```

```
Doc.CurrentPage = 5     // moves to page 5
Doc.Print(1, 1, "This is page 5")
Doc.CurrentPage = 2     // moves to page 2
Doc.Print(1, 1, "This is page 2")
```

10.7 PageFormat

Sets / returns the page format for the **current page**. You can set the page format for each page separately. All newly with PageBreak() 363 added pages will have the page format which was specified at last.

property integer VPE.PageFormat

read / write; design & runtime

Possible Values:

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 choose 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 303.

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:

```
VPE.OpenDoc
// the default printer is chosen automatically, so just copy the
dimensions
VPE.PageWidth = VPE.DevPaperWidth
VPE.PageHeight = VPE.DevPaperHeight
VPE.PageOrientation = VPE.DevOrientation
```

Copying the printer's page format to the document from a specific printer:

```
VPE.OpenDoc
// choose "Office Printer 2"
VPE.Device = "Office Printer 2"
VPE.PageWidth = VPE.DevPaperWidth
VPE.PageHeight = VPE.DevPaperHeight
VPE.PageOrientation = VPE.DevOrientation
```

Copying the printer's page format to the document from a PRS file:

ActiveX / VCL:

```
VPE.OpenDoc
// use PRINT_NO_AUTO_PAGE_DIMS, in order to be able to retrieve the
// values for the page dimensions from the PRS file:
VPE.PrintOptions = PRINT_ALL + PRINT_NO_AUTO_PAGE_DIMS
VPE.SetupPrinter "personal settings.prs", PRINTDLG_ONFAIL
VPE.PageWidth = VPE.DevPaperWidth
VPE.PageHeight = VPE.DevPaperHeight
VPE.PageOrientation = VPE.DevOrientation

// switch PrintOptions back and let VPE control the printer's page
// dimensions, i.e. clear the flag PRINT_NO_AUTO_PAGE_DIMS
VPE.PrintOptions = PRINT_ALL
```

.NET:

```
VPE.OpenDoc
// use PRINT_NO_AUTO_PAGE_DIMS, in order to be able to retrieve the
// values for the page dimensions from the PRS file:
VB: VPE.PrintOptions = PrintOptions.All + PrintOptions.NoAutoPageDims
C#: VPE.PrintOptions = PrintOptions.All | PrintOptions.NoAutoPageDims
VPE.SetupPrinter "personal settings.prs", PrintDialogControl.OnFail
VPE.PageWidth = VPE.DevPaperWidth
VPE.PageHeight = VPE.DevPaperHeight
VPE.PageOrientation = VPE.DevOrientation

// switch PrintOptions back and let VPE control the printer's page
// dimensions, i.e. clear the flag PRINT_NO_AUTO_PAGE_DIMS
VPE.PrintOptions = PrintOptions.All
```

You will notice that we are using PageWidth 375 and PageHeight 377 in the examples above. The reason is, that this circumvents a bug in many printer drivers, which return wrong values for DevPaperFormat 319 in case a user defined format has been assigned to the printer. Because of this, the construction "VPE.PageFormat = VPE.DevPaperFormat 319" **can not be used reliably**. Instead, use the code from the examples above.

ActiveX / VCL: *PageFormat* 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

.NET, Java, PHP, Python, Ruby, etc.: *PageFormat* can be one of the following values

```
public enum PageFormat
{
        A4,                         A4 Sheet, 210- by 297-millimeters
        Letter,                     US Letter, 8 1/2- by 11-inches
```

Legal,	Legal, 8 1/2- by 14-inches
CSheet,	C Sheet, 17- by 22-inches
DSheet,	D Sheet, 22- by 34-inches
ESheet,	E Sheet, 34- by 44-inches
LetterSmall,	Letter Small, 8 1/2- by 11-inches
Tabloid,	Tabloid, 11- by 17-inches
Ledger,	Ledger, 17- by 11-inches
Statement,	Statement, 5 1/2- by 8 1/2-inches
Executive,	Executive, 7 1/4- by 10 1/2-inches
A3,	A3 sheet, 297- by 420-millimeters
A4Small,	A4 small sheet, 210- by 297-millimeters
A5,	A5 sheet, 148- by 210-millimeters
B4,	B4 sheet, 250- by 354-millimeters
B5,	B5 sheet, 182- by 257-millimeter paper
Folio,	Folio, 8 1/2- by 13-inch paper
Quarto,	Quarto, 215- by 275-millimeter paper
Standard10x14,	10- by 14-inch sheet
Standard11x17,	11- by 17-inch sheet
Note,	Note, 8 1/2- by 11-inches
Envelope9,	#9 Envelope, 3 7/8- by 8 7/8-inches
Envelope10,	#10 Envelope, 4 1/8- by 9 1/2-inches
Envelope11,	#11 Envelope, 4 1/2- by 10 3/8-inches
Envelope12,	#12 Envelope, 4 3/4- by 11-inches
Envelope14,	#14 Envelope, 5- by 11 1/2-inches
EnvelopeDL,	DL Envelope, 110- by 220-millimeters
EnvelopeC5,	C5 Envelope, 162- by 229-millimeters
EnvelopeC3,	C3 Envelope, 324- by 458-millimeters
EnvelopeC4,	C4 Envelope, 229- by 324-millimeters
EnvelopeC6,	C6 Envelope, 114- by 162-millimeters
EnvelopeC65,	C65 Envelope, 114- by 229-millimeters
EnvelopeB4,	B4 Envelope, 250- by 353-millimeters
EnvelopeB5,	B5 Envelope, 176- by 250-millimeters
EnvelopeB6,	B6 Envelope, 176- by 125-millimeters
EnvelopeItaly,	Italy Envelope, 110- by 230-millimeters
EnvelopeMonarch,	Monarch Envelope, 3 7/8- by 7 1/2-inches
EnvelopePersonal,	6 3/4 Envelope, 3 5/8- by 6 1/2-inches
FanfoldUS,	US Std Fanfold, 14 7/8- by 11-inches
FanfoldStdGerman,	German Std Fanfold, 8 1/2- by 12-inches
FanfoldLglGerman,	German Legal Fanfold, 8 1/2- by 13-inches
UserDefined,	User Defined
IsoB4,	B4 (ISO) 250 x 353 mm
JapanesePostcard,	Japanese Postcard 100 x 148 mm
Standard9x11,	9 x 11 in
Standard10x11,	10 x 11 in
Standard15x11,	15 x 11 in
EnvelopeInvite,	Envelope Invite 220 x 220 mm
Reserved48,	RESERVED--DO NOT USE
Reserved49,	RESERVED--DO NOT USE
LetterExtra,	Letter Extra 9 \275 x 12 in

LegalExtra,	Legal Extra 9 \275 x 15 in
TabloidExtra,	Tabloid Extra 11.69 x 18 in
A4Extra,	A4 Extra 9.27 x 12.69 in
LetterTransverse,	Letter Transverse 8 \275 x 11 in
A4Transverse,	A4 Transverse 210 x 297 mm
LetterExtraTransverse,	Letter Extra Transverse 9\275 x 12 in
APlus,	SuperA/SuperA/A4 227 x 356 mm
BPlus,	SuperB/SuperB/A3 305 x 487 mm
LetterPlus,	Letter Plus 8.5 x 12.69 in
A4Plus,	A4 Plus 210 x 330 mm
A5Transverse,	A5 Transverse 148 x 210 mm
B5Transverse,	B5 (JIS) Transverse 182 x 257 mm
A3Extra,	A3 Extra 322 x 445 mm
A5Extra,	A5 Extra 174 x 235 mm
B5Extra,	B5 (ISO) Extra 201 x 276 mm
A2,	A2 420 x 594 mm
A3Transverse,	A3 Transverse 297 x 420 mm
A3ExtraTransverse,	A3 Extra Transverse 322 x 445 mm

Windows 2000 or Higher:

DblJapanesePostcard,	Japanese Double Postcard 200 x 148 mm
A6,	A6 105 x 148 mm
JapaneseEnvelopeKaku2,	Japanese Envelope Kaku #2
JapaneseEnvelopeKaku3,	Japanese Envelope Kaku #3
JapaneseEnvelopeChou3,	Japanese Envelope Chou #3
JapaneseEnvelopeChou4,	Japanese Envelope Chou #4
LetterRotated,	Letter Rotated 11 x 8 1/2 in
A3Rotated,	A3 Rotated 420 x 297 mm
A4Rotated,	A4 Rotated 297 x 210 mm
A5Rotated,	A5 Rotated 210 x 148 mm
B4JisRotated,	B4 (JIS) Rotated 364 x 257 mm
B5JisRotated,	B5 (JIS) Rotated 257 x 182 mm
JapanesePostcardRotated,	Japanese Postcard Rotated 148 x 100 mm
DblJapanesePostcardRotated,	Double Japanese Postcard Rotated 148 x 200 mm
A6Rotated,	A6 Rotated 148 x 105 mm
JapaneseEnvelopeKaku2Rotated,	Japanese Envelope Kaku #2 Rotated
JapaneseEnvelopeKaku3Rotated,	Japanese Envelope Kaku #3 Rotated
JapaneseEnvelopeChou3Rotated,	Japanese Envelope Chou #3 Rotated
JapaneseEnvelopeChou4Rotated,	Japanese Envelope Chou #4 Rotated
B6Jis,	B6 (JIS) 128 x 182 mm
B6JisRotated,	B6 (JIS) Rotated 182 x 128 mm
Standard12x11,	12 x 11 in
JapaneseEnvelopeYou4,	Japanese Envelope You #4
JapaneseEnvelopeYou4Rotated,	Japanese Envelope You #4 Rotated
Prc16K,	PRC 16K 146 x 215 mm
Prc32K,	PRC 32K 97 x 151 mm

Prc32KBig,	PRC 32K(Big) 97 x 151 mm
PrcEnvelope1,	PRC Envelope #1 102 x 165 mm
PrcEnvelope2,	PRC Envelope #2 102 x 176 mm
PrcEnvelope3,	PRC Envelope #3 125 x 176 mm
PrcEnvelope4,	PRC Envelope #4 110 x 208 mm
PrcEnvelope5,	PRC Envelope #5 110 x 220 mm
PrcEnvelope6,	PRC Envelope #6 120 x 230 mm
PrcEnvelope7,	PRC Envelope #7 160 x 230 mm
PrcEnvelope8,	PRC Envelope #8 120 x 309 mm
PrcEnvelope9,	PRC Envelope #9 229 x 324 mm
PrcEnvelope10,	PRC Envelope #10 324 x 458 m
Prc16KRotated,	PRC 16K Rotated
Prc32KRotated,	PRC 32K Rotated
Prc32KBigRotated,	PRC 32K(Big) Rotated
PrcEnvelope1Rotated,	PRC Envelope #1 Rotated 165 x 102 mm
PrcEnvelope2Rotated,	PRC Envelope #2 Rotated 176 x 102 mm
PrcEnvelope3Rotated,	PRC Envelope #3 Rotated 176 x 125 mm
PrcEnvelope4Rotated,	PRC Envelope #4 Rotated 208 x 110 mm
PrcEnvelope5Rotated,	PRC Envelope #5 Rotated 220 x 110 mm
PrcEnvelope6Rotated,	PRC Envelope #6 Rotated 230 x 120 mm
PrcEnvelope7Rotated,	PRC Envelope #7 Rotated 230 x 160 mm
PrcEnvelope8Rotated,	PRC Envelope #8 Rotated 309 x 120 mm
PrcEnvelope9Rotated,	PRC Envelope #9 Rotated 324 x 229 mm
PrcEnvelope10Rotated,	PRC Envelope #10 Rotated 458 x 324 mm

}

10.8 PageWidth

Sets / returns the page width for the **current page** to a user defined value. Also all newly with PageBreak() 363 added pages will have this width.

property VpeCoord VPE.PageWidth

read / write; design & runtime

Possible Values:
the width of the current page

Default:
21 (21cm = width of VPAPER_A4)

Remarks:
see PageFormat 367

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 choose 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 303.

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:

```
Vpe.Device = "Epson LQ-550";
Vpe.PageWidth = 7.60;
Vpe.PageHeight = 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.

10.9 PageHeight

Sets the page height for the **current page** to a user defined value. Also all newly with PageBreak() 363 added pages will have this page format.

property VpeCoord VPE.PageHeight

read / write; design & runtime

Possible Values:
the height of the current page

Default:
29.70 (29.7cm = height of VPAPER_A4)

Remarks:
see PageFormat 367
see PageWidth 375

10.10 PageOrientation

Sets / returns the page orientation for the **current page**. Also all newly with PageBreak() 363 added pages will have this orientation. The orientation is reflected in the preview if PaperView 268 = True. Also the printer will print the page in the specified orientation.

property PageOrientation [integer] VPE.PageOrientation

read / write; design & runtime

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VORIENT_PORTRAIT	1	Portrait	
VORIENT_LANDSCAPE	2	Landscape	

Remarks:

Changing the orientation also modifies the OutRect 395 of the current page and the DefOutRect 394 (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 318. You should always use this function to specify the orientation.

Example:

ActiveX / VCL:

```
VPE.OpenDoc
VPE.SetupPrinter("personal settings.prs", PRINTDLG_ONFAIL)
VPE.PageFormat = VPE.DevPaperFormat
VPE.PageOrientation = VPE.DevOrientation
```

.NET:

```
VPE.OpenDoc
VPE.SetupPrinter("personal settings.prs", PrintDialogControl.OnFail)
VPE.PageFormat = VPE.DevPaperFormat
VPE.PageOrientation = VPE.DevOrientation
```

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:

ActiveX / VCL:

```
Doc.DevOrientation = VORIENT_LANDSCAPE
Doc.PageBreak()
Doc.DevOrientation = VORIENT_PORTRAIT
Doc.PageBreak()
Doc.DevOrientation = VORIENT_LANDSCAPE
```

.NET:

```
Doc.DevOrientation = PageOrientation.Landscape
Doc.PageBreak()
Doc.DevOrientation = PageOrientation. Portrait
Doc.PageBreak()
Doc.DevOrientation = PageOrientation.Landscape
```

This example sets the orientation for the current page to Landscape, adds a new page and sets the Orientation of the new added page to Portrait and adds a third page and sets the Orientation of the new added page back to Landscape.

10.11 PaperBin

[Not supported by the Community Edition]

Sets the paper bin for the **current page**. Also all newly with PageBreak() 363 added pages will print to this bin.

property PaperBin [integer] VPE.PaperBin

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VBIN_UNTOUCHED	0	Untouched	
VBIN_UPPER	1	Upper	
VBIN_ONLYONE	1	OnlyOne	
VBIN_LOWER	2	Lower	
VBIN_MIDDLE	3	Middle	
VBIN_MANUAL	4	Manual	
VBIN_ENVELOPE	5	Envelope	
VBIN_ENVMANUAL	6	EnvelopeManual	
VBIN_AUTO	7	Auto	
VBIN_TRACTOR	8	Tractor	
VBIN_SMALLFMT	9	SmallFormat	
VBIN_LARGEFMT	10	LargeFormat	
VBIN_LARGECAPACITY	11	LargeCapacity	
VBIN_CASSETTE	14	Cassette	

Not all of the bin options are available on every printer. Check the printer 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 and use GetDevPaperBinID 337 (please see below for details).

Default:

VBIN_UNTOUCHED

Remarks:

This property is independent from DevPaperBin 339. 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 303.

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 (see GetDevPaperBinID 337) and the corresponding bin names only, see DevEnumPaperBins 335 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:

ActiveX / VCL:
```
Doc.PaperBin = VBIN_UPPER
```
Will instruct the printer during printing, to print this page on the upper paper bin (if available).

```
Doc.PageBreak()
Doc.PaperBin = VBIN_LOWER
```
Adds a new page and will instruct the printer during printing, to print this page on the lower paper bin (if available).

```
Doc.PageBreak()
Doc.PaperBin = 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).

.NET:
```
Doc.PaperBin = PaperBin.Upper
```
Will instruct the printer during printing, to print this page on the upper paper bin (if available).

```
Doc.PageBreak()
Doc.PaperBin = PaperBin.Lower
```
Adds a new page and will instruct the printer during printing, to print this page on the lower paper bin (if available).

```
Doc.PageBreak()
Doc.PaperBin = PaperBin.Upper
```

Adds a third page and will instruct the printer during printing, to print this page on the upper paper bin again (if available).

10.12 StoreSet

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.

method void VPE.StoreSet(
long *ID*
)

long ID
the ID under which you store the properties

Remarks:
The following properties are stored:
FontName 477
FontSize 478
PenSize 443
PenStyle 444
PenColor 445
Bold 487
Italic 490
Underline 488
StrikeOut 491
TextAlignment 486
TextColor 492
BkgColor 452
BkgMode 451
GradientStartColor 453
GradientEndColor 454
BkgGradientTriColor 455
BkgGradientMiddleColorPosition 456
BkgGradientMiddleColor 457
GradientRotation 458
HatchStyle 463
HatchColor 464
CornerRadius 465
AutoBreakMode 364
Rotation 398

PicturePage 528
PictureType 524
PictureKeepAspect 530
PictureCache 526
PictureScale2Gray 535
PictureScale2GrayFloat 536
PictureX2YResolution 533
PictureBestFit 531
PictureEmbedInDoc 529
PictureDrawExact 534

PictureDefaultDPIX 532
PictureDefaultDPIY 532

JpegExportOptions 661
TiffExportOptions 662
BmpExportOptions 663
PnmExportOptions 664
GifExportOptions 665
PictureExportColorDepth 666
PictureExportDither 667

Charset 481
CharPlacement 503
InsertAtBottomZOrder 407
RTFParagraph - The complete Paragraph Settings 623
ChartProperties 710

BarcodeMainText 546
BarcodeAddText 547
BarcodeAlignment 548
BarcodeAutoChecksum 549
BarcodeThinBar 550
BarcodeThickBar 551

Viewable 399
Printable 400
Streamable 402
Shadowed 403

CharacterCount 795
SubdividerPenSize 796
SubdividerPenColor 797
AltSubdividerNPosition 798
AltSubdividerPenSize 799
AltSubdividerPenColor 800
BottomLinePenSize 801
BottomLinePenColor 802
SubdividerStyle 803
AltSubdividerStyle 804
EditFlags 805

BookmarkDestination 951
BookmarkStyle 953
BookmarkColor 954

Bar2DAlignment 557
DataMatrixEncodingFormat 558
DataMatrixEccType 559
DataMatrixRows 560
DataMatrixColumns 561
DataMatrixMirror 562
DataMatrixBorder 563
QRCodeVersion 566

- QRCodeEccLevel 567
- QRCodeMode 568
- QRCodeBorder 569
- PDF417ErrorLevel 578
- PDF417Columns 580
- PDF417Rows 579
- AztecFlags 584
- AztecControl 585
- AztecMenu 586
- AztecMultipleSymbols 587
- AztecID 588

Example:

```
Doc.StoreSet(1)      // store the current settings
Doc.FontSize = 12    // modify the current properties,
Doc.PenSize = 6      // and output some text
Doc.WriteBox(1, 1, "Hello World!")
Doc.UseSet(1)        // return to the original settings
Doc.RemoveSet(1)     // delete the stored settings
```

10.13 UseSet

This resets all properties to the stored values.

method void VPE.UseSet(
        long *ID*
)

long ID
    the ID under which you stored the properties

See also:
    StoreSet 383

10.14 RemoveSet

Removes the setting specified under ID from memory.

```
method void VPE.RemoveSet(
        long ID
)
```

long ID
the ID under which you stored the properties

See also:
StoreSet 383

10.15 nLeft, nTop, nRight, nBottom

Returns / sets the specific coordinate of the last inserted object. Setting these properties is also possible. You may also use the V-Flags VLEFT, VTOP, VRIGHT, VBOTTOM, if you don't want to compute offsets. For the .NET component we always recommend to use the n-Properties instead of the V-Flags.

See "Dynamic Positioning" in the Programmer's Manual for details.

property VpeCoord VPE.nLeft
property VpeCoord VPE.nTop
property VpeCoord VPE.nRight
property VpeCoord VPE.nBottom

read / write; runtime only

Possible Values:

The coordinates of the last inserted object.

Example:

```
Doc.Write(1, 2, 4, 3, "Hello")
```

After executing this method the properties contain the following values:
nLeft = 1, nTop = 2, nRight = 4, nBottom = 3

You can insert an object at the right border of the last inserted object with:

```
Doc.Print(Doc.nRight, Doc.nTop, " World!")
```

The equal result (but with faster processing) is created with the following statement:

ActiveX / VCL:

```
Doc.Print(VRIGHT, VTOP, " World!")
```

.NET:

```
Doc.Print(VpeControl.VRIGHT, VpeControl.VTOP, " World!")
```

Note: for the .NET component we always recommend to use the n-Properties instead of the V-Flags.

If you want to insert the second object with an offset, you can only use:

```
Doc.Print(Doc.nRight + 0.5, Doc.nTop, " 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:

```
Doc.Print(VRIGHT + 0.5, VTOP, " World!")
```

because VRIGHT is a constant (its value is -2147483551) which instructs VPE to retrieve the value of the property nRight internally. If you add 0.5, the result is -2147483550.5, which is interpreted as a standard coordinate.

Remarks:

For completeness, we list the values of the V-Flags here.

You should always use the named constants instead of the values.

Constant	Value
VLEFT	-2147483550
VRIGHT	-2147483551
VTOP	-2147483554
VBOTTOM	-2147483555

10.16 nWidth, nHeight

Returns the width / height of the last inserted object.

See "Dynamic Positioning" in the Programmer's Manual for details.

property VpeCoord VPE.nWidth
property VpeCoord VPE.nHeight

read; runtime only

Possible Values:
The width / height of the last inserted object.

10.17 VFREE

[NOT for .NET, see nFree 392]

A flag for indicating that VPE shall compute a coordinate dynamically. For text and images it can be used for the right coordinate (width) as well as the bottom coordinate (height). For text it means that the coordinate shall be computed due to the text-length and font size when a text object is inserted. For images the coordinate will be computed based on the resolution and dimensions found in the image file. For Rich Text (RTF) you may only set the bottom coordinate to VFREE, the right coordinate can not be dynamic.

See "Dynamic Positioning" in the Programmer's Manual for details.

Example:

```
VPE.Write(1, 1, VFREE, VFREE, "Hello World!")
```

Remarks:

For completeness, we list the value of VFREE here.

You should always use the named constants instead of the values.

Constant	Value
VFREE	-2147483549

10.18 nFree

[.NET and Java Only]

Instructs VPE to compute the coordinate depending on the content of the object (does only work for text, rich text and images).

See "Dynamic Positioning" in the Programmer's Manual for details.

property VpeCoord VPE.nFree

read; runtime only

Remarks:

This property is not available for the ActiveX / VCL.
Use the constant VFREE with the ActiveX / VCL.

Example:

```
Doc.Picture(1, 1, Doc.nFree, Doc.nFree, "test.bmp")
```

Instructs VPE to compute the dimensions for the right and bottom coordinate depending on the size of the image named "test.bmp“.

10.19 nLeftMargin, nTopMargin, nRightMargin, nBottomMargin

Returns / sets the specific coordinate of the Page Margins.

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.

See "Dynamic Positioning" in the Programmer's Manual for details.

property VpeCoord VPE.nLeftMargin
property VpeCoord VPE.nTopMargin
property VpeCoord VPE.nRightMargin
property VpeCoord VPE.nBottomMargin

read / write; runtime only

Possible Values:

The coordinates of the Page Margins in metric or inch units.

Remarks:

For completeness, we list the values of the V-Flags here.

You should always use the named constants instead of the values.

Constant	Value
VLEFTMARGIN	-2147483552
VRIGHTMARGIN	-2147483553
VTOPMARGIN	-2147483556
VBOTTOMMARGIN	-2147483557

If you want to use the value of a margin directly, you may use the V-Flags VLEFTMARGIN, VTOPMARGIN, VRIGHTMARGIN, VBOTTOMMARGIN instead of using the above n-properties (in case you don't want to compute offsets).

Example: VPE.Write(VLEFTMARGIN, ...) and VPE.Write(VPE.nLeftMargin, ...) are identical, but the first version is a bit faster.

But if you want to use an offset, you must use the n-properties.

Example: VPE.Write(VPE.nLeftMargin + 0.5, ...) is correct, whilst VPE.Write(VLEFTMARGIN + 0.5, ...) is not possible.

For the .NET component we always recommend to use the n-Properties instead of the V-Flags.

See also:

SetOutRect 395

10.20 SetDefOutRect

Sets the **default** output rectangle (= virtual Page Margins), which will be used for each new created page - NOT for the current page.

The output rectangle is very important for use with the AutoBreakMode 364 option of VPE for rendering text.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

method void VPE.SetDefOutRect(
  VpeCoord *Left*,
  VpeCoord *Top*,
  VpeCoord *Right*,
  VpeCoord *Bottom*
)

VpeCoord *Left, Top, Right, Bottom*
  the output rectangle

Remarks:

The method SetOutRect() 395 calls internally SetDefOutRect() and sets the DefOutRect to the same rectangle. Therefore if you need to call SetOutRect() and SetDefOutRect() with different values at the same time, always call SetOutRect() first.

Example:

```
Doc.SetDefOutRect(3, 3, 19, 25)
```

10.21 SetOutRect

Sets the **current** output rectangle (= virtual Page Margins) 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 393 property separately.

method void VPE.SetOutRect(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*
)

VpeCoord *Left, Top, Right, Bottom*
    the output rectangle

Remarks:
    This function sets the DefOutRect 394 to the same rectangle automatically.

10.22 StorePos

Stores the coordinates Left, Top, Right, Bottom 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 RestorePos 397).

```
method void VPE.StorePos(
)
```

10.23 RestorePos

Restores the last stored coordinates 396 Left, Top, Right, Bottom from the stack.

```
method void VPE.RestorePos(
)
```

10.24 Rotation

[Not supported by the Community Edition]

Sets / returns the new rotation angle for text 474, images 520 and barcodes 544.
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.

For a detailed explanation see "Rotation of Text, Images and Barcodes" in the Programmer's Manual.

property integer VPE.Rotation

read / write; runtime only
also supported by TVPEObject **(works only for TVPEObjects which reside in a template!)**

Possible Values:
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.
Metafiles can not be rotated.

Example:

```
Doc.Rotation = 900
```

The next inserted objects will be rotated by 90 degrees clockwise.

10.25 Viewable

[Professional Edition and above]

Controls, whether the next inserted object(s) are visible in the preview.

property boolean VPE.Viewable

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	viewable
False	not viewable

Default:

True

10.26 Printable

[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.

property boolean VPE.Printable

read / write; runtime only; also supported by TVPEObject

Possible Values:

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 401 to override this behavior.

10.27 ExportNonPrintableObjects

[Professional Edition and above]

By default, objects which are marked as non-printable 400 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.

property boolean VPE.ExportNonPrintableObjects

read / write; runtime only

Possible Values:

Value	Description
True	yes, export objects which are marked as non-printable
False	no, do not export objects which are marked as non-printable

Default:
False

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.

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 ReadDoc 229("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.

10.28 Streamable

[Professional Edition and above]

Controls, whether the next inserted object(s) are streamed to the VPE Document File.

property boolean VPE.Streamable

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	yes
False	no

Default:
True

10.29 Shadowed

[Professional Edition and above]

Draws a shadow automatically, valid for all rectangular objects.

property boolean VPE.Shadowed

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	yes, draw a shadow
False	no

Default:
True

10.30 ClearPage

[Professional Edition and above]

Deletes all VPE Objects 884 of the current page 366. The current page will be blank and contains no objects after calling this function.

```
method void VPE.ClearPage(
)
```

Remarks:

The preview will refresh automatically after calling this function.

10.31 InsertPage

[Professional Edition and above]

Inserts a new blank page at the position of the current page 366 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 VPE.InsertPage(
)
```

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 884 handles of those text objects by using the property LastInsertedObject 409 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.

10.32 RemovePage

[Professional Edition and above]

Removes the current page 366 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 ClearPage 404.

```
method void VPE.RemovePage(
)
```

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 884 handles of those text objects by using the property LastInsertedObject 409 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.

10.33 InsertAtBottomZOrder

[Professional Edition and above]

Specifies that newly added objects will be inserted at the bottom z-order, i.e. below all previously added objects. Objects with a higher z-order are painted on top of objects with a lower z-order. This property is ideal for creating watermarks.

property boolean VPE.InsertAtBottomZOrder

read / write; runtime only

Possible Values:

Value	Description
True	New objects are added at the bottom of the z-order (below all previously added objects)
False	New objects are added at the top of the z-order (above all previously added objects)

Default:
False

10.34 DeleteObject

[Enterprise Edition and above]

Deletes the given VPE Object 884 from the document. References to VPE Objects can be obtained with the properties LastInsertedObject 409 and FirstObject 410.

```
method void VPE.DeleteObject(
        TVPEObject Object
)
```

TVPEObject Object
VPE Object that shall be deleted

Remarks:
Use this method with caution: if the deleted object was inserted from a dumped template, and you try to access it with InsertedVpeObject() 899, an Access Violation will occur.
As an alternative, you could make the object invisible instead of deleting it, by setting its properties Streamable 402, Viewable 399 and Printable 400 to false.

The preview will not refresh automatically after calling this function. Call Refresh() 218 to do so.

10.35 LastInsertedObject

[Enterprise Edition and above]

Returns the VPE Object 884 of the last object that has been inserted into the VPE Document.

property TVPEObject VPE.LastInsertedObject

read; runtime only

Remarks:

If there is no object available, an exception is thrown.

See also:

"Important Note for VPE-VCL Users" in the Programmer's Manual

10.36 FirstObject

[Enterprise Edition and above]

Returns the first VPE Object 884 of the current page. Each VPE Object itself offers the property NextObject 901 in order to iterate through all VPE Objects of a page.

property TVPEObject VPE.FirstObject

read; runtime only

Remarks:

If there is no object available, the ObjectHandle 888 of the returned object is null.

See also:

"Important Note for VPE-VCL Users" in the Programmer's Manual

Rendering

11 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 474 and pictures 520. 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 792 with Bottom-Line / (Alt-)Subdivider) the center of the pen is used a sbasis for computations. For example if RenderWriteBox() 419 returned as computed position / dimensions (1, 1, 2, 2) and the box of the text object has a PenSize 443 (= 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 at a given height. In most cases you will set width = VFREE and height = VFREE (see "Pictures" in the Programmer's Manual).

See also:

"Rendering Objects" in the Programmer's Manual

11.1 ComputeSingleLineChars

[Not supported by the Community Edition]

Computes the maximum number of characters that fit in a single line of given width (i.e. horizontal space in a document). Only the first line of the supplied text is considered. Any subsequent lines are ignored.

ComputeSingleLineChars() accounts for the current font settings. It also allows for a left and right border line drawn at width given by the current pen size (i.e. the width passed to this method is reduced by the pen width plus a gap to account for these lines, exactly as done by PrintBox() and WriteBox()). Typically, you would set the pen size to zero and thus use the full width for the text (i.e. no border lines drawn).

This method is useful if you wish to truncate a single line of text to fit within a given width (and not have it wrap to a further line). It is also useful for more complex text rendering tasks where there is a need for piecemeal text output. However, as a rule, the other rendering methods described in this chapter are better suited and more efficient at managing the complex layout of whole text objects.

method integer VPE.ComputeSingleLineChars(
  string *Text*,
  VpeCoord *width*,
  VslcMode [integer] *mode*
)

string Text
  the text to be rendered

VpeCoord *width*
  the width for which the number of characters is computed

int *mode*
  the method can be used in two different modes:

ActiveX / VCL	Value	Enum	Comment
VSLC_MODE_WORD	0	ModeWord	Truncate the line at the last complete word that fits in the width.
VSLC_MODE_CHAR	1	ModeChar	Truncate the line at the last character that fits in the width.

Returns:
  The number of characters in the leftmost words that fit completely within the given width (mode = VSLC_MODE_WORD) or the maximum number of leftmost characters that fit completely within the given width (mode = VSLC_MODE_CHAR) based on the currently selected font and pen size.

Remarks:
  This method only works for plain text. It does not support RTF (Rich Text). There is no equivalent method for RTF.

Example:

```
Doc.PenSize = 0
n = Doc.ComputeSingleLineChars("this is a test", 3, VSLC_MODE_WORD)
```

Computes which of the leftmost words in the string "this is a test" fit completely within the horizontal width of 3 cm (without border lines) using the currently selected font, and returns the number of characters (n) included in those words.

See also:

"Rendering Objects" in the Programmer's Manual

11.2 nRenderWidth, nRenderHeight

Returns the width / height of the last rendered object.

property VpeCoord VPE.nRenderWidth
property VpeCoord VPE.nRenderHeight

read; runtime only

Possible Values:
The width / height of the last rendered object.

See also:
"Rendering Objects" in the Programmer's Manual

11.3 RenderPrint

Computes the dimensions of a given text, based on the method Print() 497.

method RenderStatus [integer] VPE.RenderPrint(
VpeCoord *Left*,
VpeCoord *Top*,
string *Text*
)

VpeCoord *Left, Top*
position

string Text
the text to be rendered

Returns:
The method returns one of the following values, indicating the AutoBreak 364 status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:
The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

Example:

```
Doc.RenderPrint(0, 0, "X")
font_height = Doc.nRenderHeight
```

retrieves the font height for a single line

See also:
"Rendering Objects" in the Programmer's Manual

11.4 RenderPrintBox

Computes the dimensions of a given text, based on the method PrintBox() 499. The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

```
method integer VPE.RenderPrintBox(
        VpeCoord Left,
        VpeCoord Top,
        string Text
)
```

VpeCoord *Left, Top*
position

string Text
the text to be rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:
The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

See also:
"Rendering Objects" in the Programmer's Manual

11.5 RenderWrite

Computes the dimensions of a given text, based on the method Write() 697.

method integer VPE.RenderWrite(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
    position and dimensions

string Text
    the text to be rendered

Returns:
    The method returns one of the following values, indicating the AutoBreak status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:
    The computed dimensions can be retrieved with the properties:
    nRenderWidth and nRenderHeight 415

See also:
    "Rendering Objects" in the Programmer's Manual

11.6 RenderWriteBox

Computes the dimensions of a given text, based on the method WriteBox() 496. The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

method integer VPE.RenderWriteBox(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    string *Text*
)

VpeCoord Left, Top, Right, Bottom
position and dimensions

string Text
the text to be rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:
The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

See also:
"Rendering Objects" in the Programmer's Manual

11.7 FontAscent

[Professional Edition and above]

Returns the ascent of the currently selected font. The ascent is the height of a character from the top to the baseline. This is a text metric value.

property VpeCoord **VPE.FontAscent**

Returns:
The ascent of the currently selected font.

11.8 FontDescent

[Professional Edition and above]

Returns the descent of the currently selected font. The descent is the distance from the baseline of a character to the bottom. This is a text metric value.

property VpeCoord **VPE.FontDescent**

Returns:

The descent of the currently selected font.

11.9 GetCharacterHeight

[Professional Edition and above]

Returns for the currently selected font the height of the bounding box of the first character in the provided string.

```
method VpeCoord VPE.GetCharacterHeight(
        string Text
)
```

string Text

The text to be rendered, only the first character of the string is used. Any additional characters are ignored.

Returns:

The method returns for the currently selected font the height of the bounding box of the first character in the provided string.

In case of an error, -1 is returned.

Remarks:

If you provide the capital letter "M" to this method, the returned value is the same like the text metric value "**Cap Height**" defined by font designers.

Using the Ascent, Descent and Character Height, the VPE API provides ways to align text at the Cap Height, the baseline or the descent of a selected font.

Example:

The following example draws lines at text metric positions of some text. Additionally the add-on text "Test" is drawn at the cap height position of the main text.

```
VpeCoord left = 2;
VpeCoord top = 2;
VpeCoord right = 18;

doc.SetFont("Arial", 72);
VpeCoord ascent = doc.FontAscent;
VpeCoord descent = doc.FontDescent;
VpeCoord cap_height = doc.CharacterHeight("M");

// top
doc.Line(left, top, right, top);

// cap height position
VpeCoord cap_pos = ascent - cap_height;
doc.PenColor = COLOR_RED;
doc.Line(left, top + cap_pos, right, top + cap_pos);

// baseline
VpeCoord baseline = top + ascent;
doc.PenColor = COLOR_BLUE;
doc.Line(left, baseline, right, baseline);

// bottom
doc.Line(left, top + ascent + descent, right, top + ascent + descent);

// text
doc.Print(left, top, "Üg My Text.");
VpeCoord x = doc.nRight;

// add-on text, positioned at cap height of main text
doc.FontSize = 26;
VpeCoord SmallCapHeight = doc.GetCharacterHeight("M");
doc.Print(x, baseline - cap_height - (doc.FontAscent -
SmallCapHeight), "Test");
```

11.10 FontInternalLeading

[Professional Edition and above]

Returns the internal leading of the currently selected font. This value is a text metric value. It is normally not of interest and only provided for completeness of the VPE API.

property VpeCoord **VPE.FontInternalLeading**

Returns:

The internal leading of the currently selected font.

11.11 FontExternalLeading

[Professional Edition and above]

Returns the external leading of the currently selected font. This value is a text metric value. It is normally not of interest and only provided for completeness of the VPE API.

property VpeCoord **VPE.FontExternalLeading**

Returns:
The external leading of the currently selected font.

11.12 RenderPicture

Computes the dimensions of a given image, based on the method Picture() 537. The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

If the property PictureCache 526 is true, the rendered image will automatically be stored in the image cache. We recommend to set PictureCache = true always.

method void VPE.RenderPicture(
VpeCoord *Width*,
VpeCoord *Height*,
string *FileName*
)

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.

string FileName
the image file to be rendered

Remarks:
In case of an error, LastError 201 is set.

The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

Example:

ActiveX / VCL:
```
Doc.PictureCache = True
Doc.RenderPicture(VFREE, VFREE, "image.bmp")
xsize = Doc.nRenderWidth
ysize = Doc.nRenderHeight
```

.NET:
```
Doc.PictureCache = True
Doc.RenderPicture(Doc.nFree, Doc.nFree, "image.bmp")
xsize = Doc.nRenderWidth
ysize = Doc.nRenderHeight
```
Computes the width and height of the image stored in "image.bmp". By setting PictureCache 526 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.

ActiveX / VCL:
```
Doc.PictureKeepAspect = True
Doc.RenderPicture(VFREE, 5, "image.bmp")
xsize = Doc.nRenderWidth
```

.NET:

```
Doc.PictureKeepAspect = True
Doc.RenderPicture(Doc.nFree, 5, "image.bmp")
xsize = Doc.nRenderWidth
```

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

11.13 RenderPictureStream

[Professional Edition and above]

Identical to RenderPicture() 426, but renders a picture 520 from a memory stream.

```
method void VPE.RenderPictureStream(
        TVPEStream stream,
        VpeCoord Width,
        VpeCoord Height,
        string Identifier
)
```

TVPEStream stream
The stream-object where the picture is read from. The stream must have been created before by calling CreateMemoryStream() 694, 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.

string 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

11.14 RenderPictureResID

[Not available in .NET, use RenderPictureDIB 431]

Computes the dimensions of a given image, based on the method PictureResID() 540. The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

```
method void VPE.RenderPictureResID(
        VpeCoord Width,
        VpeCoord Height,
        long hInstance,
        long ResourceID
)
```

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.

long hInstance
  Instance-Handle (of type HINSTANCE)

long ResourceID
  the resource id

Remarks:
  In case of an error, LastError 201 is set.

  The computed dimensions can be retrieved with the properties:
  nRenderWidth and nRenderHeight 415

Example:
  see RenderPicture 426

See also:
  "Rendering Objects" in the Programmer's Manual

11.15 RenderPictureResName

[Not available in .NET, use RenderPictureDIB 431]

Computes the dimensions of a given image, based on the method PictureResName() 541.
The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

method void VPE.RenderPictureResName(
    VpeCoord *Width*,
    VpeCoord *Height*,
    long *hInstance*,
    string *ResourceName*
)

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.

long hInstance
    Instance-Handle (of type HINSTANCE)

string ResourceName
    the resource name

Remarks:
    In case of an error, LastError 201 is set.

    The computed dimensions can be retrieved with the properties:
    nRenderWidth and nRenderHeight 415

Example:
    see RenderPicture 426

See also:
    "Rendering Objects" in the Programmer's Manual

11.16 RenderPictureDIB

Computes the dimensions of a given image, based on the method PictureDIB() 542. The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

method void VPE.RenderPictureDIB(
VpeCoord *Width*,
VpeCoord *Height*,
long *hDIB*
)

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.

long hDIB
handle (of type HGLOBAL) to the DIB to be rendered

Remarks:
In case of an error, LastError 201 is set.

The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

Example:
see RenderPicture 426

See also:
"Rendering Objects" in the Programmer's Manual

11.17 RenderRTF

[Professional Edition and above]

Computes the dimensions of a given RTF text, based on the method WriteRTF() 613.

```
method integer VPE.RenderRTF(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string Text
)
```

VpeCoord *Left, Top, Right, Bottom*
position and dimensions, y2 may be set to VFREE

string Text
the RTF text to be rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:
VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

In case of an error, LastError 201 is set.

The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

See also:
"Rendering Objects" in the Programmer's Manual

11.18 RenderBoxRTF

[Professional Edition and above]

Computes the dimensions of a given RTF text, based on the method WriteBoxRTF() 615. The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

```
method integer VPE.RenderBoxRTF(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string Text
)
```

VpeCoord *Left, Top, Right, Bottom*
position and dimensions, y2 may be set to VFREE

string Text
the RTF text to be rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:
VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

In case of an error, LastError 201 is set.

The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

See also:
"Rendering Objects" in the Programmer's Manual

11.19 RenderRTFFile

[Professional Edition and above]

Computes the dimensions of a given RTF text file, based on the method WriteRTFFile() 617
.

```
method integer VPE.RenderRTFFile(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string Text
)
```

VpeCoord *Left, Top, Right, Bottom*
position and dimensions, y2 may be set to VFREE

string FileName
the file with RTF text that is rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:
VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

In case of an error, LastError 201 is set.

The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

See also:
"Rendering Objects" in the Programmer's Manual

11.20 RenderBoxRTFFile

[Professional Edition and above]

Computes the dimensions of a given RTF text file, based on the method WriteBoxRTFFile() 617. The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

method integer VPE.RenderBoxRTFFile(
VpeCoord *Left*,
VpeCoord *Top*,
VpeCoord *Right*,
VpeCoord *Bottom*,
string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
position and dimensions, y2 may be set to VFREE

string FileName
the file with RTF text that is rendered

Returns:
The method returns one of the following values, indicating the AutoBreak status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:
VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

In case of an error, LastError 201 is set.

The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

See also:
"Rendering Objects" in the Programmer's Manual

11.21 RenderRTFStream

[Professional Edition and above]

Computes the dimensions of a given RTF stream, based on the method WriteRTFFile() 617.

```
method integer VPE.RenderRTFStream(
        TVPEStream Stream,
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom
)
```

TVPEStream Stream
  Stream Object

VpeCoord *Left, Top, Right, Bottom*
  position and dimensions, y2 may be set to VFREE

Returns:
  The method returns one of the following values, indicating the AutoBreak status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:
  VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

  In case of an error, LastError 201 is set.

  The computed dimensions can be retrieved with the properties:
  nRenderWidth and nRenderHeight 415

See also:
  "Rendering Objects" in the Programmer's Manual

11.22 RenderBoxRTFStream

[Professional Edition and above]

Computes the dimensions of a given RTF stream, based on the method WriteBoxRTFFile() 617. The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

```
method integer VPE.RenderBoxRTFStream(
        TVPEStream Stream,
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom
)
```

TVPEStream Stream
  Stream Object

VpeCoord *Left, Top, Right, Bottom*
  position and dimensions, y2 may be set to VFREE

Returns:

The method returns one of the following values, indicating the AutoBreak status:

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur
RENDER_BREAK	1	PageBreak	Auto Page Break will occur
RENDER_SKIP_BREAK	2	SkipPageBreak	Auto Page Break will occur, but no text will be placed on the current page, all text will be skipped to the next page (nRenderWidth and nRenderHeight are not set)

Remarks:

VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

In case of an error, LastError 201 is set.

The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

See also:

"Rendering Objects" in the Programmer's Manual

11.23 RenderFormField

[Enterprise Edition and above]

Computes the dimensions of a given FormField text, based on the method FormField 793. The dimensions are computed including the surrounding frame, if PenSize 443 is <> 0.

```
int VPE.RenderFormField(
        VpeCoord x,
        VpeCoord y,
        VpeCoord x2,
        VpeCoord y2,
        LPCSTR s
)
```

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

ActiveX / VCL	Value	Enum	Comment
RENDER_NO_BREAK	0	NoPageBreak	NO Auto Page Break will occur

Remarks:
If x2 = VFREE, the FormField will behave the same as if VpeWrite 495 (Box) was used.

The computed dimensions can be retrieved with the properties:
nRenderWidth and nRenderHeight 415

See also:
"Rendering Objects" in the Programmer's Manual

Drawing Functions

12 Drawing Functions

This section deals with the basic drawing functions, e.g. lines 446, polylines 448, polygons 467, boxes 466, ellipses 470, etc.; and the basic style settings for these objects, which will inherit to the text functions 474 in the next section.

12.1 SetPen

Sets the style of the pen - all properties at once. All drawing objects that are inherited from the pen-object will use the pen (see "The Object-Oriented Style" in the Programmer's Manual for details). The pen can be made invisible by setting the PenSize to 0.

You can use the psXyz pen styles from Windows GDI, but pen styles other than psSolid 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 psSolid until the GDI changes.

```
method void VPE.SetPen(
        VpeCoord Size,
        PenStyle [integer] Style,
        Color Color
)
```

also supported by TVPEObject

VpeCoord *Size*
the thickness of the pen

PenStyle [integer] Style
possible values are:

ActiveX / VCL	Value	Enum	Comment
psSolid	0	Solid	
psDash	1	Dash	-------
psDot	2	Dot	....
psDashDot	3	DashDot	_._._._
psDashDotDot	4	DashDotDot	_.._.._

Color Color
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
0.03 (which is 0.3 mm), psSolid, COLOR_BLACK

12.2 NoPen

Hides the pen (the property PenSize 443 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.

```
method void VPE.NoPen(
)
```

also supported by TVPEObject

12.3 PenSize

Returns / sets the pen size.

property VpeCoord **VPE.PenSize**

read / write; runtime only; also supported by TVPEObject

Possible Values:
the thickness of the pen

Default:
0.03 (which is 0.3 mm)

Example:

```
Doc.PenSize = 0.06    // 0.6 mm
```

12.4 PenStyle

Sets the style for the pen. You can use the psXyz pen styles from Windows GDI, but pen styles other than psSolid 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 psSolid until the GDI changes.

property PenStyle [integer] VPE.PenStyle

read / write; runtime only; also supported by TVPEObject

Possible Values:
one of the windows pen styles; possible values are:

ActiveX / VCL	Value	Enum	Comment
psSolid	0	Solid	
psDash	1	Dash	// -------
psDot	2	Dot	// ....
psDashDot	3	DashDot	// _._._._
psDashDotDot	4	DashDotDot	// _.._.._

Default:
psSolid

12.5 PenColor

Sets the color of the pen.

property Color VPE.PenColor

read / write; runtime only; also supported by TVPEObject

Possible Values:

Color Color

any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:

COLOR_BLACK

12.6 Line

Draws a line with the current pen from the coordinate (Left, Top) to the coordinate (Right, Bottom).

```
method void VPE.Line(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom
)
```

VpeCoord *Left, Top*
starting coordinates

VpeCoord *Right, Bottom*
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 520 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.

12.7 VpeLine

[ActiveX only]

The same as Line() 446. This method is for the use with Visual Basic. Visual Basic has problems with the keywords "Print, Write, Line and Scale". VB doesn't recognize, that these are methods and properties of an ActiveX.

Draws a line with the current pen from the coordinate (Left, Top) to the coordinate (Right, Bottom).

method void VPE.VpeLine(
VpeCoord *Left*,
VpeCoord *Top*,
VpeCoord *Right*,
VpeCoord *Bottom*
)

VpeCoord *Left, Top*
starting coordinates

VpeCoord *Right, Bottom*
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 520 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.

12.8 PolyLine

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 AddPolyPoint() 449 (.NET: AddPoint 450).

```
method TVPEPolyLine [pointer] VPE.PolyLine(
        long Count
)
```

long Count
maximum number of points (coordinates) that can be stored in this object. One point is a pair of one x and one y value.

Returns:

VCL	a handle to the object, which can be used in further calls to AddPolyPoint()
ActiveX, .NET, Java, ...	a TVPEPolyLine object, which offers the method AddPoint().

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 Line() 446.

There is also an "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

12.9 AddPolyPoint

[VCL only, for all other component types see the method TVPEPolyLine.AddPoint 450]

Adds a new point to the PolyLine 448 object specified in hPolyLine.

```
method void VPE.AddPolyPoint(
        pointer hPolyLine,
        VpeCoord X,
        VpeCoord Y
)
```

pointer 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

12.10 TVPEPolyLine.AddPoint

[All component types, except VCL]

Adds a new point to the PolyLine 448 object.

```
method void VPE.TVPEPolyLine.AddPoint(
        VpeCoord X,
        VpeCoord Y
)
```

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

12.11 BkgMode

Sets the background mode. The background is the area inside of an object.

property BkgMode [integer] VPE.BkgMode

read / write; runtime only; also supported by TVPEObject

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VBKG_SOLID	0	Solid	solid background color
VBKG_TRANSPARENT	1	Transparent	transparent background
VBKG_GRD_LINE	2	GradientLine	line gradient background
VBKG_GRD_RECT	3	GradientRectangle	rectangular gradient background
VBKG_GRD_ELLIPSE	4	GradientEllipse	elliptic gradient background

Default:
VBKG_TRANSPARENT

Community Edition:

The Community Edition only supports VBKG_SOLID and VBKG_TRANSPARENT.

12.12 BkgColor

Sets / returns the background color. The background color is the color inside of an object. To make the background color painted, the Background Mode 451 must be VBKG_SOLID.

property Color VPE.BkgColor

read / write; runtime only; also supported by TVPEObject

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_WHITE (but the Transparent Background Mode is activated!)

12.13 BkgGradientStartColor

[Not supported by the Community Edition]

Specifies the gradient start color. To make the gradient painted, the Background Mode 451 must be set to a gradient mode.

property Color VPE.BkgGradientStartColor

read / write; runtime only; also supported by TVPEObject

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_WHITE (but the Transparent Background Mode is activated!)

Remarks:
Gradients do only work for rectangular objects like Boxes 466, Text, Ellipses etc. Gradients are not drawn in Pies 471 and Polygons. If a gradient mode is selected, hatching for the same object is not possible.

Example:

ActiveX / VCL:
```
Doc.BkgMode = VBKG_GRD_LINE
Doc.BkgGradientStartColor = COLOR_BLUE
Doc.BkgGradientEndColor = COLOR_GREEN
Doc.PrintBox(1, 1, "Hello World!")
```

.NET:
```
Doc.BkgMode = BkgMode.GradientLine
Doc.BkgGradientStartColor = Color.Blue
Doc.BkgGradientEndColor = Color.Green
Doc.PrintBox(1, 1, "Hello World!")
```

Will draw the text "Hello World!" with a gradient running from blue to green in the background.

12.14 BkgGradientEndColor

[Not supported by the Community Edition]

Specifies the gradient end color. To make the gradient painted, the Background Mode 451 must be set to a gradient mode.

property Color VPE.BkgGradientEndColor

read / write; runtime only; also supported by TVPEObject

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_BLACK (but the Transparent Background Mode is activated!)

Remarks:
Gradients do only work for rectangular objects like Boxes 466, Text, Ellipses etc. Gradients are not drawn in Pies 471 and Polygons. If a gradient mode is selected, hatching for the same object is not possible.

Example:

ActiveX / VCL:
```
Doc.BkgMode = VBKG_GRD_LINE
Doc.BkgGradientStartColor = COLOR_BLUE
Doc.BkgGradientEndColor = COLOR_GREEN
Doc.PrintBox(1, 1, "Hello World!")
```

.NET:
```
Doc.BkgMode = BkgMode.GradientLine
Doc.BkgGradientStartColor = Color.Blue
Doc.BkgGradientEndColor = Color.Green
Doc.PrintBox(1, 1, "Hello World!")
```

Will draw the text "Hello World!" with a gradient running from blue to green in the background.

12.15 BkgGradientTriColor

[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.

property bool VPE.BkgGradientTriColor

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	On
False	Off

Default:
False

Remarks:
Tri-color gradients can only be used, when the Background Mode 451 is set to VBKG_GRD_LINE.

12.16 BkgGradientMiddleColorPosition

[Not supported by the Community Edition]

Specifies the position of the middle color for tri-color gradients.

property int VPE.BkgGradientMiddleColorPosition

read / write; runtime only; also supported by TVPEObject

Possible Values:

the position in percent of the object’s height (or width, if the gradient is rotated)

Default:

50

Remarks:

Tri-color gradients can only be used, when the Background Mode 451 is set to VBKG_GRD_LINE.

Example:

ActiveX / VCL:

```
Doc.BkgGradientTriColor = True
Doc.BkgMode = VBKG_GRD_LINE
Doc.BkgGradientMiddleColorPosition = 35
Doc.BkgGradientStartColor = COLOR_LTGRAY
Doc.BkgGradientMiddleColor = COLOR_WHITE
Doc.BkgGradientEndColor = COLOR_DKGRAY
Doc.Print(1, 1, "Hello World!")
```

.NET:

```
Doc.BkgGradientTriColor = True
Doc.BkgMode = BkgMode.GradientLine
Doc.BkgGradientMiddleColorPosition = 35
Doc.BkgGradientStartColor = Color.LightGray
Doc.BkgGradientMiddleColor = Color.White
Doc.BkgGradientEndColor = Color.DarkGray
Doc.Print(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.

12.17 BkgGradientMiddleColor

[Not supported by the Community Edition]

Specifies the gradient middle color for tri-color gradients. To make the gradient painted, the Background Mode 451 must be set to VBKG_GRD_LINE and the Tri Color Mode 455 must be activated.

property Color VPE.BkgGradientMiddleColor

read / write; runtime only; also supported by TVPEObject

Possible Values:

any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

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 466, Text, Ellipses etc. Gradients are not drawn in Pies 471 and Polygons. If a gradient mode is selected, hatching for the same object is not possible.

Example:

ActiveX / VCL:

```
Doc.BkgGradientTriColor = True
Doc.BkgMode = VBKG_GRD_LINE
Doc.BkgGradientMiddleColorPosition = 35
Doc.BkgGradientStartColor = COLOR_LTGRAY
Doc.BkgGradientMiddleColor = COLOR_WHITE
Doc.BkgGradientEndColor = COLOR_DKGRAY
Doc.Print(1, 1, "Hello World!")
```

.NET:

```
Doc.BkgGradientTriColor = True
Doc.BkgMode = BkgMode.GradientLine
Doc.BkgGradientMiddleColorPosition = 35
Doc.BkgGradientStartColor = Color.LightGray
Doc.BkgGradientMiddleColor = Color.White
Doc.BkgGradientEndColor = Color.DarkGray
Doc.Print(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.

12.18 BkgGradientRotation

[Not supported by the Community Edition]

Specifies the rotation for the line gradient.

property integer VPE.BkgGradientRotation

read / write; runtime only; also supported by TVPEObject

Possible Values:

rotation angle clockwise in 0.1 degrees
possible values are: 0, 900, 1800, 2700

Default:

0 degrees

Example:

ActiveX / VCL:

```
Doc.BkgGradientRotation = 900
Doc.BkgMode = VBKG_GRD_LINE
Doc.BkgGradientStartColor = COLOR_BLUE
Doc.BkgGradientEndColor = COLOR_GREEN
Doc.PrintBox(1, 1, "Hello World!")
```

.NET:

```
Doc.BkgGradientRotation = 900
Doc.BkgMode = BkgMode.GradientLine
Doc.BkgGradientStartColor = Color.Blue
Doc.BkgGradientEndColor = Color.Green
Doc.PrintBox(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.

12.19 BkgGradientPrint

[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.

property integer VPE.BkgGradientPrint

write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VGRD_PRINT_AUTO	0	Auto	if printer is a color printer, the gradient is printed, otherwise the alternate solid color is used
VGRD_PRINT_GRADIENT	1	Gradient	the gradient is always printed
VGRD_PRINT_SOLID	2	Solid	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:

ActiveX / VCL:

```
Doc.BkgGradientPrint = VGRD_PRINT_SOLID
Doc.BkgGradientPrintSolidColor = COLOR_LTGRAY
```

.NET:

```
Doc.BkgGradientPrint = BkgGradientPrint.Solid
Doc.BkgGradientPrintSolidColor = Color.LightGray
```

Will force VPE to draw a very light grey color in the background of gradient objects when they are printed.

ActiveX / VCL:

```
Doc.BkgGradientPrint = VGRD_PRINT_AUTO
Doc.BkgGradientPrintSolidColor = Color.LightGray
```

.NET:

```
Doc.BkgGradientPrint = BkgGradientPrint.Auto
Doc.BkgGradientPrintSolidColor = Color.LightGray
```

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.

12.20 BkgGradientPrintSolidColor

[Not supported by the Community Edition]

Specifies the alternative solid color, which shall be drawn depending on the setting for BkgGradientPrint 459.

property Color VPE.BkgGradientPrintSolidColor

write; runtime only

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_LTGRAY

Remarks:
In contrast to all other graphical object properties, this property is valid for ALL objects in the document at once.

12.21 TransparentMode

Sets / returns the current background mode (transparent / not transparent).
The background is the area inside of an object. You can use BkgMode 451 also.

property boolean VPE.TransparentMode

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	on (transparent)
False	off (not transparent)

Default:
True

12.22 HatchStyle

[Not supported by the Community Edition]

Sets / returns the current hatch style. All rectangular objects - except charts 710, barcodes 544 and images - can be hatched with the predefined windows hatch styles.

property long VPE.HatchStyle

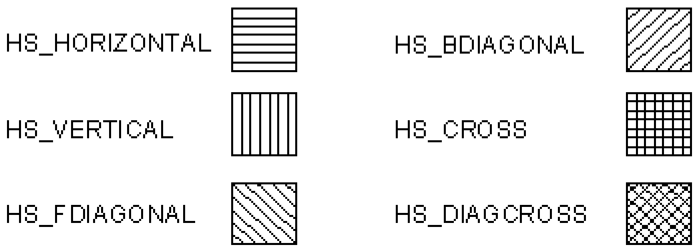
read / write; runtime only; also supported by TVPEObject

Possible Values:

ActiveX / VCL	Value	Enum	Comment
hsNone	-1	None	see Possible styles below
hsHorizontal	0	Horizontal	see Possible styles below
hsVertical	1	Vertical	see Possible styles below
hsFDiagonal	2	ForwardDiagonal	see Possible styles below
hsBDiagonal	3	BackwardDiagonal	see Possible styles below
hsCross	4	Cross	see Possible styles below
hsDiagCross	5	DiagonalCross	see Possible styles below

Default:
hsNone, which means NO hatching

Possible styles are:



12.23 HatchColor

[Not supported by the Community Edition]

Sets / returns the color of the hatching.

property Color VPE.HatchColor

read / write; runtime only; also supported by TVPEObject

Possible Values:

any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:

COLOR_BLACK

12.24 CornerRadius

[Not supported by the Community Edition]

Sets / returns 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.

property VpeCoord VPE.CornerRadius

read / write; runtime only

Possible Values:
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.

12.25 Box

Draws a box object at position Left, Top with the right border at Right and the bottom border at Bottom. The current pen style, background mode and background color are used.

```
method void VPE.Box(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom
)
```

VpeCoord *Left, Top, Right, Bottom*
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 520 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.

12.26 Polygon

Creates a Polygon object using the current pen, background and hatchstyle 463. After a call to this method, the created Polygon object is ready for use with the method AddPolygonPoint() 468 (.NET: AddPoint 469). See also: "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

```
method TVPEPolygon [pointer] VPE.Polygon(
        long Count
)
```

long Count

maximum number of points (coordinates) that can be stored in this object. One point is a pair of one x and one y value.

Returns:

VCL	a handle to the object, which can be used in further calls to AddPolygonPoint() 468.
all other Control-Types	a TVPEPolygon object, which offers the method AddPoint()

Remarks:

There is also an "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

12.27 AddPolygonPoint

[VCL only, for all other component types see the method TVPEPolygon.AddPoint 469]

Adds a new point to the Polygon 467 object specified by the handle *hPolygon*.

```
method void VPE.AddPolygonPoint(
        pointer hPolygon,
        VpeCoord X,
        VpeCoord Y
)
```

pointer hPolygon
Polygon 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

12.28 TVPEPolygon.AddPoint

[All component types, except VCL]

Adds a new point to the Polygon 467 object.

```
method void VPE.TVPEPolygon.AddPoint(
        VpeCoord X,
        VpeCoord Y
)
```

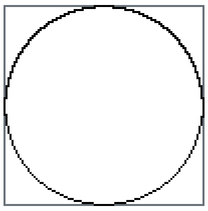
VpeCoord X, Y
coordinate of new point

- 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

12.29 Ellipse

Draws an ellipse or circle within the given rectangle.

Left, Top



Right, Bottom

```
method void VPE.Ellipse(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom
)
```

VpeCoord *Left, Top, Right, Bottom*
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 520 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 "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

12.30 Pie

This function is identical to Ellipse() 470, but it draws a pie from StartAngle to EndAngle.

```
method void VPE.Pie(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        integer StartAngle,
        integer EndAngle
)
```

long Left, long Top, long Right, long Bottom
surrounding rectangle of the pie

integer BeginAngle
angle in 0.1 degrees, clockwise

integer EndAngle
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 520 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 "Important Note About Pens, Lines, Frames, Circles and Ellipses" in the Programmer's Manual.

This page is intentionally left blank.

Text Functions

13 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".

13.1 SetFont

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.

method void VPE.SetFont(
    string *Name*,
    integer *Size*
)

also supported by TVPEObject

string 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".

13.2 SelectFont

The same as SetFont() 475. 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.

method void VPE.SelectFont(
  string *Name*,
  integer *Size*
)

also supported by TVPEObject

string 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".

13.3 FontName

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.

property string VPE.FontName

read / write; runtime only; also supported by TVPEObject

Possible Values:

font name (e.g. "Times New Roman")

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".

13.4 FontSize

Sets a font size.

property long VPE.FontSize

read / write; runtime only; also supported by TVPEObject

Possible Values:
font size in points (not in metric or inch units!)

Default:
10 pt

13.5 SetFontSubstitution

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.

method void VPE.SetFontSubstitution(
string *OriginalFont,*
string *SubstFont*
)

string *OriginalFont*
the font which shall be subtituted by the SubstFont

string SubstFont
the font which shall be used in place of the OriginalFont

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 PurgeFontSubstitution() 480.

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 SetFontControl() 942 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:

```
Doc.SetFontSubstitution("Arial", "Helvetica")
```

Substitutes the True-Type font "Arial" with the Base 14 font "Helvetica", so Helvetica is used in place of Arial.

13.6 PurgeFontSubstitution

Resets all font substitution settings at once. This clears all font substitution rules, so no substitution takes place.

method void VPE.PurgeFontSubstitution()

13.7 CharSet

Sets the charset.

property CharSet [long] VPE.CharSet

read / write; runtime only; also supported by TVPEObject

Possible Values:

character set, possible values are:

ActiveX / VCL	Value	Enum	Comment
VCHARSET_DEFAULT	1	Default	Windows: Chooses 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	Symbol	Required for using Symbol Fonts, like WingDings
VCHARSET_MAC_ROMAN	77	MacRoman	Macintosh Roman
Character sets compatible to the character sets of the Windows operating system:			
VCHARSET_WIN_ANSI	0	WinAnsi	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	WinHebrew	Hebrew
VCHARSET_WIN_ARABIC	178	WinArabic	Arabic
VCHARSET_WIN_GREEK	161	WinGreek	Greek
VCHARSET_WIN_TURKISH	162	WinTurkish	Turkish
VCHARSET_WIN_VIETNAMESE	163	WinVietnamese	Vietnamese
VCHARSET_WIN_THAI	222	WinThai	Thai
VCHARSET_WIN_EAST_EUROPE	238	WinEastEurope	Albanian, Belarusian, Croatian, Czech, Hungarian, Polish, Romanian, Serbian, Slovak, Slovenian
VCHARSET_WIN_CYRILLIC	204	WinCyrillic	Bulgarian, Russian, Serbian and Ukrainian (Cyrillic)
VCHARSET_WIN_BALTIC	186	WinBaltic	Estonian, Latvian and Lithuanian
Character sets compatible to the ISO character sets (taken from unicode.org):			

VCHARSET_ISO_LATIN_1	50	IsoLatin1	Latin-1, Western (ISO-8859-1)
VCHARSET_ISO_LATIN_2	51	IsoLatin2	Latin-2, East European (ISO-8859-2)
VCHARSET_ISO_LATIN_3	52	IsoLatin3	Latin-3, South European (ISO-8859-3)
VCHARSET_ISO_LATIN_4	53	IsoLatin4	Latin-4, Baltic (ISO-8859-4)
VCHARSET_ISO_CYRILLIC	54	IsoCyrillic	Cyrillic (ISO-8859-5)
VCHARSET_ISO_ARABIC	55	IsoArabic	Arabic (ISO-8859-6)
VCHARSET_ISO_GREEK	56	IsoGreek	Greek (ISO-8859-7)
VCHARSET_ISO_HEBREW	57	IsoHebrew	Hebrew (ISO-8859-8)
VCHARSET_ISO_LATIN_5	58	IsoLatin5	Latin-5, Turkish (ISO-8859-9)
VCHARSET_ISO_LATIN_6	59	IsoLatin6	Latin-6, Nordic (ISO-8859-10)
VCHARSET_ISO_THAI	60	isoThai	Thai (ISO-8859-11)
VCHARSET_ISO_LATIN_7	62	IsoLatin7	Latin-7, Baltic (ISO-8859-13)
VCHARSET_ISO_LATIN_8	63	IsoLatin8	Latin-8, Celtic (ISO-8859-14)
VCHARSET_ISO_LATIN_9	64	IsoLatin9	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 VCHARSET_ SYMBOL. VPE switches automatically the Charset to Symbol, if a Symbol font is selected. Vice versa, VPE switches back to the previously used Charset, if a non-Symbol font is selected.

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.

13.8 SetFontAttr

Sets all font-attributes at once.

method void VPE.SetFontAttr(
TextAlignment [integer] *Alignment*,
boolean *Bold*,
boolean *Underlined*,
boolean *Italic*,
boolean *Strikeout*
)

also supported by TVPEObject

TextAlignment [integer] Alignment
possible values are:

ActiveX / VCL	Value	Enum	Comment
ALIGN_LEFT	0	Left	
ALIGN_RIGHT	1	Right	
ALIGN_CENTER	2	Center	
ALIGN_JUSTIFIED	3	Justified	
ALIGN_JUSTIFIED_AB	5	JustifiedAb	

boolean Bold

Value	Description
True	bold
False	not bold

boolean Underlined

Value	Description
True	underlined
False	not underlined

boolean Italic

Value	Description
True	italic
False	not italic

boolean 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.

13.9 TextAlignment

Sets the text alignment

property integer VPE.TextAlignment

read / write; runtime only; also supported by TVPEObject

Possible Values:

ActiveX / VCL	Value	Enum	Comment
ALIGN_LEFT	0	Left	
ALIGN_RIGHT	1	Right	
ALIGN_CENTER	2	Center	
ALIGN_JUSTIFIED	3	Justified	
ALIGN_JUSTIFIED_AB	5	JustifiedAb	

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.

13.10 TextBold

Sets text bold on / off

property boolean VPE.TextBold

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	bold
False	not bold

Default:

False

13.11 TextUnderline

Sets text underlined on / off

property boolean VPE.TextUnderline

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	underlined
False	not underlined

Default:
False

13.12 TextUnderlined

The same as TextUnderline 488. Sets text underlined on / off

property boolean VPE.TextUnderlined

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	underlined
False	not underlined

Default:

False

13.13 TextItalic

Sets text italic on / off

property boolean VPE.TextItalic

read / write; runtime only; also supported by TVPEObject

Possible Values:

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.

13.14 TextStrikeOut

Sets text strikeout on / off

property boolean VPE.TextStrikeOut

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	strikeout
False	not strikeout

Default:

False

13.15 TextColor

Sets the text color. This is the foreground color which also applies to Barcodes 544, RTF 612 and Charts 710.

property Color VPE.TextColor

read / write; runtime only; also supported by TVPEObject

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_BLACK

13.16 Write

Outputs text formatted with the current alignment settings within a rectangle at position Left, Top with the right border at Right and the bottom border at Bottom.

The pen is invisible.

```
method VpeCoord VPE.Write(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string Text
)
```

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

string 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 418 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) 612.

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 520 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.

See also:

VpeWrite 495

13.17 VpeWrite

[ActiveX only]

The same as Write() 697. This method is for the use with Visual Basic. Visual Basic has problems with the keywords "Print, Write, Line and Scale". VB doesn't recognize, that these are methods and properties of an ActiveX.

Outputs text formatted with the current alignment settings within a rectangle at position Left, Top with the right border at Right and the bottom border at Bottom.

The pen is invisible.

method VpeCoord VPE.VpeWrite(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
    position and dimensions

string Text
    the text to output

Returns:
    the bottom y-coordinate generated by the output

Remarks:
    see also Write 697

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) 612.

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 520 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.

13.18 WriteBox

The same as Write() 697, but pen 441- and box 466-settings are used. Outputs text formatted with the current alignment settings within a rectangle at position Left, Top with the right border at Right and the bottom border at Bottom.

method VpeCoord VPE.WriteBox(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

string Text
the text to output

Returns:
the bottom y-coordinate generated by the output

Remarks:
see also Write 697

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) 612.

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 520 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.

13.19 Print

The same as Write() 697 with the parameters (Left, Top, VFREE, VFREE).

The pen is invisible.

If the right border of the page (nRightMargin 393, 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 Write() 697 or WriteBox() 496 not using VFREE.

```
method VpeCoord VPE.Print(
        VpeCoord Left,
        VpeCoord Top,
        string Text
)
```

VpeCoord *Left, Top*
position

string Text
the text to output

Returns:
the bottom y-coordinate generated by the output

Remarks:
see also Write 697

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) 612.

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 520 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.

See also:
VpePrint 498

13.20 VpePrint

[ActiveX only]

The same as Print() 497. This method is for the use with Visual Basic. Visual Basic has problems with the keywords "Print, Write, Line and Scale". VB doesn't recognize, that these are methods and properties of an ActiveX.

The method works like Write() 697 with the parameters (Left, Top, VFREE, VFREE).

The pen is invisible.

If the right border of the page (nRightMargin 393, 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 Write() 697 or WriteBox() 496 not using VFREE.

method VpeCoord VPE.VpePrint(
  VpeCoord *Left*,
  VpeCoord *Top*,
  string *Text*
)

VpeCoord *Left, Top*
 position

string Text
 the text to output

Returns:
 the bottom y-coordinate generated by the output

Remarks:
 see also Write 697

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) 612.

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 520 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.

13.21 PrintBox

The same as WriteBox() 496 with the parameters (Left, Top, VFREE, VFREE).

If the right border of the page (nRightMargin 393, 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 Write() 697 or WriteBox() 496 not using VFREE.

method VpeCoord VPE.PrintBox(
        VpeCoord *Left*,
        VpeCoord *Top*,
        string *Text*
)

VpeCoord *Left, Top*
    position

string Text
    the text to output

Returns:
    the bottom y-coordinate generated by the output

Remarks:
    see also Write 697

    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) 612.

    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 520 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.

13.22 EmbeddedFlagParser

[Not supported by the Community Edition]

Using this property, you can control whether the Embedded Flag Parser is active. When you assign the value *false* to this property, the Embedded Flag Parser is turned off, so text beginning with a "[" will not be interpreted as embedded flags. Instead the "[" and all following text will be inserted into the document.

property boolean VPE.EmbeddedFlagParser

read / write; runtime only

Possible Values:

Value	Description
True	The Embedded Flag Parser is active.
False	The Embedded Flag Parser is turned off.

Default:

True = The Embedded Flag Parser is active

13.23 DefineHeader

Exactly the same as WriteBox() 496, 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.

```
method void VPE.DefineHeader(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string Text
)
```

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

string Text
the header text to output

Remarks:
After the header has been defined once, it can not be redefined nor deleted.

13.24 DefineFooter

Exactly the same as WriteBox() 496, 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.

```
method void VPE.DefineFooter(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string Text
)
```

VpeCoord *Left, Top, Right, Bottom*
    position and dimensions

string Text
    the footer text to output

Remarks:
    After the footer has been defined once, it can not be redefined nor deleted.

13.25 CharPlacement

[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 for each letter. The provided offset is understood as a "character cell width", VPE will print each character centered within this "cell".

property VpeCoord **VPE.CharPlacement**

write; runtime only

Possible Values:
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:

```
Doc.CharPlacement = 0.5
Doc.Print(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.

This page is intentionally left blank.

Text Block Object

14 Text Block Object

[Not supported by the Community Edition]

The standard text output functions like Print() and Write() are in regular sufficient to create complex and rich documents. For special use cases VPE provides a Text Block Object, which provides more complex text layout functionality.

The Text Block Object allows to split a large block of text into smaller parts, so small pieces of the whole continuous text can be rendered and spread in any position at any width with any number of lines within a document.

As a result VPE can perform multi-column layout, text can flow around other objects or regions and text can be spread over any number of pages under full control, without using the AutoBreak event-handler.

This can be done with plain text, as well as Rich Text (RTF, requires Professional Edition or higher).

14.1 CreateTextBlock

[Not supported by the Community Edition]

Creates a Text Block Object from a given text. The current font properties are assigned to the object, i.e. font name, font size, bold, italic, underlined.

Professional Edition and higher: The font properties can be changed while fragments of the text are inserted into the document.

```
method TVPETextBlock VPE.CreateTextBlock(
        string Text
)
```

string Text
    the text which is assigned to the Text Block Object

Returns:
    A newly created Text Block Object.

14.2 CreateTextBlockRTF

[Professional Edition and higher]

Creates a Text Block Object from a given Rich Text (RTF).

```
method TVPETextBlock VPE.CreateTextBlockRTF(
        string Text
)
```

string Text
the Rich Text which is assigned to the Text Block Object

Returns:
A newly created Text Block Object.

14.3 Delete

[Not supported by the Community Edition]

Destroys a Text Block Object and removes it from memory.

method void TextBlock.Delete()

Remarks:

A Text Block Object remains in memory, until this method is called, or when the parent document is closed. When a VPE document is closed, all Text Block Objects that belong to the document are released. It is always a good idea to call this method, if you are not using a Text Block Object anymore, in order to keep memory usage low.

14.4 HasText

[Not supported by the Community Edition]

Tests whether there is text in the supplied Text Block Object.

property boolean TextBlock.HasText

read; runtime only

Possible Values:

Value	Description
True	There is text in the Text Block Object
False	There is no text in the Text Block Object

14.5 Width

[Not supported by the Community Edition]

The current width of a Text Block Object. If a value is written to this property, the text within the Text Block Object is formatted accordingly to the specified width, and properties like TextBlock.Height are updated. When you call WriteTextBlock(), the text will be inserted into a document using the width you have specified with this property.

property VpeCoord TextBlock.Width

read / write; runtime only

Possible Values:

The height of a Text Block Object.

Remarks:

When a new width is assigned, VPE re-computes the internal layout of a Text Block Object.

14.6 Height

[Not supported by the Community Edition]

Returns the height of a Text Block Object, according to the width, contained text and font properties. When you output a portion of a Text Block Object, the text is removed from the Text Block Object. Therefore the height will change.

property VpeCoord TextBlock.Height

read; runtime only

Possible Values:
The height of a Text Block Object.

14.7 GetRangeHeight

[Not supported by the Community Edition]

Returns the height of a line-range of a Text Block Object, according to the width, contained text and font properties.

```
method VpeCoord TextBlock.GetRangeHeight(
        int nFirstLine,
        int nLineCount,
)
```

int nFirstLine
the starting text line index, from which the height is computed.
The first line has the index zero.

int nLineCount
the number of text lines, for which the height is computed

Returns:
The height for a range of lines of a Text Block Object.

14.8 LineCount

[Not supported by the Community Edition]

Returns the number of text lines of a Text Block Object, according to the width, contained text and font properties. When you output a portion of a Text Block Object, the text is removed from the Text Block Object. Therefore the line count will change.

property int TextBlock.LineCount

read; runtime only

Possible Values:

The number of text lines of a Text Block Object.

14.9 WriteTextBlock

[Not supported by the Community Edition]

Outputs a portion of text from the beginning of a Text Block Object, with the specified number of text lines, according to the current width and font properties.

When you output a portion of a Text Block Object, the text is removed from the Text Block Object. With successive calls to this method, you can continue to output the next part in another column or on another page.

VpeCoord VPE.WriteTextBlock(
    TextBlock *TextBlock,*
    VpeCoord *Left,*
    VpeCoord *Top,*
    int *nLineCount*
)

TextBlock TextBlock
    Text Block Object

VpeCoord Left, Top
    position where the text shall be placed in the document

int nLineCount
    the number of text lines, which shall be placed in the document

Returns:
    The bottom coordinate of the inserted text.

Remarks:
    Prior to calling this function, **you need to assign a value to** TextBlock.Width 511 at least once.

Assigning a value to TextBlock.Width causes to retrigger the rendering. If the font attributes (e.g. font size) for plain text are changed, you need to assign a value to TextBlock.Width to re-render the Text Block Object. This does only apply to the Professional Edition and higher. Lower editions will use the font, pen, etc. properties at the time when CreateTextBlock() 507 is called. RTF will also ignore changes of font properties.

The AutoBreakMode, PenSize and Rotation are not in effect, neither for plain text, nor for RTF.

Differences in values returned by TextBlock.Height 512 and TextBlock.GetRangeHeight() 513 compared to WriteTextBlock(), Render() and other text output functions: Due to historical reasons (16-bit), Print(), Write() and Render() add a small offset to the bottom coordinate of a text object. For compatibilty, WriteTextBlock() does the same. If you wish to place parts of a text block object consecutively below each other, use the values of TextBlock.Height or TextBlock.GetRangeHeight() to compute the top coordinate of the next block.

Rich Text (RTF):

Paragraph spacing properties do not work, because for each newly inserted partial text block the renderer must assume that it is the first line of the first paragraph. As a result the FirstIndent and SpaceBefore are applied to each partial text block, and SpaceBetween and SpaceAfter are ignored. As a rule of thumb, there should not be any paragraph spacing settings applied to Rich Text, which is used for Text Block Objects.

Example:

The source codes shipped with VPE contain detailed example code for many different programming languages like C#, Delphi, Java, C/C++, etc.

For using plain text with a Text Block Object, please see the 'vpedemo' source codes.

For using Rich Text (RTF) with a Text Block Object, please see the 'vppdemo' source codes.

14.10 RenderTextBlock

[Not supported by the Community Edition]

This method is useful to compute space-requirements of a Text Block Object, without inserting it into a document. Using this method, your application can make decisions regarding the layout of a Text Block, for example to issue a page break in advance.

The method renders a portion of text from the beginning of a Text Block Object, with the specified number of text lines, according to the current width and font properties.

When you render a portion of a Text Block Object, the text is removed from the Text Block Object. With successive calls to this method, you can continue to render the next part in another column or on another page.

When you have finished rendering, you can call the method Reset() 518 to restore the whole original text you had initially supplied to the Text Block Object. This way you can re-use the Text Block Object, and call WriteTextBlock() 515 - and all of its other methods and properties - to insert it finally into the document.

```
VpeCoord VPE.RenderTextBlock(
        TextBlock TextBlock,
        int nLineCount
)
```

TextBlock TextBlock
Text Block Object

int nLineCount
the number of text lines, which shall be rendered in the document

Returns:
The bottom coordinate of the rendered text.

Remarks:
All remarks of the method WriteTextBlock() 515 do also apply to this method.

14.11 Reset

[Not supported by the Community Edition]

Restores the whole original text you had initially supplied to a Text Block Object.

method void TextBlock.Reset()

Picture Functions

15 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.

15.1 PictureCacheSize

[Not supported by the Community Edition]

To increase performance, VPE has a build-in dynamic image cache. With this property you can set the maximum amount of memory used by VPE to cache image files.

See "Image Cache" in the Programmer's Manual.

property long VPE.PictureCacheSize

read / write; runtime only

Possible Values:
size of image cache in kilobytes (NOT megabytes)

Default:
65536 (= 64 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

15.2 PictureCacheUsed

Returns the used space of the image cache 521 in kilobytes.

See "Image Cache" in the Programmer's Manual.

property long VPE.PictureCacheUsed

read; runtime only

Returns:
the used space 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

15.3 GetPictureTypes

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

method string VPE.GetPictureTypes( )

Returns:

A list of picture types 524 of the following form:
"*.WMF;*.BMP;*.TIF;*.JPG;*.PCX;*.TIF"

Example:

```
string s
s = Doc.GetPictureTypes()
```

See also:

"Pictures" in the Programmer's Manual

15.4 PictureType

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.

property PictureType [long] VPE.PictureType

read / write; runtime only; also supported by TVPEObject

Possible Values:

ActiveX / VCL	Value	Enum	Comment
PIC_TYPE_AUTO	255	Auto	Automatic determination by filename-suffix (default)
PIC_TYPE_BMP	0	BMP	Windows-Bitmap
PIC_TYPE_WMF	5	WMF	WMF
PIC_TYPE_EMF	6	EMF	EMF
PIC_TYPE_TIFF	64	TIFF	TIFF-File
PIC_TYPE_GIF	65	GIF	GIF-File
PIC_TYPE_PCX	66	PCX	PCX-File
PIC_TYPE_JPEG	68	JPEG	JPEG-File
PIC_TYPE_PNG	69	PNG	PNG (Portable Network Graphic)
PIC_TYPE_ICO	70	ICO	ICO (Windows Icon)
PIC_TYPE_JNG	71	JNG	JNG (JPEG Network Graphics)
PIC_TYPE_KOALA	72	KOALA	KOA (C64 Koala Graphics)
PIC_TYPE_IFF	73	IFF	IFF/LBM (Interchangeable File Format - Amiga/Deluxe Paint)
PIC_TYPE_MNG	74	MNG	MNG (Multiple-Image Network Graphics)
PIC_TYPE_PBM	75	PBM	PBM (Portable Bitmap [ASCII])
PIC_TYPE_PBM_RAW	76	PBM_RAW	PBM (Portable Bitmap [RAW])
PIC_TYPE_PCD	77	PCD	PCD (Kodak PhotoCD)
PIC_TYPE_PGM	78	PGM	PGM (Portable Greymap [ASCII])
PIC_TYPE_PGM_RAW	79	PGM_RAW	PGM (Portable Greymap [RAW])

PIC_TYPE_PPM	80	PPM	PPM (Portable Pixelmap [ASCII])
PIC_TYPE_PPM_RAW	81	PPM_RAW	PPM (Portable Pixelmap [RAW])
PIC_TYPE_RAS	82	RAS	RAS (Sun Raster Image)
PIC_TYPE_TARGA	83	TARGA	TGA/TARGA (Truevision Targa)
PIC_TYPE_WBMP	84	WBMP	WAP/WBMP/WBM (Wireless Bitmap)
PIC_TYPE_PSD	85	PSD	PSD (Adobe Photoshop)
PIC_TYPE_CUT	86	CUT	CUT (Dr. Halo)
PIC_TYPE_XBM	87	XBM	XBM (X11 Bitmap Format)
PIC_TYPE_XPM	88	XPM	XPM (X11 Pixmap Format)
PIC_TYPE_DDS	89	DDS	DDS (DirectX Surface)
PIC_TYPE_HDR	90	HDR	HDR (High Dynamic Range Image)
PIC_TYPE_FAX_G3	91	FAX_G3	G3 (Raw fax format CCITT G.3)
PIC_TYPE_SGI	92	SGI	SGI (SGI Image Format)

Default:

PIC_TYPE_AUTO

Remarks:

Setting this property for a VPE Object 884 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.

We recommend to set this flag to PIC_TYPE_AUTO always.

Example:

ActiveX / VCL:

```
Doc.PictureType = PIC_TYPE_TIFF
Doc.Picture(1, 1, VFREE, VFREE, "image.001")
```

Will import "image.001" as TIFF file.

.NET:

```
Doc.PictureType = PictureType.TIFF
Doc.Picture(1, 1, Doc.nFree, Doc.nFree, "image.001")
```

Will import "image.001" as TIFF file.

See also:

"Pictures" in the Programmer's Manual

15.5 PictureCache

[Not supported by the Community Edition]

Instructs VPE to move all images, which are subsequently added to the document with the methods Picture() 537 and RenderPicture() 426, into the dynamic image cache.

See "Image Cache" in the Programmer's Manual.

We recommend to set this property always to true.

property boolean VPE.PictureCache

read / write; runtime only

Possible Values:

Value	Description
True	enabled
False	disabled

Default:
True

Community Edition:
The Community Edition does not provide caching of images.

See also:
"Pictures" in the Programmer's Manual

15.6 GetPicturePageCount

[Enhanced Edition and above]

Retrieves the number of available pages stored in the specified image file. At present only multipage TIFF files are supported.

method long VPE.GetPicturePageCount(
    string *FileName*
)

string FileName
  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 201 is set.

Example:

```
n = Doc.GetPicturePageCount("test.tif")
```

See also:
  "Pictures" in the Programmer's Manual

15.7 PicturePage

[Enhanced Edition and above]

Returns / sets the number of the page which will be loaded from a multipage image file by the next call to Picture() 537. The first page starts with 1.

property long VPE.PicturePage

read / write; runtime only

Possible Values:

the page number which shall be loaded from a multipage image file

Default:

1

Example:

ActiveX / VCL:

```
n = Doc.GetPicturePageCount("test.tif")
for i = 1 to n
   Doc.PicturePage = i
   Doc.Picture(0, 0, VFREE, VFREE, "test.tif")
   if i < n then
         Doc.PageBreak
   end if
next i
```

.NET:

```
n = Doc.GetPicturePageCount("test.tif")
for i = 1 to n
   Doc.PicturePage = i
   Doc.Picture(0, 0, Doc.nFree, Doc.nFree, "test.tif")
   if i < n then
         Doc.PageBreak()
   end if
next i
```

See also:

"Pictures" in the Programmer's Manual

15.8 PictureEmbedInDoc

When writing a VPE document to file (either by working with SwapFileName 195, or WriteDoc() 225), 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 220 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".

property boolean VPE.PictureEmbedInDoc

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	enabled
False	disabled

Default:
True

Remarks:
ATTENTION: When you read one or more VPE document files into the current document using ReadDoc() 229, 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

15.9 PictureKeepAspect

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.

property boolean VPE.PictureKeepAspect

write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	on (pictures will be scaled undistorted)
False	Off

Default:
True

Example:

```
Doc.PictureKeepAspect = True
Doc.Picture(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

15.10 PictureBestFit

Keeps a picture's aspect and allows for the largest zoom within a given rectangle without distortion. In contrast to PictureKeepAspect 530 this flag is useful, if you specify all four coordinates.

See "Scaling" in the Programmer's Manual for details.

property boolean VPE.PictureBestFit

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	enabled
False	disabled

Default:
False

See also:
"Pictures" in the Programmer's Manual

15.11 SetPictureDefaultDPI

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 660 bitmaps will have the resolution specified with this method.

```
method void VPE.SetPictureDefaultDPI(
        integer DPIX,
        integer DPIY
)
```

integer 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

15.12 PictureX2YResolution

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.

property boolean VPE.PictureX2YResolution

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	enabled
False	disabled

Default:
False

See also:
"Pictures" in the Programmer's Manual

15.13 PictureDrawExact

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.

property boolean VPE.PictureEmbedInDoc

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	enabled
False	disabled

Default:
False

See also:
"Pictures" in the Programmer's Manual

15.14 PictureScale2Gray

[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, Enhanced Metafiles and DXF files.

See "Scale-to-Gray Technology" in the Programmer's Manual for details.

property boolean VPE.PictureScale2Gray

read / write; runtime only

Possible Values:

Value	Description
True	enabled
False	disabled

Default:
False

See also:
"Pictures" in the Programmer's Manual

15.15 PictureScale2GrayFloat

[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 535 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.

property boolean VPE.PictureScale2GrayFloat

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	enabled
False	disabled

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

15.16 Picture

Inserts a picture object into the document. For supported file formats, please see the chapter "Pictures" in the Programmer's Manual.

method VpeCoord VPE.Picture(
VpeCoord *Left*,
VpeCoord *Top*,
VpeCoord *Right*,
VpeCoord *Bottom*,
string *FileName*
)

VpeCoord *Left, Top, Right, Bottom*

position and dimensions

Dimensions: if *Right* is VFREE, it is computed automatically; if *Bottom* 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 530 and PictureBestFit 531).

string FileName

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: PictureType 524)

Returns:

the bottom coordinate of the inserted image

Remarks:

In case of an error, LastError 201 is set.

The frame drawn around images can be made invisible by setting the PenSize 443 to zero before importing the image.

We recommend to use the methods Picture() and PictureStream() 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:

ActiveX / VCL:

```
Doc.Picture(1, 1, VFREE, VFREE, "test.tif")
```

.NET:

```
Doc.Picture(1, 1, Doc.nFree, Doc.nFree, "test.tif")
```

See also:

"Pictures" in the Programmer's Manual

15.17 PictureStream

[Professional Edition and above]

Identical to Picture() 537, but imports a picture from a memory stream.

```
method VpeCoord VPE.PictureStream(
        TVPEStream stream,
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string Identifier
)
```

TVPEStream stream
The stream-object where the picture is imported from. The stream must have been created before by calling CreateMemoryStream() 694, and of course it must have been initialized with valid image data.

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

string 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 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.

15.18 PictureResID

[Not available in .NET, use PictureDIB 542]

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 to the instance of the loaded DLL

```
method VpeCoord VPE.PictureResID(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        long hInstance,
        long ResourceID
)
```

VpeCoord *Left, Top, Right, Bottom*
    position and dimensions

long hInstance
    Instance-Handle (of type HINSTANCE)

long ResourceID
    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

15.19 PictureResName

[Not available in .NET, use PictureDIB 542]

The same as PictureResID() 540, 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 to the instance of the loaded DLL

```
method VpeCoord VPE.PictureResName(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        long hInstance,
        string ResourceName
)
```

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

long hInstance
Instance-Handle (of type HINSTANCE)

string ResourceName
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

15.20 PictureDIB

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

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

method VpeCoord VPE.PictureDIB(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    IntPtr [long] *hDIB*
)

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

IntPtr [long] hDIB
handle to DIB (of type HGLOBAL)

Returns:
the bottom coordinate (y2) of the inserted image

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.

PictureDIB() expects a handle for a memory block that was allocated with GlobalAlloc(). PictureDIB() 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

Barcode Functions (1D)

16 Barcode Functions (1D)

[Enhanced Edition and above only]

These methods and properties deal with the generation and presentation of the various barcode formats offered by VPE.

See also:

"Barcodes (1D)" in the Programmer's Manual

16.1 SetBarcodeParms

[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).

method void VPE.SetBarcodeParms(
    BarcodeParms [long] *MainCode*,
    BarcodeParms [long] *AddCode*
)

also supported by TVPEObject

BarcodeParms [long] MainCode
one of the constants below

BarcodeParms [long] AddCode
one of the constants below

Default:
BCP_BOTTOM, BCP_BOTTOM

Possible values for TopBottom and AddTopBottom are:

ActiveX / VCL	Value	Enum	Comment
BCP_BOTTOM	0	Bottom	Text at Bottom
BCP_TOP	1	Top	Text at Top
BCP_HIDE	2	Hide	Text Hidden
BCP_DEFAULT	3	Default	Text at default position appropriate to barcode type

See also:
"Barcodes (1D)" in the Programmer's Manual

16.2 BarcodeMainTextParms

Sets the position of the main barcode label text.

property BarcodeParms [integer] VPE.BarcodeMainTextParms

read / write; runtime only; also supported by TVPEObject

Possible Values:

ActiveX / VCL	Value	Enum	Comment
BCP_BOTTOM	0	Bottom	Text at Bottom
BCP_TOP	1	Top	Text at Top
BCP_HIDE	2	Hide	Text Hidden
BCP_DEFAULT	3	Default	Text at default position appropriate to barcode type

Default:
BCP_BOTTOM

See also:
"Barcodes (1D)" in the Programmer's Manual

16.3 BarcodeAddTextParms

Sets the position of the add-on barcode label text.

property BarcodeParms [integer] VPE.BarcodeMainTextParms

read / write; runtime only; also supported by TVPEObject

Possible Values:

ActiveX / VCL	Value	Enum	Comment
BCP_BOTTOM	0	Bottom	Text at Bottom
BCP_TOP	1	Top	Text at Top
BCP_HIDE	2	Hide	Text Hidden
BCP_DEFAULT	3	Default	Text at default position appropriate to barcode type

Default:
BCP_BOTTOM

See also:
"Barcodes (1D)" in the Programmer's Manual

16.4 BarcodeAlignment

[Enhanced Edition and above]

Sets the alignment of barcodes within the rectangle they are drawn.

property BarcodeAlignment [integer] VPE.BarcodeAlignment

read / write; runtime only; also supported by TVPEObject

Possible Values:

ActiveX / VCL	Value	Enum	Comment
ALIGN_LEFT	0	Left	
ALIGN_RIGHT	1	Right	
ALIGN_CENTER	2	Center	

Default:

ALIGN_LEFT

See also:

"Barcodes (1D)" in the Programmer's Manual

16.5 BarcodeAutoChecksum

[Enhanced Edition and above]

Specifies, whether an auto checksum for a barcode will be generated automatically or not.

property integer VPE.BarcodeAutoChecksum

read / write; runtime only; also supported by TVPEObject

Possible Values:

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

16.6 BarcodeThinBar

[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 551, the ratio of thin to thick modules can be adjusted.

property integer VPE.BarcodeThinBar

read / write; runtime only; also supported by TVPEObject

Possible Values:

the relative width for thin barcode modules, a value of zero will cause VPE to choose an optimal value in accordance to the available space for drawing the barcode

Default:

0

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

16.7 BarcodeThickBar

[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 550 and BarcodeThickBar, the ratio of thin to thick modules can be adjusted.

property integer VPE.BarcodeThickBar

read / write; runtime only; also supported by TVPEObject

Possible Values:

the relative width for thick barcode modules, a value of zero will cause VPE to choose an optimal value in accordance to the available space for drawing the barcode

Default:

0

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

16.8 Barcode

[Enhanced Edition and above]

Generates and draws a barcode within a rectangle at position Left, Top with the right border at Right and the bottom border at Bottom.

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 by TextColor 492. The font used for the barcode labels can be specified with the property FontName 477, for example UPC-A and UPC-E require an OCR-B font. In addition the font size for barcode labels can be specified with the property FontSize 478.

method void VPE.Barcode(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    BarcodeType [integer] *CodeType*,
    string *Code*,
    string *AddCode*
)

VpeCoord *Left, Top, Right, Bottom*
    position and dimensions

BarcodeType [integer] CodeType
    see below

string Code
    string with the code (e.g. "123456")

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

the parameter "CodeType" can be one of the following values:

ActiveX / VCL	Value	Enum	Comment
BCT_EAN13	1	Ean13	Checkdigit required, OCR-B Font required
BCT_EAN13_2	10	Ean13_2	Checkdigit required, OCR-B Font required
BCT_EAN13_5	11	Ean13_5	Checkdigit required, OCR-B Font required
BCT_EAN8	2	Ean8	Checkdigit required, OCR-B Font required

BCT_EAN8_2	12	Ean8_2	Checkdigit required, OCR-B Font required
BCT_EAN8_5	13	Ean8_5	Checkdigit required, OCR-B Font required
BCT_UPCA	3	UpcA	Checkdigit required, OCR-B Font required
BCT_UPCA_2	14	UpcA_2	Checkdigit required, OCR-B Font required
BCT_UPCA_5	15	UpcA_5	Checkdigit required, OCR-B Font required
BCT_UPCE	9	UpcE	first digit always '0', Checkdigit required,OCR-B Font required
BCT_UPCE_2	16	UpcE_2	first digit always '0', Checkdigit required, OCR-B Font required
BCT_UPCE_5	17	UpcE_5	first digit always '0', Checkdigit required, OCR-B Font required
BCT_EAN2	28	Ean2	no Checkdigit
BCT_EAN5	29	Ean5	no Checkdigit
BCT_CODABAR	5	Codabar	Checkdigit optional, uThick / uThin
BCT_2OF5	7	B2Of5	2 of 5 Industrial (25ID), Checkdigit optional, uThick / uThin
BCT_INTERLEAVED2OF5	8	Interleaved2Of5	2 of 5 Interleaved (25IL), Checkdigit optional, uThick / uThin
BCT_CODE39	6	Code39	Code 3 of 9, Checkdigit optional, uThick / uThin
BCT_CODE39EXT	30	Code39Ext	Code 3 of 9 Extended, Checkdigit optional, uThick / uThin
BCT_CODE93	21	Code93	Code 93, Checkdigit optional
BCT_CODE93EXT	31	Code93Ext	Code 93 Extended, Checkdigit optional
BCT_POSTNET	22	Postnet	uThick / uThin are fixed
BCT_CODE128	26	Code128	set of CODE 128A, B and C, Checkdigit required for GS1-128 / UCC-128 use EAN128
BCT_EAN128	27	Ean128	set of EAN 128A, B and C, Checkdigit required
BCT_RM4SCC	32	Rm4Scc	Royal Mail 4 State Customer Code, Checkdigit required
BCT_MSI	33	Msi	Checkdigit optional, uThick / uThin
BCT_ISBN	34	Isbn	International Standard Book Number, Checkdigit required
BCT_ISBN_5	35	Isbn_5	ISBN + EAN5 (for pricing), Checkdigit required
BCT_IDENTCODE	36	Identcode	Identcode of the Deutsche Post AG, Checkdigit required
BCT_LEITCODE	37	Leitcode	Leitcode of the Deutsche Post AG, Checkdigit required
BCT_PZN	38	Pzn	Pharma Zentral Code, Checkdigit required, uThick / uThin
BCT_CODE11	39	Code11	Code 11, Checkdigit optional, uThick / uThin
BCT_2OF5MATRIX	40	B2OF5Matrix	2 of 5 Matrix, Checkdigit optional, uThick / uThin
BCT_TELEPENA	41	TelepenA	Telepen-A, Checkdigit optional
BCT_INTELLIGENT_MAIL	42	IntelligentMail	Intelligent Mail

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 520 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 the **ActiveX**, do not supply the value 0 (zero) for the AddCode parameter, if the barcode type does not support add-on codes.*
Instead, supply an empty string, i.e. "".

Example:

```
Doc.Barcode 1, 1, -4, -1, BCT_INTERLEAVED2OF5, "12345", ""
```

Barcode Functions (2D)

17 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

17.1 Bar2DAlignment

[Professional Edition and above only]

Sets the horizontal and vertical alignment of the 2D-barcodes within the given rectangle.

property Bar2DAlignment [integer] VPE.Bar2DAlignment

read / write; runtime only; also supported by TVPEObject

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VBAR2D_ALIGN_CENTER	0	Center	horizontally and vertically centered
VBAR2D_ALIGN_CENTER_H	0	CenterH	horizontally centered
VBAR2D_ALIGN_CENTER_V	0	CenterV	vertically centered
VBAR2D_ALIGN_LEFT	1	Left	
VBAR2D_ALIGN_RIGHT	2	Right	
VBAR2D_ALIGN_TOP	4	Top	
VBAR2D_ALIGN_BOTTOM	8	Bottom	

Default:

VBAR2D_ALIGN_CENTER

Remarks:

You combine values for horizontal and vertical alignment by adding two flags.

Example:

ActiveX / VCL:

```
Doc.Bar2DAlignment = VBAR2D_ALIGN_LEFT + VBAR2D_ALIGN_TOP
```

Will align the barcode to the top-left corner.

.NET:

```
VB: Doc.Bar2DAlignment = Bar2DAlignment.Left + Bar2DAlignment.Top
C#: Doc.Bar2DAlignment = Bar2DAlignment.Left | Bar2DAlignment.Top
```

Will align the barcode to the top-left corner.

See also:

"Barcodes (2D)" in the Programmer's Manual

17.2 DataMatrixEncodingFormat

[Professional Edition and above only]

Sets the encoding format ID for DataMatrix symbologies.

property DataMatrixEncodingFormat [integer]
VPE.DataMatrixEncodingFormat

read / write; runtime only; also supported by TVPEObject

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VBAR2D_DATA_MATRIX_ENC_BASE11	1	Base11	numeric data, 3.5 bits per digit
VBAR2D_DATA_MATRIX_ENC_BASE27	2	Base27	upper case alphabetic, 4.8 bits per character
VBAR2D_DATA_MATRIX_ENC_BASE41	3	Base41	upper case alphanumeric and punctuation, 5.5 bits per character
VBAR2D_DATA_MATRIX_ENC_BASE37	4	Base37	upper case alphanumeric, 5.25 bits per character
VBAR2D_DATA_MATRIX_ENC_ASCII	5	Ascii	128 ASCII set, 7 bits per character
VBAR2D_DATA_MATRIX_ENC_BINARY	6	Binary	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

17.3 DataMatrixEccType

[Professional Edition and above only]

Sets the error checking and correction scheme for DataMatrix symbologies.

property DataMatrixEccType [integer] VPE.DataMatrixEccType

read / write; runtime only; also supported by TVPEObject

Possible Values:

the error correction scheme:

ActiveX / VCL	Value	Enum	Comment
VBAR2D_DATA_MATRIX_ECC000	0	Ecc000	
VBAR2D_DATA_MATRIX_ECC010	1	Ecc010	
VBAR2D_DATA_MATRIX_ECC040	2	Ecc040	
VBAR2D_DATA_MATRIX_ECC050	3	Ecc050	
VBAR2D_DATA_MATRIX_ECC060	4	Ecc060	
VBAR2D_DATA_MATRIX_ECC070	5	Ecc070	
VBAR2D_DATA_MATRIX_ECC080	6	Ecc080	
VBAR2D_DATA_MATRIX_ECC090	7	Ecc090	
VBAR2D_DATA_MATRIX_ECC100	8	Ecc100	
VBAR2D_DATA_MATRIX_ECC110	9	Ecc110	
VBAR2D_DATA_MATRIX_ECC120	10	Ecc120	
VBAR2D_DATA_MATRIX_ECC130	11	Ecc130	
VBAR2D_DATA_MATRIX_ECC140	12	Ecc140	
VBAR2D_DATA_MATRIX_ECC200	26	Ecc200	

Default:

VBAR2D_DATA_MATRIX_ECC200

See also:

"Barcodes (2D)" in the Programmer's Manual

17.4 DataMatrixRows

[Professional Edition and above only]

Sets the desired number of matrix rows (0: choose automatically based on message).

property integer VPE.DataMatrixRows

read / write; runtime only; also supported by TVPEObject

Possible Values:
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

17.5 DataMatrixColumns

[Professional Edition and above only]

Sets the desired number of matrix columns (0: choose automatically based on message).

property integer VPE.DataMatrixColumns

read / write; runtime only; also supported by TVPEObject

Possible Values:

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

17.6 DataMatrixMirror

[Professional Edition and above only]

Specifies, whether a mirrored version of the barcode shall be created (applies to ECC000 – ECC140 only).

property boolean VPE.DataMatrixMirror

read / write; runtime only; also supported by TVPEObject

Possible Values:

Value	Description
True	do not mirror
False	Mirror

Default:

False

See also:

"Barcodes (2D)" in the Programmer's Manual

17.7 DataMatrixBorder

[Professional Edition and above only]

Sets the number of modules used for the border (usually 1).

property integer VPE.DataMatrixBorder

read / write; runtime only; also supported by TVPEObject

Possible Values:
the number of modules used for the border (usually 1)

Default:
1

See also:
"Barcodes (2D)" in the Programmer's Manual

17.8 DataMatrix

[Professional Edition and above only]

Inserts a DataMatrix barcode into the VPE document.

method void VPE.DataMatrix(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
    coordinates

string Text
    the text of the barcode

Example:

```
Doc.DataMatrix(1, 1, -3, -3, "30Q324343430794<OQQ")
```

Remarks:
    sets LastError 201

See also:
    "Barcodes (2D)" in the Programmer's Manual

17.9 RenderDataMatrix

[Professional Edition and above only]

Computes the dimensions of a DataMatrix barcode.

method void VPE.RenderDataMatrix(
    string *Text*,
    VpeCoord *nWidth*,
    VpeCoord *nHeight*,
    VpeCoord *nModuleWidth*
)

string Text
    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 choose itself an optimum barcode module width. If it is non-zero, you can specify the width of a barcode module.

Example:

```
// Compute the smallest possible rectangle for a given text
Doc.RenderDataMatrix("30Q324343430794<OQQ", 0, 0, 0)
xsize = Doc.nRenderWidth
ysize = Doc.nRenderHeight

// Insert the barcode into the document
Doc.DataMatrix(1, 1, -xsize, -ysize, "30Q324343430794<OQQ")
```

Remarks:
    sets LastError 201

See also:
    "Barcodes (2D)" in the Programmer's Manual

17.10 QRCodeVersion

[Professional Edition and above only]

Defines the maximum data capacity for QR Codes (depending on EccLevel and encoding Mode), see http://www.qrcode.com.

The symbol versions of QR Code range from Version 1 to Version 40. Each version has a different module configuration or number of modules. (The module refers to the black and white dots that make up QR Code.)

"Module configuration" refers to the number of modules contained in a symbol, commencing with Version 1 (21 × 21 modules) up to Version 40 (177 × 177 modules). Each higher version number comprises 4 additional modules per side.

property integer VPE.QRCodeVersion

read / write; runtime only; also supported by TVPEObject

Possible Values:
the version, 0 = automatic

Default:
0 = automatic

See also:
"Barcodes (2D)" in the Programmer's Manual

17.11 QRCodeEccLevel

[Professional Edition and above only]

Sets the error correction code level for QR Codes. The higher the level, i.e. the higher the error tolerance, the less user data can be encoded within a symbol.

property QRCodeEccLevel [integer] VPE.QRCodeEccLevel

read / write; runtime only; also supported by TVPEObject

Possible Values:

the error correction code level

ActiveX / VCL	Value	Enum	Comment
VBAR2D_QRCODE_ECC_LEVEL_L	0	L	7% of codewords can be restored
VBAR2D_QRCODE_ECC_LEVEL_M	1	M	15% of codewords can be restored
VBAR2D_QRCODE_ECC_LEVEL_Q	2	Q	25% of codewords can be restored
VBAR2D_QRCODE_ECC_LEVEL_H	3	H	30% of codewords can be restored

Default:

0 = VBAR2D_QRCODE_ECC_LEVEL_L

See also:

"Barcodes (2D)" in the Programmer's Manual

17.12 QRCodeMode

[Professional Edition and above only]

Sets encoding mode for QR Codes. The QR Code provides four different encoding modes, the user data capacity depends on the encoding and error correction level:

Encoding Modes and Data Capacity	
Numeric code only	max. 7,089 characters
Alphanumeric	max. 4,296 characters
Binary (8 bits)	max. 2,953 bytes
Kanji/Kana	max. 1,817 characters

VPE chooses automatically the best encoding. Only for Kanji you need to specify that the Kanji encoding shall be used.

property QRCodeMode [integer] VPE.QRCodeMode

read / write; runtime only; also supported by TVPEObject

Possible Values:

the encoding mode

ActiveX / VCL	Value	Enum	Comment
VBAR2D_QRCODE_MODE_DEFAULT	2	Default	VPE chooses automatically between numeric, alphanumeric and binary mode
VBAR2D_QRCODE_MODE_KANJI	3	Kanji	Text is encoded in Kanji (Shift-JIS)

Default:

VBAR2D_QRCODE_MODE_DEFAULT

See also:

"Barcodes (2D)" in the Programmer's Manual

17.13 QRCodeBorder

[Professional Edition and above only]

Specifies the number of modules used for the border (usually 4) around QR Codes. The QR Code symbol area requires a border or "quiet zone" around it to be used. The border is a clear area around a symbol where nothing is printed. QR Code requires a four-module (or more) wide border at all sides of a symbol.

If you position no other objects near a QR Code, you might wish to set the border to zero, so the whole rectangle specified for the code is used for drawing the symbol pattern.

property integer VPE.QRCodeBorder

read / write; runtime only; also supported by TVPEObject

Possible Values:
the number of modules used for the border (usually 4)

Default:
4

See also:
"Barcodes (2D)" in the Programmer's Manual

17.14 QRCode

[Professional Edition and above only]

Inserts a QR Code barcode into the VPE document.

method void VPE.QRCode(
VpeCoord *Left*,
VpeCoord *Top*,
VpeCoord *Right*,
VpeCoord *Bottom*,
string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
coordinates

string Text
the text of the barcode

Example:

```
Doc.QRCode(1, 1, -3, -3, "Hello World")
```

Remarks:
sets LastError 201

See also:
"Barcodes (2D)" in the Programmer's Manual

17.15 RenderQRCode

[Professional Edition and above only]

Computes the dimensions of a QR Code barcode.

method void VPE.RenderQRCode(
string *Text*,
VpeCoord *nWidth*,
VpeCoord *nHeight*,
VpeCoord *nModuleWidth*
)

string Text
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 choose itself an optimum barcode module width. If it is non-zero, you can specify the width of a barcode module.

Example:

```
// Compute the smallest possible rectangle for a given text
Doc.RenderQRCode("Hello World", 0, 0, 0)
xsize = Doc.nRenderWidth
ysize = Doc.nRenderHeight

// Insert the barcode into the document
Doc.QRCode(1, 1, -xsize, -ysize, "Hello World")
```

Remarks:
sets LastError 201

See also:
"Barcodes (2D)" in the Programmer's Manual

17.16 MaxiCode

[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 Text.

method void VPE.MaxiCode(
  VpeCoord *Left*,
  VpeCoord *Top*,
  VpeCoord *Right*,
  VpeCoord *Bottom*,
  string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
  coordinates

string Text
  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"
   "!!!!!!!!!!!!!!!!!!!!!!!!!!"

Doc.MaxiCode(1, 6, -3, -3, szUPS)
```

Remarks:
  sets LastError 201

See also:
  "Barcodes (2D)" in the Programmer's Manual

17.17 RenderMaxiCode

[Professional Edition and above only]

Computes the dimensions of a MaxiCode barcode.

method void VPE.RenderMaxiCode(
        string *Text*
)

string Text
    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 201.

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"
   "!!!!!!!!!!!!!!!!!!!!!!!!!!"

Doc.RenderMaxiCode(szUPS)
xsize = Doc.nRenderWidth
ysize = Doc.nRenderHeight
```

See also:
    "Barcodes (2D)" in the Programmer's Manual

17.18 MaxiCodeEx

[Professional Edition and above only]

Inserts a MaxiCode barcode into the VPE document. It is recommended to use MaxiCode() 572 instead. This function has been implemented for completeness.

```
method void VPE.MaxiCodeEx(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        int nMode,
        string Zip,
        int nCountryCode,
        int nServiceClass,
        string Message,
        int nSymbolNumber,
        int nSymbolCount
)
```

VpeCoord *Left, Top, Right, Bottom*
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)

string Zip
Postal code, for example D40211 or 42119

int nCountryCode
ISO 3166 country code (for example 276 for Germany)

int nServiceClass
Carrier-dependent service class

string Message
Secondary message

int nSymbolNumber
Total number of symbols (structured append)

int nSymbolCount
Current number of symbol (structured append)

Example:

```
Doc.MaxiCodeEx(1, 11, -5, -5, 2, "068107317", 840, 001,
               "AOE This is MaxiCode AOE", 1, 1)
```

Remarks:
sets LastError 201

See also:
"Barcodes (2D)" in the Programmer's Manual

17.19 RenderMaxiCodeEx

[Professional Edition and above only]

Computes the dimensions of a MaxiCode barcode.

```
method void VPE.MaxiCodeEx(
        int nMode,
        string Zip,
        int nCountryCode,
        int nServiceClass,
        string Message,
        int nSymbolNumber,
        int nSymbolCount
)
```

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)

string Zip

Postal code, for example D40211 or 42119

int nCountryCode

ISO 3166 country code (for example 276 for Germany)

int nServiceClass

Carrier-dependent service class

string Message

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 201.

Example:

```
Doc.RenderMaxiCodeEx(1, 11, -5, -5, 2, "068107317", 840, 001,
                     "AOE This is MaxiCode AOE", 1, 1)
xsize = Doc.nRenderWidth
ysize = Doc.nRenderHeight
```

See also:

"Barcodes (2D)" in the Programmer's Manual

17.20 PDF417ErrorLevel

[Professional Edition and above only]

Sets the error level for PDF417 581 barcodes.

property integer VPE.PDF417ErrorLevel

read / write; runtime only; also supported by TVPEObject

Possible Values:

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

17.21 PDF417Rows

[Professional Edition and above only]

Sets the desired number of matrix rows (0: choose automatically based on message).

property integer VPE.PDF417

read / write; runtime only; also supported by TVPEObject

Possible Values:
Desired number of matrix rows (0: choose automatically based on message).

Default:
0

See also:
"Barcodes (2D)" in the Programmer's Manual

17.22 PDF417Columns

[Professional Edition and above only]

Sets the desired number of matrix columns (0: choose automatically based on message).

property integer VPE.PDF417Columns

read / write; runtime only; also supported by TVPEObject

Possible Values:

Desired number of matrix columns (0: choose automatically based on message).

Default:

0

See also:

"Barcodes (2D)" in the Programmer's Manual

17.23 PDF417

[Professional Edition and above only]

Inserts a PDF417 barcode into the VPE document.

method void VPE.PDF417(
VpeCoord *Left*,
VpeCoord *Top*,
VpeCoord *Right*,
VpeCoord *Bottom*,
string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
coordinates

string Text
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\tP/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";

Doc.PDF417(1, 17, -5.5, -3.5, szStr);
```

Remarks:
sets LastError 201

See also:
"Barcodes (2D)" in the Programmer's Manual

17.24 RenderPDF417

[Professional Edition and above only]

Computes the dimensions of a PDF417 581 barcode.

```
method void VPE.RenderPDF417(
        string Text,
        VpeCoord nWidth,
        VpeCoord nHeight,
        VpeCoord nModuleWidth
)
```

string Text
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 choose 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\tP/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";

// Compute the smallest possible rectangle for a given
textDoc.RenderPDF417(szStr, 0, 0, 0)
xsize = Doc.nRenderWidth
ysize = Doc.nRenderHeight

// Insert the barcode into the document
Doc.PDF417(1, 1, -xsize, -ysize, szStr)
```

Remarks:

sets LastError 201

See also:

"Barcodes (2D)" in the Programmer's Manual

17.25 AztecFlags

[Professional Edition and above only]

Sets the flag for the interpretation of the Text string parameter of the Aztec() 589-method.

property boolean VPE.AztecFlags

read / write; runtime only; also supported by TVPEObject

Possible Values:

If true, the lpszString parameter of the Aztec() 589-function uses "<Esc>n" for FLG(n), and "<Esc><Esc>" for "<Esc>"

Default:

False

See also:

"Barcodes (2D)" in the Programmer's Manual

17.26 AztecControl

[Professional Edition and above only]

Sets the flag for the interpretation of the lpszString parameter of the Aztec() 589 -function.

property integer VPE.AztecControl

read / write; runtime only; also supported by TVPEObject

Possible Values:

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

17.27 AztecMenu

[Professional Edition and above only]

Specifies, if this is a "Menu" symbol.

property boolean VPE.AztecMenu

read / write; runtime only; also supported by TVPEObject

Possible Values:

specifies, if this is a "Menu" symbol

Default:

False

See also:

"Barcodes (2D)" in the Programmer's Manual

17.28 AztecMultipleSymbols

[Professional Edition and above only]

Specifies # of symbols for message… 1 (normal) to 26.

property integer VPE.AztecMultipleSymbols

read / write; runtime only; also supported by TVPEObject

Possible Values:

# of symbols for message… 1 (normal) to 26

Default:

1

See also:

"Barcodes (2D)" in the Programmer's Manual

17.29 AztecID

[Professional Edition and above only]

Specifies an optional null-terminated ID field for append.

property string VPE.AztecID

write; runtime only; also supported by TVPEObject

Possible Values:
optional ID field for append

Default:
NULL (empty)

See also:
"Barcodes (2D)" in the Programmer's Manual

17.30 Aztec

[Professional Edition and above only]

Inserts an Aztec barcode into the VPE document.

method void VPE.Aztec(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
    coordinates

string Text
    the text of the barcode

Example:

```
Doc.Aztec(1, 23, -3, -3, "This is a message")
```

Remarks:
    sets LastError 201

See also:
    "Barcodes (2D)" in the Programmer's Manual

17.31 RenderAztec

[Professional Edition and above only]

Computes the dimensions of an Aztec 589 barcode.

method void VPE.RenderAztec(
string *Text*,
VpeCoord *nWidth*,
VpeCoord *nHeight*,
VpeCoord *nModuleWidth*
)

string Text
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 choose itself an optimum barcode module width. If it is non-zero, you can specify the width of a barcode module.

Example:

```
// Compute the smallest possible rectangle for a given text
Doc.RenderAztec("This is a message", 0, 0, 0)
xsize = Doc.nRenderWidth
ysize = Doc.nRenderHeight

// Insert the barcode into the document
Doc.Aztec(1, 1, -xsize, -ysize, "This is a message")
```

Remarks:
sets LastError 201

See also:
"Barcodes (2D)" in the Programmer's Manual

E-Mail Functions

18 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.

Note:

VPE is not an e-mail component. It does only provide basic e-mail support. If you need extended e-mailing features, process the event VPE_BEFORE_MAIL and cancel the event. After cancelling the event you can execute your own mailing code, using sophisticated MAPI components. To attach for example the current VPE document as PDF to an e-mail created by a MAPI component, simply write the current VPE document to a temporary file using WriteDoc() and attach it to the e-mail. You could even generate and attach at this moment a new temporary VPE document with different content, for example with annotations or a watermark.

18.1 Sending Mail on 64-Bit Windows

VPE is using the Simple MAPI or Extended MAPI subsystem of the Windows operating system. Due to the design of both application interfaces, either subsystem is bound to the CPU architecture, for which your application is compiled / executed.

At the time of this writing, this means: the 64-bit version of your applications require a 64-bit mail client, and the 32-bit version of your applications require a 32-bit mail client.

Maybe Microsoft - or the developers of mail clients - will address this issue in a later version and provide bridging from one CPU architecture to the other.

18.2 IsMAPIInstalled

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

Checks the machine if MAPI is installed. This property is obsolete, use the property MAPIType 596 instead.

property MAPIStatus [integer] VPE.IsMAPIInstalled

read; runtime only

Returns:

ActiveX / VCL	Value	Enum	Comment
VMAPI_NOT_INSTALLED	0	NotInstalled	MAPI is not installed
VMAPI_INSTALLED	1	Installed	MAPI is installed
VMAPI_UNSURE	2	Unsure	unsure (obsolete, was for 16-bit version)

Remarks:

In case of an error, LastError 201 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 (Win9x / 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() 608
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() 189, if you are sure a working MAPI client is installed (see EnableMailButton 244).

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.

18.3 MAPIType

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

Checks the machine, what type of MAPI is installed. This property supersedes the property IsMAPIInstalled 594 and should be used instead.

property MAPIType [integer] VPE.MAPIType

read / write; runtime only

Returns:

ActiveX / VCL	Value	Enum	Comment
VMAPI_TYPE_NOT_INSTALLED	0	NotInstalled	MAPI is not installed
VMAPI_TYPE_EXTENDED	1	Extended	Extended MAPI is used
VMAPI_TYPE_SIMPLE	2	Simple	Simple MAPI is used

Remarks:

Extended MAPI and Simple MAPI are both APIs designed by Microsoft. Extended MAPI is a rather complex API, which is not supported by some mail clients, e.g. Thunderbird. VPE determines the MAPI type which is supported by the default mail client, and returns it in this property. Sending the **body** of an e-mail message (this is different from attaching a VPE document as exported HTML) as HTML or RTF is only supported reliably, if Extended MAPI is supported.

If both MAPI types are available on the machine, VPE selects Simple MAPI by default. Due to the rather chaotic infrastructure of the Windows messaging subsystem, Simple MAPI causes less problems.

18.4 MailSender

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

Allows you to specify the message sender by code.

property string VPE.MailSender

write; runtime only

Possible Values:
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.

If Extended MAPI is active, setting this property has no effect.

Example:

```
Doc.MailSender = "Support@IdealSoftware.com"
```

18.5 AddMailReceiver

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

Allows you to specify a list of e-mail receivers by code.

```
method void VPE.AddMailReceiver(
        string Receiver,
        RecipientClass [long] RecipientClass
)
```

string Receiver
the name / e-mail address of the receiver

RecipientClass [long] RecipientClass
posssible values are:

ActiveX / VCL	Value	Enum	Comment
VMAIL_ORIG	0	Orig	Recipient is message originator
VMAIL_TO	1	To	Recipient is a primary recipient
VMAIL_CC	2	Cc	Recipient is a copy recipient
VMAIL_BCC	3	Bcc	Recipient is blind copy recipient
VMAIL_RESOLVE_NAME	-2147483648	ResolveName	try to resolve the name from the mail client's address book (Simple MAPI only)

Remarks:

The list of recipients will be valid until ClearMailReceivers() 600 is called.

Simple MAPI:
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.

```
E.g. Doc.AddMailReceiver("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 external 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 607 = 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.

Please note that VMAIL_RESOLVE_NAME is not supported, if VPE uses Extended MAPI.

Example:

ActiveX / VCL:
```
Doc.AddMailReceiver("max@alpha.com", VMAIL_CC)
```

.NET:
```
Doc.AddMailReceiver("max@alpha.com", RecipientClass.Cc)
```
Will add "max@alpha.com" as carbon copy recipient for the same e-mail.

ActiveX / VCL:
```
Doc.AddMailReceiver("[FAX: +49 1234 12345678]", VMAIL_TO)
```

.NET:
```
Doc.AddMailReceiver("[FAX: +49 1234 12345678]", RecipientClass.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.

18.6 ClearMailReceivers

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

Clears the list of mail recipients created with AddMailReceiver() 598.

```
method void VPE.ClearMailReceivers(
)
```

18.7 AddMailAttachment

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

Allows you to specify a list of e-mail attachments (files) by code.

```
method void VPE.AddMailAttachment(
        String PathName,
        String FileName
)
```

String PathName

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 MailDoc() 608 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 MailDoc() method 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.

String FileName

The filename seen by the recipient. This name can differ from the filename in PathName if temporary files are being used. If FileName is NULL or empty (""), the filename from PathName is used. If the attachment is an OLE object, FileName 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 SwapFileName 195 is set - the document file name. To clear that default, call ClearMailAttachments() 604 before specifying your own attachments.

The property MailAutoAttachDocType 603 controls, whether a VPE Document or a PDF Document is attached to the e-mail.

Examples:

```
Doc.ClearMailAttachments()
```

Deletes the default attachment, which is the VPE Document itself.

```
Doc.AddMailAttachment("c:\important_file.zip", "")
```

Sets "c:\important_file.zip" as the ONLY attachment. The VPE Document is not sent, because ClearMailAttachments() deleted the default attachment of the VPE Document.

```
Doc.AddMailAttachment("c:\important_file.txt", "")
```

Adds "c:\important_file.txt" as second attachment.

```
Doc.WriteDoc("c:\tmp\doc.vpe")
Doc.AddMailAttachment("c:\tmp\doc.vpe", "")
```

Adds "c:\tmp\doc.vpe" as third attachment.

18.8 MailAutoAttachDocType

[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 AddMailAttachment 601).

property DocExportType [integer] VPE.MailAutoAttachDocType

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VPE_DOC_TYPE_AUTO	0	Auto	MailDoc() 608 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	VPE	MailDoc() will attach the document as VPE document file, regardless of the file suffix
VPE_DOC_TYPE_PDF	2	PDF	MailDoc() will attach the document as PDF document file, regardless of the file suffix
VPE_DOC_TYPE_HTML	3	HTML	MailDoc() will attach the document as HTML document file, regardless of the file suffix
VPE_DOC_TYPE_XML	4	XML	VPE XML Format
VPE_DOC_TYPE_ODT	5	ODT	OpenOffice Document Text

Default:

VPE_DOC_TYPE_PDF

18.9 ClearMailAttachments

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

Clears the list of mail attachments created with AddMailAttachment 601.

```
method void ClearMailAttachments(
)
```

18.10 MailSubject

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

Allows you to specify the e-mail subject by code.

property string VPE.MailSubject

write; runtime only

Possible Values:

the subject of the message

Example:

```
Doc.MailSubject = "New VPE v3.0"
```

18.11 MailText

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

Allows you to specify the e-mail text by code.

property string VPE.MailText

write; runtime only

Possible Values:

the message text

Remarks:

If VPE is using Extended MAPI (see MAPIType 596), you can also use **HTML or RTF** for the body text of e-mail messages. For HTML the message text must begin with “<html>” and for RTF the message text must begin with “{\rtf”.

The Community Edition does not support Extended MAPI.

Example:

```
Doc.MailText = "Dear customer of VPE," + chr$(13) + chr$(10) +
               "we have a new version released."
```

18.12 MailWithDialog

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

Specifies, whether a dialog is shown to the user with input fields for recipients 598, subject 605, text 606 and attachments 601 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() 608 by code. It is very useful to set **MailWithDialog** to false, if you do automatic serial e-mailings in the background without user interaction.

property boolean VPE.MailWithDialog

write; runtime only

Possible Values:

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 AddMailReceiver() 598), 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.

18.13 MailDoc

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

Sends a mail with the current Mail-Properties.

method boolean VPE.MailDoc(
)

Returns:

Value	Description
True	success
False	failure

Remarks:

In case of an error, LastError 201 is set.

The name of the default attachment is either the DevJobName 353 (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 that contain pictures 520 or UDO's 646. For details please see WriteDoc 225.

Example:

```
Doc.MailSubject = "VPE Demo"
Doc.AddMailReceiver("MrX@dummyxyz.com", VMAIL_TO)
Doc.AddMailReceiver("MrY@dummyxyz.com", VMAIL_CC)
Doc.AddMailAttachment("c:\data\report.vpe", "")
Doc.MailDoc()
Doc.AddMailReceiver("[FAX: +49 1234 12345678]", VMAIL_TO)
Doc.MailDoc()
```

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.

This page is intentionally left blank.

RTF Functions

19 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 "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 SetFontSubstitution() 479 available to substitute fonts, so you can use other fonts in place of the fonts used within an RTF document.

19.1 WriteRTF

Outputs RTF text within a rectangle at position *Left, Top* with the right border at *Right* and the bottom border at *Bottom*.

The pen is invisible.

method VpeCoord VPE.WriteRTF(
 VpeCoord *Left*,
 VpeCoord *Top*,
 VpeCoord *Right*,
 VpeCoord *Bottom*,
 string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
 position and dimensions

string Text
 the string to output

Returns:
 the bottom y-coordinate generated by the output

Remarks:
 VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

 In case of an error, LastError 201 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 520 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:

ActiveX / VCL:
```
Doc.SetFont("Arial", 12)
Doc.WriteRTF(1, 1, -5, VFREE, "Hello \b World!")
Doc.SetFont("Times New Roman", 16)
Doc.TextUnderline = True
Doc.WriteRTF(VLEFT, VBOTTOM, VRIGHT, VFREE, "Hello \b World!")
```

.NET:
```
Doc.SetFont("Arial", 12)
Doc.WriteRTF(1, 1, -5, Doc.nFree, "Hello \b World!")
Doc.SetFont("Times New Roman", 16)
Doc.TextUnderline = True
Doc.WriteRTF(Doc.nLeft, Doc.nBottom, Doc.nRight, Doc.nFree,
     "Hello \b World!")
```

Produces the following output:

Hello **World!**

<u>Hello **World!**</u>

See also:

"RTF - Rich Text Format" in the Programmer's Manual

19.2 WriteBoxRTF

Same as WriteRTF() 613, but pen 441- and box 466-settings are used. Outputs RTF text within a rectangle at position *Left, Top* with the right border at *Right* and the bottom border at *Bottom*.

```
method VpeCoord VPE.WriteBoxRTF(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string Text
)
```

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

string Text
the string to output

Returns:
the bottom y-coordinate generated by the output

Remarks:
VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

In case of an error, LastError 201 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 520 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

19.3 WriteRTFFile

Outputs RTF text, which is read from the file specified in *FileName,* within a rectangle at position *Left, Top* with the right border at *Right* and the bottom border at *Bottom*. This is very useful for processing text already created by your end user with an RTF editor.

The pen is invisible.

```
method VpeCoord VPE.WriteRTFFile(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string FileName
)
```

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

string FileName
the file with RTF text that is imported

Returns:
the bottom y-coordinate generated by the output

Remarks:
VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

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 201 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 520 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

19.4 WriteBoxRTFFile

Same as WriteRTFFile() 616, but pen 441- and box 466-settings are used. Outputs RTF text, which is read from the file specified in *FileName,* within a rectangle at position *Left, Top* with the right border at *Right* and the bottom border at *Bottom*. This is very useful for processing text already created by your end user with an RTF editor.

method VpeCoord VPE.WriteBoxRTFFile(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    string *FileName*
)

VpeCoord *Left, Top, Right, Bottom*
    position and dimensions

string FileName
    the file with RTF text that is imported

Returns:
    VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

    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 201 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 520 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

19.5 WriteRTFStream

Outputs RTF text, which is read from the supplied stream, within a rectangle at position *Left, Top* with the right border at *Right* and the bottom border at *Bottom*. This is very useful for processing rich text stored in a database.

The pen is invisible.

method VpeCoord VPE.WriteRTFStream(
  TVPEStream Stream,
  VpeCoord *Left*,
  VpeCoord *Top*,
  VpeCoord *Right*,
  VpeCoord *Bottom*
)

TVPEStream Stream
  stream object

VpeCoord *Left, Top, Right, Bottom*
  position and dimensions

Returns:
  the bottom y-coordinate generated by the output

Remarks:
  VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

  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 201 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 520 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

19.6 WriteBoxRTFStream

Same as WriteRTFStream() 618, but pen 441- and box 466-settings are used. Outputs RTF text, which is read from the supplied stream, within a rectangle at position *Left, Top* with the right border at *Right* and the bottom border at *Bottom*. This is very useful for processing rich text stored in a database.

method VpeCoord VPE.WriteBoxRTFStream(
TVPEStream Stream,
VpeCoord *Left*,
VpeCoord *Top*,
VpeCoord *Right*,
VpeCoord *Bottom*
)

TVPEStream Stream
stream object

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

Returns:
VFREE: only the Bottom coordinate may be set to VFREE, not the Right coordinate.

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 201 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 520 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

19.7 SetRTFFont

Modify the build-in RTF font table.

method void VPE.SetRTFFont(
    long *ID*,
    string *Name*
)

long ID
    the entry ID

string Name
    the name of the font that shall be assigned to the given entry-id

Remarks:

The current value of the CharSet 481 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:

ActiveX / VCL:

```
Doc.CharSet = SYMBOL_CHARSET
Doc.SetRTFFont(1, "Wingdings")
```

.NET:

```
Doc.CharSet = CharSet.Symbol
Doc.SetRTFFont(1, "Wingdings")
```

Or you can add as many fonts to the table as you like, for example:

ActiveX / VCL:

```
Doc.CharSet = SYMBOL_CHARSET
Doc.SetRTFFont(3, "Wingdings")
```

.NET:

```
Doc.CharSet = CharSet.Symbol
Doc.SetRTFFont(1, "Wingdings")
```

See also:

"RTF - Rich Text Format" in the Programmer's Manual

19.8 SetRTFColor

Modify the build-in RTF color table.

```
method void VPE.SetRTFColor(
        long ID,
        Color Color
)
```

long ID
the entry ID

Color Color
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure
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 case you are using 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

ActiveX / VCL

```
Doc.SetRTFColor(1, RGB(10, 10, 10))
```

.NET

```
Doc.SetRTFColor(1, Color.FromArgb(10, 10, 10))
```

Or you can add as many colors to the table as you like, for example:

ActiveX / VCL

```
Doc.SetRTFColor(31, COLOR_GREEN)
```

.NET

```
Doc.SetRTFColor(31, Doc.COLOR_GREEN)
or
Doc.SetRTFColor(31, Color.Green
```

See also:

"RTF - Rich Text Format" in the Programmer's Manual

19.9 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 setting for 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

19.10 FirstIndent

RTF: First-Line indent. Specifies the left indent of the first line of a new paragraph.

property VpeCoord **VPE.FirstIndent**

write; runtime only

Possible Values:

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:
Doc.FirstIndent = 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:

```
Doc.FirstIndent = -1
Doc.LeftIndent = 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

19.11 LeftIndent

RTF: Left indent. Specifies the left indent of all lines of a new paragraph.

property VpeCoord **VPE.LeftIndent**

write; runtime only

Possible Values:
the left indent

Default:
0

Remarks:
The left indent is relative to the left border of the object's rectangle. For example:
Doc.LeftIndent = 2 means, that the left margin is set to:

[left border of object rectangle] + 2 cm

The first-line indent 624 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:

```
Doc.FirstIndent = -1
Doc.LeftIndent = 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

19.12 RightIndent

RTF: Right indent. Specifies the right indent of all lines of a new paragraph.

property VpeCoord **VPE.RightIndent**

write; runtime only

Possible Values:
the right indent

Default:
0

Remarks:
The right indent is relative to the right border of the object’s rectangle. For example: Doc.RightIndent = 2 means, that the right margin is set to:

[right border of object rectangle] - 2 cm

Example:

```
Doc.RightIndent = 2
```

See also:
"RTF - Rich Text Format" in the Programmer's Manual

19.13 SpaceBefore

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).

property VpeCoord **VPE.SpaceBefore**

write; runtime only

Possible Values:
the space before a new paragraph

Default:
0

See also:
"RTF - Rich Text Format" in the Programmer's Manual

19.14 SpaceAfter

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.

property VpeCoord **VPE.SpaceAfter**

write; runtime only

Possible Values:
the space after a paragraph

Default:
0

See also:
"RTF - Rich Text Format" in the Programmer's Manual

19.15 SpaceBetween

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.

property VpeCoord **VPE.SpaceBetween**

write; runtime only

Possible Values:
the space between paragraphs

Default:
0

See also:
"RTF - Rich Text Format" in the Programmer's Manual

19.16 DefaultTabSize

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.

property VpeCoord **VPE.DefaultTabSize**

write; runtime only

Possible Values:
the default TAB advance width

Default:
1.25 (= 1.25 cm)

See also:
"RTF - Rich Text Format" in the Programmer's Manual

19.17 SetTab

Sets an individual RTF tab position. You may specify as many tab positions as you like.

method void VPE.SetTab(
    VpeCoord *TabPosition*,
    long *Reserved*
)

VpeCoord *TabPosition*
    tab position

long Reserved
    for future use and should be set to zero

Example:

```
Doc.SetTab(1)
Doc.SetTab(2)
Doc.SetTab(5)
```

Sets three individual tab positions at 1cm, 2cm and 5cm

```
Doc.ClearTab(2)
```

Removes the tab-stop at position 2cm (which was previously set by SetTab()).

See also:
    "RTF - Rich Text Format" in the Programmer's Manual

19.18 ClearTab

Removes an individual RTF tab position.

method void VPE.ClearTab(
    VpeCoord *TabPosition*
)

VpeCoord *TabPosition*
    tab position

Example:

```
Doc.SetTab(1)
Doc.SetTab(2)
Doc.SetTab(5)
```

Sets three individual tab positions at 1cm, 2cm and 5cm

```
Doc.ClearTab(2)
```

Removes the tab-stop at position 2cm (which was previously set by SetTab() 631).

See also:
"RTF - Rich Text Format" in the Programmer's Manual

19.19 ClearAllTabs

Removes all RTF tab settings, which had been set previously by SetTab() 631.

method void VPE.ClearAllTabs(
)

See also:

"RTF - Rich Text Format" in the Programmer's Manual

19.20 ResetParagraph

Resets all RTF paragraph settings to their default values.

```
method void VPE.ResetParagraph(
)
```

See also:

"RTF - Rich Text Format" in the Programmer's Manual

19.21 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 364 is set to AUTO_BREAK_ON or AUTO_BREAK_FULL.

The RTF properties KeepLines 636, KeepNextParagraph 637 and ParagraphControl 638 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

19.22 KeepLines

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.

property boolean VPE.KeepLines

write; runtime only

Possible Values:

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 KeepNextParagraph 637)

KeepLines, KeepNextParagraph and ParagraphControl 638 may be used together.

See also:

"RTF - Rich Text Format" in the Programmer's Manual

19.23 KeepNextParagraph

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.

property boolean VPE.KeepNextParagraph

write; runtime only

Possible Values:

Value	Description
True	yes
False	no

Default:
False

Remarks:
Note: Page breaks may occur between the lines of paragraphs (see KeepLines 636).

KeepLines, KeepNextParagraph and ParagraphControl 638 may be used together.

See also:
"RTF - Rich Text Format" in the Programmer's Manual

19.24 ParagraphControl

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 636, KeepNextPar and ParControl may be used together.

property boolean VPE.ParagraphControl

write; runtime only

Possible Values:

Value	Description
True	yes
False	no

Default:
False

Remarks:
KeepLines 636, KeepNextParagraph 637 and ParagraphControl may be used together.

See also:
"RTF - Rich Text Format" in the Programmer's Manual

Clickable Objects

20 Clickable Objects

[GUI Control 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 DoObjectClicked() 153 (VCL: OnObjectClicked() 179, .NET: ObjectClicked Event 73) 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.

20.1 EnableClickEvents

[GUI Control 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.

property boolean VPE.EnableClickEvents

write; runtime only

Possible Values:

Value	Description
True	enable
False	disable

Default:
True

20.2 ObjectID

[GUI Control 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 DoObjectClicked 153 (VCL: OnObjectClicked 179, .NET: ObjectClicked 73) 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.

property long VPE.ObjectID

read / write; runtime only

Possible Values:
an Object ID to identify the clicked object when processing the DoObjectClicked() event (VCL: OnObjectClicked(), .NET: ObjectClicked())

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

Remarks:
You can read the value of this property. It is only accessible for reading while your application is processing the event DoObjectClicked() (VCL: OnObjectClicked(), .NET: ObjectClicked()). Reading this property while not processing the event will return invalid - and therefore useless - data.

Example:

```
Doc.ObjectID = 1
Doc.Print(1, 1, "Monthly Report")
```

Assigns the ObjectID 1 to the text object "Monthly Report".

```
Doc.ObjectID = 2
Doc.Picture(1, 1, 3, 3, "data.bmp")
```

Assigns the ObjectID 2 to the image "data.bmp".

```
Doc.ObjectID = 0
Doc.Print(1, Doc.nBottom + 30, "This image shows data.")
```

Assigns NO ObjectID to the text object "This image shows data.". Therefore this object is not clickable.

Processing of the DoObjectClicked() event in Visual Basic:

```
Private Sub Doc_DoObjectClicked(ByVal ObjectID As Long)
   MsgBox "Object #" + Str(ObjectID) + "was clicked"
End Sub
```

The event handler in the example above will show a message box with the Object ID of the clicked object.

20.3 ClickedObject

[GUI Control Only, Enterprise Edition and above]

Returns the VPE Object 884 - instead of the ObjectID 642, which is only an integer - of the object that fired the DoObjectClicked 153 (VCL: OnObjectClicked 179, .NET: ObjectClicked 73) event.

property TVPEObject VPE.ClickedObject

read; runtime only

Remarks:

This property is only accessible while your application is processing the DoObjectClicked() (VCL: OnObjectClicked(), .NET: ObjectClicked()) event.

Reading this property while not processing the event, will throw an exception.

See also:

"Important Note for VPE-VCL Users" in the Programmer's Manual

UDO - User Defined Objects

21 UDO - User Defined Objects

[Professional Edition and above, not available for Java, PHP, Python, Ruby]

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 466-Object (see "The Object-Oriented Style" in the Programmer's Manual). It inherits all properties of the Box object, including BkgMode 451, PenSize 443, PenStyle 444, 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 DoUDOPaint() 154 (VCL: OnUDOPaint() 180, .NET: UDOPaint Event 74). 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 648, which can be assigned an OLE / COM object so it is painted automatically into the rectangle of the UDO!

21.1 CreateUDO

[Professional Edition and above, not available for Java, PHP, Python, Ruby]

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 DoUDOPaint() 154 (VCL: OnUDOPaint() 180, .NET: UDOPaint() 74).

method void VPE.CreateUDO(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*,
    long *lParam*
)

VpeCoord *Left, Top, Right, Bottom*
position and dimensions of the UDO

long lParam
numeric ID to identify the UDO during the DoUDOPaint() event (VCL: OnUDOPaint(), .NET: UDOPaint())

Example:
see UDODC 650

Remarks:
Special care must be taken, when creating VPE Document files that contain pictures 520 or UDO's.
For details please see WriteDoc 225.

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.

21.2 UDOPicture

[Professional Edition and above, ActiveX only]

This property is only accessible while you are processing the event DoUDOPaint() 154 (VCL: OnUDOPaint() 180, .NET: UDOPaint() 74). Accessing this porperty while not processing the event will return invalid - and therefore useless - data.

property IPictureDisp VPE.UDOPicture

write; runtime only

Possible Values:

Picture datatype offered by OLE / COM objects (of type IPictureDisp)

Example:

ActiveX - Painting an OLE Object with UDO (example in Visual Basic):

First, insert an OLE Control named OLE1 into a form. For example a Microsoft Excel Spreadsheet or a Corel Draw object.

Then enter the following code:

```
Private Sub Form_Load()
   rem Make sure, the server application is running by executing the
   rem following code:
   OLE1.DoVerb (vbOLEShow)
   OLE1.Visible = False

   Doc.OpenDoc
   Call Doc.CreateUDO(1, 1, -18, -18, 1)
   Call Doc.Preview
End Sub

Private Sub Doc_DoUDOPaint ()
   If Doc.UDOlParam = 1 Then
         Doc.UDOPicture = OLE1.Picture
   End If
End Sub
```

21.3 UDOlParam

[Professional Edition and above, not available for Java, PHP, Python, Ruby]

Sets / returns the long integer value lParam from a UDO.

property long VPE.UDOlParam

read; runtime only
also supported by TVPEObject: read / write; runtime only

Returns:

the id of the UDO that needs to be drawn

Remarks:

Only in case you are using the property of the VPE Document Object (ActiveX: VpeControl and VCL: TVPEngine): This property is only accessible while you are processing the event DoUDOPaint() 154 (VCL: OnUDOPaint() 180, .NET: UDOPaint() 74). Reading this property while not processing the event will return invalid - and therefore useless - data. It returns the long integer value lParam you had specified when creating the UDO (by calling CreateUDO() 647). The value returned should be used to identify the object which needs to be drawn.

Example:

see UDODC 650

21.4 UDODC

[Professional Edition and above, not available for Java, PHP, Python, Ruby. Not available for .NET, see UDOGraphics 652]

This property is only accessible while you are processing the event DoUDOPaint() 154 (VCL: OnUDOPaint() 180, .NET: UDOPaint() 74). 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.

property long VPE.UDODC

read; runtime only

Returns:
the HDC of the UDO that has to be painted

Remarks:
Before VPE fires the UDOPaint - event, it draws the Box 466 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:

```
Doc.CreateUDO(1, 1, -18, -18, 1);
```

Creates a UDO with the current properties of the Box-Object and the ID (=lParam) "1"

Processing of the DoUDOPaint() event in Visual Basic:

```
Make the following declarations in a global module:
Declare Sub MoveToEx Lib "gdi32" (ByVal hDC As Long, ByVal x As Long,
                                  ByVal y As Long,
                                  ByVal dummy As Long)
Declare Sub LineTo Lib "gdi32" (ByVal hDC As Long, ByVal x As Long,
                                  ByVal y As Long)
```

Create a UDO with:

```
Private Sub Form_Load()
   Doc.OpenDoc
   Call Doc.CreateUDO(1, 1, -18, -18, 1)
   Call Doc.Preview
End Sub
```

The event handler itself:

```
Private Sub Doc_DoUDOPaint()
   rem Windows GDI LineTo() draws a line from the current position up
to,
   rem but not including, the specified point.
   rem Therefore we add 1 to each coordinate.

   If Doc.UDOlParam = 1 Then
        Call MoveToEx(Doc.UDODC, Doc.nUDOLeft, Doc.nUDOTop, 0)
        Call LineTo(Doc.UDODC, Doc.nUDORight + 1, Doc.nUDOBottom + 1)
        Call MoveToEx(Doc.UDODC, Doc.nUDORight, Doc.nUDOTop, 0)
        Call LineTo(Doc.UDODC, Doc.nUDOLeft - 1, Doc.nUDOBottom + 1)
   End If
End Sub
```

The example draws two crossing lines within the UDO.

21.5 UDOGraphics

[.NET only]

This property is only accessible while you are processing the event DoUDOPaint() 154 (VCL: OnUDOPaint() 180, .NET: UDOPaint() 74). Reading this property while not processing the event will return a null value.

It returns the current GDI+ drawing surface, which VPE is painting on (this maybe the screen or a printer). Using this GDI+ drawing surface, you can call any GDI+ method or property and paint yourself onto the drawing surface.

property Graphics VPE.UDOGraphics

read; runtime only

Returns:
the Graphics object of the UDO that has to be painted

Remarks:
Before VPE fires the UDOPaint() event, it draws the Box 466 object the UDO object is inherited from.

Example:

```
Doc.CreateUDO(1, 1, -18, -18, 1);
```

Creates a UDO with the current properties of the Box-Object and the ID (=lParam) "1"

Processing of the UDOPaint() event in C#: creating UDO's

```
private void Form1_Load(object sender, System.EventArgs e)
{
     Doc.OpenDoc();
     Doc.PenSize = 0;    // no frame
     Doc.CreateUDO(1, 1, -11.5, -1, 1);
     Doc.CreateUDO(1, 3, -11.5, -1, 2);
     Doc.Preview();
}
```

The event handler for painting the content of the UDO:

```
private void Doc_UDOPaint(object sender, System.EventArgs e)
{
     int left = Doc.nUDOLeft;
     int top = Doc.nUDOTop;
     int width = Doc.nUDORight - vpe.nUDOLeft;
     int height = Doc.nUDOBottom - vpe.nUDOTop;

     Doc.UDOGraphics.DrawRectangle(
          new Pen(Color.Green, 3), left, top, width, height);

     if (Doc.UDOlParam == 1)
     {
          Doc.UDOGraphics.DrawString(
               "Hello, I am a UDO (User Defined Object).",
               new Font("Arial", 10, FontStyle.Bold),
               new SolidBrush(Color.Black),
               left, top);
     }
     else if (Doc.UDOlParam == 2)
     {
          Doc.UDOGraphics.DrawString(
               "Hello, I am UDO #2.",
               new Font("Arial", 12, FontStyle.Italic),
               new SolidBrush(Color.Green),
               left, top);
     }
}
```

The example draws the contents of two different UDO’s. You will notice that the same UDOPaint() event handler is called for each UDO that appears in the document. The UDO that needs to be drawn is identified by the property UDOlParam 649. When we had called CreateUDO() 647 we had specified for the first UDO for *UDOlParam* the value 1, and for the second the value 2.

21.6 UDOIsPrinting

[Professional Edition and above, not available for Java, PHP, Python, Ruby]

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 655 property for such a task. Do not use the property IsPrinting 312 because while the print job is running and *IsPrinting = true*, it can still happen that the UDO needs to be painted onto the screen surface, for example because you moved a window of another application over the preview of VPE.

property boolean VPE.UDOIsPrinting

read; runtime only

Returns:

Value	Description
True	yes, the UDO has to be drawn onto a printer's surface
False	no, the UDO has not to be drawn onto a printer's surface

Remarks:

This property is useful for the ActiveX and VCL. The .NET control sets the UDOGraphics 652.*ScaleTransform()* already accordingly to the output surface and any applied scaling factor.

21.7 UDOIsExporting

[Professional Edition and above, not available for Java, PHP, Python, Ruby]

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 654 property for such a task.

property boolean VPE.UDOIsExporting

read; runtime only

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

Remarks:

This property is useful for the ActiveX and VCL. The .NET control sets the UDOGraphics 652.*ScaleTransform()* already accordingly to the output surface and any applied scaling factor.

21.8 UDODpiX

[Professional Edition and above, not available for Java, PHP, Python, Ruby]

Returns the X-resolution of the UDO's current output surface in DPI (Dots Per Inch).

property boolean VPE.UDODpiX

read; runtime only

Returns:

The X-resolution of the UDO's current output surface in DPI (Dots Per Inch).

Remarks:

This property is useful for the ActiveX and VCL. The .NET control sets the UDOGraphics 652.*ScaleTransform()* already accordingly to the output resolution.

21.9 UDODpiY

[Professional Edition and above, not available for Java, PHP, Python, Ruby]

Returns the Y-resolution of the UDO's current output surface in DPI (Dots Per Inch).

property boolean VPE.UDODpiY

read; runtime only

Returns:

The Y-resolution of the UDO's current output surface in DPI (Dots Per Inch).

Remarks:

This property is useful for the ActiveX and VCL. The .NET control sets the UDOGraphics 652 .*ScaleTransform()* already accordingly to the output resolution.

21.10 nUDOLeft, nUDOTop, nUDORight, nUDOBottom

[Professional Edition and above, not available for Java, PHP, Python, Ruby]

Return each coordinate of the UDO rectangle (this is exactly the area you may paint in).

The properties are only accessible while you are processing the event DoUDOPaint() 154 (VCL: OnUDOPaint() 180, .NET: UDOPaint() 74). Retrieving the values while not processing the event will return invalid - and therefore useless - data.

property VpeCoord VPE.nUDOLeft
property VpeCoord VPE.nUDOTop
property VpeCoord VPE.nUDORight
property VpeCoord VPE.nUDOBottom

read; runtime only

Returns:
the coordinates of the UDO which fired the currently processed UDO-Paint-Event

Remarks:
In contrast to the other n-Properties the nUDO-Properties 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 PictureExport 660, too!).

Picture Export Functions

22 Picture Export Functions

[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 544 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.

22.1 JpegExportOptions

[Windows Platform Only, Professional Edition and above]

Specifies options for exported JPEG images.

property JpegExportOptions [long] VPE.JpegExportOptions

read, write; runtime only

Possible Values:

several options

ActiveX / VCL	Value	Enum	Comment
PICEXP_JPEG_DEFAULT	0	Default	Default, saves with good quality (75:1)
PICEXP_JPEG_HI_QUALITY	128	HiQuality	Saves with superb quality (100:1)
PICEXP_JPEG_GOOD_QUALITY	256	GoodQuality	Saves with good quality (75:1)
PICEXP_JPEG_MID_QUALITY	512	MidQuality	Saves with normal quality (50:1)
PICEXP_JPEG_LO_QUALITY	1024	LoQuality	Saves with average quality (25:1)
PICEXP_JPEG_BAD_QUALITY	2048	BadQuality	Saves with bad quality (10:1)
Integer X in [0..100]	0 – 100	n/a	Save with quality X:1
PICEXP_JPEG_PROGRESSIVE	8192	Progressive	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:

```
Doc.JpegExportOptions = PICEXP_JPEG_DEFAULT + PICEXP_JPEG_PROGRESSIVE
```

22.2 TiffExportOptions

[Windows Platform Only, Professional Edition and above]

Specifies options for exported TIFF images.

property TiffExportOptions [long] VPE.TiffExportOptions

read, write; runtime only

Possible Values:

several options

ActiveX / VCL	Value	Enum	Comment
PICEXP_TIFF_DEFAULT	0	Default	Default, save using CCITTFAX4 compression for 1-bit bitmaps and LZW compression for any other bitmaps
PICEXP_TIFF_PACKBITS	256	Packbits	Save using PACKBITS compression
PICEXP_TIFF_DEFLATE	512	Deflate	Save using DEFLATE compression (a.k.a. ZLIB compression)
PICEXP_TIFF_ADOBE_DEFLATE	1024	AdobeDeflate	Save using ADOBE DEFLATE compression
PICEXP_TIFF_NONE	2048	None	Save without any compression
PICEXP_TIFF_CCITTFAX3	4096	Ccittfax3	Save using CCITT Group 3 fax encoding
PICEXP_TIFF_CCITTFAX4	8192	Ccittfax4	Save using CCITT Group 4 fax encoding
PICEXP_TIFF_LZW	16384	Lzw	Save using LZW compression
PICEXP_TIFF_JPEG	32768	Jpeg	Save using JPEG compression (8-bit greyscale and 24-bit only. Default to LZW for other bitdepths.)
PICEXP_TIFF_APPEND	1073741824	Append	Append file to multi-page TIFF

Default:

PICEXP_TIFF_DEFAULT

Remarks:

The flag PICEXP_TIFF_APPEND can be combined with other flags.

Example:

```
Doc.TiffExportOptions = PICEXP_TIFF_DEFAULT + PICEXP_TIFF_APPEND
```

22.3 BmpExportOptions

[Windows Platform Only, Professional Edition and above]

Specifies options for exported BMP images.

property BmpExportOptions [long] VPE.BmpExportOptions

read, write; runtime only

Possible Values:

several options

ActiveX / VCL	Value	Enum	Comment
PICEXP_BMP_DEFAULT	0	Default	Default, save without any compression
PICEXP_BMP_RLE	1	Rle	Save RLE compressed

Default:

PICEXP_BMP_DEFAULT

Example:

```
Doc.BmpExportOptions = PICEXP_BMP_RLE
```

22.4 PnmExportOptions

[Windows Platform Only, Professional Edition and above]

Specifies options for exported PBM, PGM, PPM images.

property PnmExportOptions [long] VPE.PnmExportOptions

read, write; runtime only

Possible Values:

several options

ActiveX / VCL	Value	Enum	Comment
PICEXP_PNM_DEFAULT	0	Default	PBM, PGM, PPM files are written RAW
PICEXP_PNM_ASCII	1	Ascii	PBM, PGM, PPM files are written ASCII

Default:

PICEXP_PNM_DEFAULT

Example:

```
Doc.PnmExportOptions = PICEXP_PNM_ASCII
```

22.5 GifExportOptions

[Windows Platform Only, Professional Edition and above]

Specifies options for exported GIF images.

property GifExportOptions [long] VPE.GifExportOptions

read, write; runtime only

Possible Values:

several options

ActiveX / VCL	Value	Enum	Comment
PICEXP_GIF_DEFAULT	0	Default	nothing special
PICEXP_GIF_APPEND	1073741824	Append	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:

```
Doc.GifExportOptions = PICEXP_GIF_DEFAULT + PICEXP_GIF_APPEND
```

22.6 PictureExportColorDepth

[Windows Platform Only, Professional Edition and above]

Specifies the color depth of an exported image.

property long VPE.PictureExportColorDepth

read; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
PICEXP_COLOR_MONO	1	ColorMono	
PICEXP_COLOR_16	4	Color16	
PICEXP_COLOR_256	8	Color256	
PICEXP_COLOR_HI	16	ColorHi	
PICEXP_COLOR_TRUE	24	ColorTrue	

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 667

22.7 PictureExportDither

[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.

property long VPE.PictureExportDither

read; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
PICEXP_DITHER_NONE	0	DitherNone	
PICEXP_DITHER_MONO	1	DitherMono	
PICEXP_DITHER_256	3	Dither256	
PICEXP_DITHER_256_WU	4	Dither256Wu	

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 666 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 666 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:

ActiveX / VCL:

```
Doc.PictureExportColorDepth = PICEXP_COLOR_256
Doc.PictureExportDither = PICEXP_DITHER_MONO
Doc.PictureExportPage("test.bmp", 1)
```

.NET:

```
Doc.PictureExportColorDepth = PictureExportColorDepth.Color256
Doc.PictureExportDither = PictureExportDither.DitherMono
Doc.PictureExportPage("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.

22.8 PictureExportPage

[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 666 and PictureExportDither 667 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 524.

The resolution of the image is defined with DefaultPictureDPI() 532.

method boolean VPE.PictureExportPage(
 string *FileName*,
 long *PageNo*
)

string FileName
(path- and) filename of exported image

long PageNo
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 201 is set.

During export, the BusyProgressBar 263 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:

ActiveX / VCL:
```
Doc.PictureExportColorDepth = PICEXP_COLOR_256
Doc.PictureExportDither = PICEXP_DITHER_MONO
Doc.SetDefaultPictureDPI(300, 300)
Doc.PictureExportPage("test.bmp", 1)
```

.NET:

```
Doc.PictureExportColorDepth = PictureExportColorDepth.Color256
Doc.PictureExportDither = PictureExportDither.DitherMono
Doc.SetDefaultPictureDPI(300, 300)
Doc.PictureExportPage("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:

ActiveX / VCL:

```
// Export as True-Color image
Doc.PictureExportColorDepth = 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:
Doc.PictureType = PIC_TYPE_PNG

// PNG is always written with Deflate (ZLib) compression

// Set 96 DPI resolution:
Doc.SetDefaultPictureDPI(96, 96)
Doc.PictureExportPage("image.001", 2);
```

.NET:

```
// Export as True-Color image
Doc.PictureExportColorDepth = PictureExportColorDepth.ColorTrue

// 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:
Doc.PictureType = PictureType.PNG

// Select ZIP compression:
Doc.PictureExportOptions = PictureExportOptions.WriteCompressed

// Set 96 DPI resolution:
Doc.SetDefaultPictureDPI(96, 96)
Doc.PictureExportPage("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:

ActiveX / VCL:

```
// Export as dithered Black and White image:
Doc.PictureExportDither = PICEXP_DITHER_MONO

```

```
// Export as TIFF image:
Doc.PictureType = PIC_TYPE_TIFF

// Select FaxG4 as export format:
Doc.PictureExportOptions = PICEXP_TIFF_CCITTFAX4

// Set 200 DPI resolution:
Doc.SetDefaultPictureDPI(200, 200)
Doc.PictureExportPage("image.002", 3)
```

.NET:

```
// Export as dithered Black and White image:
Doc.PictureExportDither = PictureExportDither.Mono

```

```
// Export as TIFF image:
Doc.PictureType = PictureType.TIFF

// Select FaxG4 as export format:
Doc.PictureExportOptions = PictureExportOptions.TIFF_CCITT_Fax4

// Set 200 DPI resolution:
Doc.SetDefaultPictureDPI(200, 200)
Doc.PictureExportPage("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".

22.9 PictureExportPageStream

[Windows Platform Only, Professional Edition and above]

Identical to PictureExportPage() 669, but exports the picture to a stream.

```
method boolean VPE.PictureExportPageStream(
        TVPEStream stream,
        long PageNo
)
```

TVPEStream stream
The stream-object where the picture is written to. The stream must have been created before by calling CreateMemoryStream() 694.

long PageNo
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.

22.10 PictureExport

[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 666 and PictureExportDither 667 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 524.

The resolution of the image is defined with DefaultPictureDPI() 532.

method boolean VPE.PictureExport(
    string *FileName*,
    long *PageNo*,
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*
)

string FileName
    (path- and) filename of exported image

long PageNo
    page number of the page that shall be exported from the document

VpeCoord *Left, Top, Right, Bottom*
    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 201 is set.

During export, the BusyProgressBar 263 is shown.

For *Left, Top, Right, Bottom* you can also specify the V-Flags and for *Right, Bottom* 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:

ActiveX / VCL:
```
Doc.PictureExportColorDepth = PICEXP_COLOR_256
Doc.PictureExportDither = PICEXP_DITHER_MONO
Doc.SetPictureDefaultDPI(300, 300)
Doc.PictureExport("test.bmp", 1, 1, 1, -5, -5)
```

.NET:
```
Doc.PictureExportColorDepth = PictureExportColorDepth.Color256
Doc.PictureExportDither = PictureExportDither.DitherMono
Doc.SetPictureDefaultDPI(300, 300)
Doc.PictureExport("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".

ActiveX / VCL:
```
Doc.PictureExport("test.bmp", 1, VLEFT, VTOP, VRIGHT, VBOTTOM)
```

.NET:
```
Doc.PictureExport("test.bmp", 1, Doc.nLeft, Doc.nTop, Doc.nRight,
                  Doc.nBottom)
```

Instructs VPE to export the last inserted object to an image file.

22.11 PictureExportStream

[Windows Platform Only, Professional Edition and above]

Identical to PictureExport() 673, but exports the picture to a stream.

method int VPE.PictureExportPageStream(
  TVPEStream *stream*,
  long *PageNo*,
  VpeCoord *Left*,
  VpeCoord *Top*,
  VpeCoord *Right*,
  VpeCoord *Bottom*
)

TVPEStream stream
  The stream-object where the picture is written to. The stream must have been created before by calling CreateMemoryStream() 694.

long PageNo
  page number of the page that shall be exported from the document

VpeCoord *Left, Top, Right, Bottom*
  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.

This page is intentionally left blank.

Memory Streams

23 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.

Background:

For export, the basic idea is to write a VPE (or PDF, HTML, etc.) document to a memory stream first, using WriteDocStream(stream). In order to be able to access the data of the memory stream (for sending it to a client's browser or to store it as BLOB in a database), you need to use TVPEStream.Read() afterwards. This will copy the data from the stream to a buffer of your application.

Note that you can also export pages as images like TIF, JPG, etc. to memory streams using equivalently the methods PictureExportStream() and PictureExportPageStream().

Vice versa for import, the basic idea is to write a VPE document as BLOB from a database (or from anywhere else, e.g. a network stream) to a memory stream first, using TVPEStream.Write(). This will copy the data from a buffer of your application to the stream. In order to import the document from the stream into VPE, you need to use ReadDocStream(stream) afterwards.

Note that you can also read (import) images from memory streams using equivalently the methods PictureStream() or RenderPictureStream(). And you can also read (import) RTF (Rich Text) from memory streams using WriteRTFStream(), WriteBoxRTFStream(), RenderRTFStream() and RenderBoxRTFStream(). Please don’t be confused that the RTF-methods begin with the word "write", this will not write to the stream, but create the RTF objects from the given stream. The names have been chosen accordingly to the RTF-methods which create RTF objects from strings.

Important:

After any write or read operation to a memory stream, the position of the internal memory stream pointer is, where the last read or write operation did stop. E.g. after calling WriteDocStream(), a ReadDocStream() on the written stream will fail. You must call Seek(0) first, to position the internal memory stream pointer at the beginning of the memory stream, before executing ReadDocStream().

The same is true, if you wish to read data from a memory stream using TVPEStream.Read() – where the memory stream has been filled before by WriteDocStream().
Call Seek(0) first!

This applies to all read and write operations, i.e. PictureStream(), etc.

Example:

```
Vpe.OpenDoc()

// Write the current document to a memory stream
TVPEStream Stream = Vpe.CreateMemoryStream(0)
Vpe.WriteDocStream(Stream)

// Create a second document
Vpe2.OpenDoc()

// Read the memory stream into the new document,
// seek to position 0 first!
Stream.Seek(0);
Vpe.ReadDocStream(Stream);

// Cleanup
Stream.Close();
```

See also:

WriteDocStream 227
WriteDocStreamPageRange 228

ReadDocStream 231
ReadDocStreamPageRange 232

PictureStream 539
RenderPictureStream 428

PictureExportStream 675
PictureExportPageStream 672

WriteRTFStream 618
WriteBoxRTFStream 619
RenderRTFStream 436
RenderBoxRTFStream 437

23.1 CreateMemoryStream

[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.

This is a method of the VPE document. All other methods and properties described throughout this chapter belong to the TVPEStream object, which is created by this method.

method TVPEStream VPE.CreateMemoryStream(
    long *chunk_size*
)

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 TVPEStream object for the created memory stream.

In case of an error an exception is thrown and LastError 201 is set.

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). Neither calling CloseDoc() 193, nor destroying a TVPEStream object will close a stream.

Returns:
sets LastError 201

23.2 Close

[Professional Edition and above]

Closes the given stream.

```
method void TVPEStream.Close(
)
```

Remarks:

When closing a stream, the internal stream handle is destroyed. You may NOT use this object in any further calls to a method, which takes a stream object as parameter.

23.3 Read

[Professional Edition and above]

Reads data from the supplied stream.

```
method int TVPEStream.Read(
        BYTE *buffer,
        int size
)
```

*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.

Remarks:
When reading from a stream, which was written by a VPE method, e.g. WriteDocStream() or PictureStream() etc., you need to call Seek(0) first before calling Read() in order to reset the internal stream pointer to the beginning of the stream.

Python:
The parameter *buffer* is of type *ctypes.c_char_p*. See *ctypes.create_string_buffer()* to allocate a buffer.

23.4 Write

[Professional Edition and above]

Writes data to the supplied stream.

```
method int TVPEStream.Write(
        BYTE *buffer,
        int size
)
```

*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:
The number of bytes written to the stream.

Remarks:
If you want to call a VPE method (for example ReadDocStream() or PictureStream()), after you have written data to a stream, you need to call Seek(0) first before calling Write() in order to reset the internal stream pointer to the beginning of the stream.

Python:
The parameter *buffer* is of type *ctypes.c_char_p*. See *ctypes.create_string_buffer()* to allocate a buffer.

23.5 Size

[Professional Edition and above]

Returns the size - in bytes - of the given stream.

property long TVPEStream.Size

read; runtime only

Returns:
The size of the stream in bytes.

23.6 IsEof

[Professional Edition and above]

Returns the End Of File status of the given stream.

This property returns true after the first read operation that attempts to read past the end of the file. IsEof continues to report true, until Seek() (see "Seek (ActiveX / VCL) 688" on page 11 688) is called.

property boolean TVPEStream.IsEof

read; runtime only

Returns:

Value	Description
True	EOF
False	not EOF

23.7 State

[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.

property boolean TVPEStream.State

read; runtime only

Returns:

Value	Description
True	status ok
False	failure

23.8 Position

[Professional Edition and above]

Returns the current position - in bytes - of the internal file pointer.

property long TVPEStream.Position

read; runtime only

Returns:

The current position within the stream.

23.9 Seek

[Professional Edition and above]

Moves the file pointer to the given position.

```
method void TVPEStream.Seek(
        long pos
)
```

long pos
    the new position

23.10 SeekEnd

[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.

```
method void TVPEStream.SeekEnd(
        long pos
)
```

long pos
the new position, must always be negative or null

23.11 SeekRel

[Professional Edition and above]

Moves the file pointer to the given position, which is relative to the current position.

```
method void TVPEStream.SeekRel(
        long offset
)
```

long pos
  the new position, relative to the current position

Memory Streams (.NET)

24 Memory Streams (.NET)

[Professional Edition and above; .NET only]

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.

This chapter describes the methods and functions for the VPE .NET component, where the stream class has been derived from System.IO.Stream and is therefore compatible to the stream classes of .NET.

Important:

After any write or read operation to a memory stream, the position of the internal memory stream pointer is, where the last read or write operation did stop. E.g. after calling WriteDocStream(), a ReadDocStream() on the written stream will fail. You must call Seek(0) first, to position the internal memory stream pointer at the beginning of the memory stream, before executing ReadDocStream().

The same is true, if you wish to read data from a memory stream using TVPEStream.Read() – where the memory stream has been filled before by WriteDocStream().
Call Seek(0) first!

This applies to all read and write operations.

Example:

```
Vpe.OpenDoc()

// Write the current document to a memory stream
TVPEStream Stream = Vpe.CreateMemoryStream(0)
Vpe.WriteDocStream(Stream)

// Create a second document
Vpe2.OpenDoc()

// Read the memory stream into the new document,
// seek to position 0 first!
Stream.Seek(0);
Vpe.ReadDocStream(Stream);

// Cleanup
Stream.Close();
```

See also:

WriteDocStream 227
WriteDocStreamPageRange 228

ReadDocStream 231

ReadDocStreamPageRange 232

PictureStream 539
RenderPictureStream 428

PictureExportStream 675
PictureExportPageStream 672

WriteRTFStream 618
WriteBoxRTFStream 619
RenderRTFStream 436
RenderBoxRTFStream 437

24.1 CreateMemoryStream (.NET)

[Professional Edition and above; .NET only]

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.

This is a method of the VPE document. All other methods and properties described throughout this chapter belong to the TVPEStream object, which is created by this method.

method TVPEStream VPE.CreateMemoryStream(
    long *chunk_size*
)

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 TVPEStream object for the created memory stream.

In case of an error an exception is thrown and LastError 201 is set.

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). Neither calling CloseDoc() 193, nor destroying a TVPEStream object will close a stream.

Remarks:
sets LastError 201

24.2 Close (.NET)

[Professional Edition and above; .NET only]

Closes the given stream.

```
method override void TVPEStream.Close(
)
```

Remarks:

When closing a stream, the internal stream handle is destroyed. You may NOT use this object in any further calls to a method, which takes a stream object as parameter.

24.3 Read (.NET)

[Professional Edition and above; .NET only]

Reads data from the supplied stream.

method override int TVPEStream.Read(
    [InAttribute] [OutAttribute] byte[] *buffer*,
    int *offset*,
    int *count*
)

byte[] buffer
  the buffer, to which the data will be copied from the stream

int offset
  offset into the buffer

int count
  the number of bytes that shall be read

Returns:
  The number of bytes read from the stream.

Remarks:
  When reading from a stream, which was written by a VPE method, e.g. WriteDocStream() or PictureStream() etc., you need to call Seek(0) first before calling Read() in order to reset the internal stream pointer to the beginning of the stream.

24.4 Write (.NET)

[Professional Edition and above; .NET only]

Writes data to the supplied stream.

```
method override void TVPEStream.Write(
        [InAttribute] [OutAttribute] byte[] buffer,
        int offset,
        int count
)
```

byte[] buffer
the buffer, from which the data will be copied to the stream

int offset
offset into the buffer

int count
the number of bytes that shall be written

Remarks:
If you want to call a VPE method (for example ReadDocStream() or PictureStream()), after you have written data to a stream, you need to call Seek(0) first before calling Write() in order to reset the internal stream pointer to the beginning of the stream.

24.5 WriteTo (.NET)

[Professional Edition and above; .NET only]

Writes the entire contents of this memory stream to another stream.

```
method virtual void TVPEStream.WriteTo(
        Stream stream
)
```

Stream stream
  The stream to write this memory stream to.

24.6 Length (.NET)

[Professional Edition and above; .NET only]

Returns the size - in bytes - of the given stream.

property override long TVPEStream.Length

read; runtime only

Returns:
The size of the stream in bytes.

24.7 IsEof (.NET)

[Professional Edition and above; .NET only]

Returns the End Of File status of the given stream.

This property returns true after the first read operation that attempts to read past the end of the file. IsEof continues to report true, until Seek() (see “Seek (.NET) 703” on page 25 703) is called.

property virtual boolean TVPEStream.IsEof

read; runtime only

Returns:

Value	Description
True	EOF
False	not EOF

24.8 State (.NET)

[Professional Edition and above; .NET only]

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.

property boolean TVPEStream.State

read; runtime only

Returns:

Value	Description
True	status ok
False	failure

24.9 Position (.NET)

[Professional Edition and above; .NET only]

Returns the current position - in bytes - of the internal file pointer.

property long TVPEStream.Position

read; runtime only

Returns:

The current position within the stream.

24.10 Seek (.NET)

[Professional Edition and above; .NET only]

Moves the file pointer to the given position.

method override long TVPEStream.Seek(
long *offset*,
SeekOrigin *loc*
)

long offset
The new position within the stream. This is relative to the loc parameter, and can be positive or negative.

SeekOrigin loc
A value of type SeekOrigin, which acts as the seek reference point.

Possible values are:

Value	Description
Begin	the beginning of a stream.
Current	Specifies the current position within a stream.
End	Specifies the end of a stream.

24.11 SetLength (.NET)

[Professional Edition and above; .NET only]

Not supported by TVPEStream, throws a *NotSupportedException*.

```
method override void TVPEStream.SetLength (
        long value
)
```

24.12 Flush (.NET)

[Professional Edition and above; .NET only]

Overridden, does nothing.

```
method override void TVPEStream.Flush (
)
```

24.13 CanRead (.NET)

[Professional Edition and above; .NET only]

Overridden, returns true.

property override bool CanRead

read; runtime only

24.14 CanSeek (.NET)

[Professional Edition and above; .NET only]

Overridden, returns true.

property override bool CanSeek

read; runtime only

24.15 CanWrite (.NET)

[Professional Edition and above; .NET only]

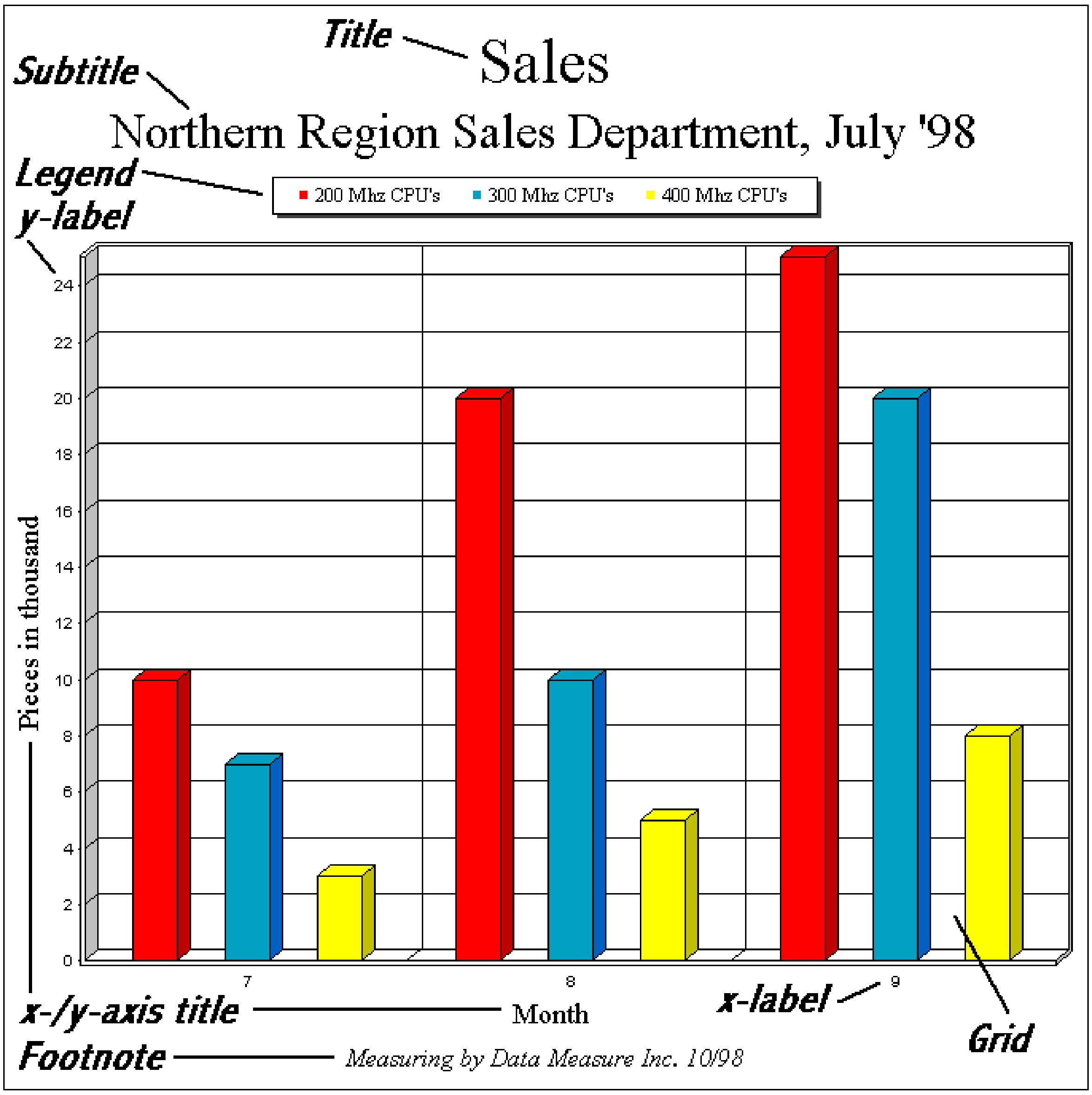
Overridden, returns true.

property override bool CanWrite

read; runtime only

Charts

25 Charts



The chart was created with the following short code sequence:

VCL:

```
hData: Pointer
hData = Doc.ChartDataCreate(3, 3)     // 3 Rows and 3 Columns

Doc.ChartTitle = "Sales"
Doc.ChartSubTitle = "Northern Region Sales Department, July '98"
Doc.ChartFootNote = "Measuring by Data Measure Inc. 10/98"
Doc.ChartLegendPosition = VCHART_LEGENDPOS_TOP

Doc.ChartDataSetXAxisTitle(hData, "Month")
Doc.ChartDataSetYAxisTitle(hData, "Pieces in thousand")
Doc.ChartDataAddLegend(hData, "200 Mhz CPU's")
Doc.ChartDataAddLegend(hData, "300 Mhz CPU's")
Doc.ChartDataAddLegend(hData, "400 Mhz CPU's")

Doc.ChartDataAddValue(hData, 0, 10)   // Row 0
Doc.ChartDataAddValue(hData, 0, 20)
Doc.ChartDataAddValue(hData, 0, 25)
Doc.ChartDataAddValue(hData, 1, 7)    // Row 1
Doc.ChartDataAddValue(hData, 1, 10)
Doc.ChartDataAddValue(hData, 1, 20)
Doc.ChartDataAddValue(hData, 2, 3)    // Row 2
Doc.ChartDataAddValue(hData, 2, 5)
Doc.ChartDataAddValue(hData, 2, 8)

Doc.ChartXLabelStartValue = 7
Doc.Chart(1, 1, -18, -18, hData, VCHART_3D_BAR)
```

All other component types (ActiveX, .NET, Java, …):

```
Dim Data as TVPEChartData
Data = Doc.ChartDataCreate(3, 3) // 3 Rows and 3 Columns

Doc.ChartTitle = "Sales"
Doc.ChartSubTitle = "Northern Region Sales Department, July '98"
Doc.ChartFootNote = "Measuring by Data Measure Inc. 10/98"
Doc.ChartLegendPosition = ChartLegendPosition.Top

Data.XAxisTitle = "Month"
Data.YAxisTitle = "Pieces in thousand"
Data.AddLegend("200 Mhz CPU's")
Data.AddLegend("300 Mhz CPU's")
Data.AddLegend("400 Mhz CPU's")

Data.AddValue(0, 10)    // Row 0
Data.AddValue(0, 20)
Data.AddValue(0, 25)
Data.AddValue(1, 7)     // Row 1
Data.AddValue(1, 10)
Data.AddValue(1, 20)
Data.AddValue(2, 3)     // Row 2
Data.AddValue(2, 5)
Data.AddValue(2, 8)

Doc.ChartXLabelStartValue = 7
Doc.Chart(1, 1, -18, -18, Data, ChartType.Bar3D)
```

25.1 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.

25.2 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

For .NET the Chart Data is encapsulated in the TVPEChartData object. For the ActiveX / VCL the Chart Data object is managed with a handle, i.e. a long integer which identifies the object within VPE.

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 544, etc. - inherited from the Box 466 Object (see "The Object-Oriented Style" in the Programmer's Manual), so the properties for a Box (like BkgMode 451, PenSize 443 and PenStyle 444, etc.) apply to the Chart Object.

25.3 ChartDataCreate

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 have assigned the same ChartData object, will display the same ChartData in a different style.

method TVPEChartData [pointer] VPE.ChartDataCreate(
 long *Columns*,
 long *Rows*
)

long Columns
 number of columns that shall be reserved

long Rows
 number of rows that shall be reserved

Returns:

VCL	the handle to the created ChartData object. In case of an error (out of memory) the handle is null (0).
ActiveX, .NET, Java, ...	a newly created TVPEChartData object reference. This object provides several methods and properties to add data to rows and columns, and to specify the color and appearance of each row. In case of an error (out of memory) the returned object reference is null.

Remarks:
The created ChartData object may only be used within the context of the document where it was created. Don’t use a ChartData object in a different document, this will cause a GPF (General Protection Fault).

Example:

VCL:
```
hData: Pointer
hData = Doc.ChartDataCreate(2, 4)
```

ActiveX, .NET, Java, …:
```
dim Data as TVPEChartData
Data = Doc.ChartDataCreate(2, 4)
```
Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

25.4 AddValue

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

Adds a new value to the specified column of the ChartData object.

method void TVPEChartData.AddValue(
int *Column*,
double *Value*
)

int Column
column, where to add the value

double Value
value to add

Example:

```
dim Data as TVPEChartData
Data = Doc.ChartDataCreate(2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

```
Data.AddValue(0, 10)
Data.AddValue(0, 20)
Data.AddValue(0, 30)
Data.AddValue(0, 40)
```

```
Data.AddValue(1, 5)
Data.AddValue(1, 10)
Data.AddValue(1, 15)
Data.AddValue(1, 20)
```

Now the internal data table of the Chart Data 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

25.5 AddLegend

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

Adds a new descriptive string for a column to the legend.

method void TVPEChartData.AddLegend(
    string *Legend*
)

string Legend
    legend string

Example:

```
dim Data as TVPEChartData
Data = Doc.ChartDataCreate(2, 4)
```

```
Data.AddLegend("Apples")
Data.AddLegend("Bananas")
```

Creates a ChartData object with two columns, titled "Apples" and "Bananas" in the legend.

25.6 XAxisTitle

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

Sets the x-axis title of the chart. This is the descriptive text of the x-axis.

property string TVPEChartData.XAxisTitle

write; runtime only

Possible Values:

title for the x-axis

25.7 YAxisTitle

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

Sets the y-axis title of the chart. This is the descriptive text of the y-axis.

property string TVPEChartData.YAxisTitle

write; runtime only

Possible Values:

title for the y-axis

25.8 AddXLabel

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

Adds a new X-Label to the chart. By default, the x-axis is automatically labeled. With ChartXLabelState() 771 you can switch to user defined labels and then add X-Labels to the chart. (see also ChartXLabelStartValue 777)

method void TVPEChartData.AddXLabel(
string *XLabel*
)

string XLabel
string of label to add

Remarks:
You can control if, how and on what position y-labels are drawn with the following properties:

- ChartXLabelState 771
- ChartXGridStep 766
- ChartXLabelStep 775

Example:

```
dim Data as TVPEChartData
Data = Doc.ChartDataCreate(2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

```
Data.AddValue(0, 10)
Data.AddValue(0, 20)
Data.AddValue(0, 30)
Data.AddValue(0, 40)

Data.AddValue(1, 5)
Data.AddValue(1, 10)
Data.AddValue(1, 15)
Data.AddValue(1, 20)

Data.AddLegend("Apples")
Data.AddLegend("Bananas")
```

Now the internal data table of the Chart Data 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

```
Doc.ChartXLabelState = ChartLabelState.User
Data.AddXLabel("1. Quarter")
Data.AddXLabel("2. Quarter")
Data.AddXLabel("3. Quarter")
Data.AddXLabel("4. Quarter")
```

the internal data table of the Chart Data 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

25.9 AddYLabel

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

Adds a new Y-Label to the chart. By default, the y-axis is labeled automatically. With ChartYLabelState 778 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.

```
method void TVPEChartData.AddYLabel(
        string YLabel
)
```

string YLabel
string of label to add

Remarks:
You can control if, how and on what position y-labels are drawn with the following properties:

- ChartYLabelState 778
- ChartYGridStep 767
- ChartYLabelStep 781
- ChartYLabelDivisor 782
- ChartDataSetMinimum 742
- ChartDataSetMaximum 743

25.10 SetColor

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

By default, VPE assigns itself default colors to each data column. With this method you can specify individual colors for each column.

```
method void TVPEChartData.SetColor(
        int Column,
        Color Color
)
```

int Column
column the color shall be assigned to

Color Color
any of the "COLOR_xyz" constants described in Programmer's Manual
or any member of the .NET Color structure

25.11 SetLineStyle

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

With this method you can specify individual line styles for each column.

method void TVPEChartData.SetLineStyle(
    int *Column*,
    PenStyle *LineStyle*
)

int Column
column the line style shall be assigned to

PenStyle [integer] LineStyle
possible values are:

Value	Comment
Solid	
Dash	-------
Dot	....
DashDot	_._._._
DashDotDot	_.._.._

Default:
PenStyle.Solid

Example:

```
Data.SetLineStyle(0, PenStyle.Solid)
```

25.12 SetHatchStyle

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

If you create a Bar-, Area- or Pie Chart from the Chart Data, you can specify individual hatch styles for each column with this method. By default, no hatching is used.

method void TVPEChartData.SetHatchStyle(
        int *Column*,
        HatchStyle *HatchStyle*
)

int Column
    column the line style shall be assigned to

HatchStyle HatchStyle
    possible values are:

Value	Comment
None	see HatchStyle 463 – possible styles
Horizontal	see HatchStyle – possible styles
Vertical	see HatchStyle – possible styles
ForwardDiagonal	see HatchStyle – possible styles
BackwardDiagonal	see HatchStyle – possible styles
Cross	see HatchStyle – possible styles
DiagonalCross	see HatchStyle – possible styles

Default:
    HatchStyle.None (no hatching is used)

Example:

```
Data.SetHatchStyle(0, HatchStyle.DiagonalCross)
```

25.13 SetPointType

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

If you create a Point Chart from the Chart Data, you can specify individual point styles for each column with this method.

method void TVPEChartData.SetPointType(
    int *Column*,
    ChartSymbol *PointType*
)

int Column
column the color shall be assigned to

ChartSymbol PointType
possible values are:

Value	Comment
None	
Square	
Triangle	
Circle	
Cross	
X	
Point	

Default:
ChartSymbol.X

Example:

```
Data.SetPointType(0, ChartSymbol.Square)
```

25.14 Minimum

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

VPE determines the minimum value of the numeric data stored in a Chart Data object itself. You can change the minimum value with this method.

property double TVPEChartData.Minimum

write; runtime only

Possible Values:

the minimum value of the numeric data of the Chart Data object

Remarks:

Because VPE compares - and maybe updates - the current determined minimum value with each value added by TVPEChartData.AddValue(), you should call this method **after** you have finished adding values.

25.15 Maximum

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

VPE determines the maximum value of the numeric data stored in a Chart Data object itself. You can change the maximum value with this method.

property double TVPEChartData.Maximum

write; runtime only

Possible Values:

the maximum value of the numeric data of the Chart Data object

Remarks:

Because VPE compares - and maybe updates - the current determined minimum 726 value with each value added by TVPEChartData.AddValue(), you should call this method **after** you have finished adding values.

25.16 AddGap

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

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.

```
method void TVPEChartData.AddGap(
        int Column
)
```

int Column
the column where to add the gap

25.17 AddRow

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

Resizes the internal data table of the ChartData object, 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).

```
method void TVPEChartData.AddRow(
)
```

25.18 AddColumn

[ActiveX, .NET, Java, PHP, etc., method of the TVPEChartData object]

Resizes the internal data table of the ChartData object, 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).

```
method void TVPEChartData.AddColumn(
)
```

25.19 ChartDataAddValue

[VCL only]

Adds a new value to the specified column of the ChartData object specified in the parameter hData.

method void VPE.ChartDataAddValue(
  pointer *hData*,
  long *Column*,
  double *Value*
)

pointer hData
  handle to ChartData object 714

long Column
  column, where to add the value

double Value
  value to add

Example:

```
hData: Pointer
hData = Doc.ChartDataCreate(2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

```
Doc.ChartDataAddValue(hData, 0, 10)
Doc.ChartDataAddValue(hData, 0, 20)
Doc.ChartDataAddValue(hData, 0, 30)
Doc.ChartDataAddValue(hData, 0, 40)

Doc.ChartDataAddValue(hData, 1, 5)
Doc.ChartDataAddValue(hData, 1, 10)
Doc.ChartDataAddValue(hData, 1, 15)
Doc.ChartDataAddValue(hData, 1, 20)
```

Now the internal data table of the Chart Data 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

25.20 ChartDataAddLegend

[VCL only]

Adds a new descriptive string for a column to the legend.

method void VPE.ChartDataAddLegend(
    pointer *hData*,
    string *Legend*
)

pointer hData
    handle to ChartData object 714

string Legend
    legend string

Example:

```
hData: Pointer
hData = Doc.ChartDataCreate(2, 4)
Doc.ChartDataAddLegend(hData, "Apples")
Doc.ChartDataAddLegend(hData, "Bananas")
```

Creates a ChartData object with two columns, titled "Apples" and "Bananas" in the legend.

25.21 ChartDataSetXAxisTitle

[VCL only]

Sets the x-axis title of the chart. This is the descriptive text of the x-axis.

```
method void VPE.ChartDataSetXAxisTitle(
        pointer hData,
        string XAxisTitle
)
```

pointer hData
handle to ChartData object 714

string XAxisTitle
title for x-axis

25.22 ChartDataSetYAxisTitle

[VCL only]

Sets the y-axis title of the chart. This is the descriptive text of the y-axis.

```
method void VPE.ChartDataSetYAxisTitle(
        pointer hData,
        string YAxisTitle
)
```

pointer hData
handle to ChartData object 714

string YAxisTitle
title for y-axis

25.23 ChartDataAddXLabel

[VCL only]

Adds a new X-Label to the chart. By default, the x-axis is automatically labeled. With ChartXLabelState 778 you can switch to user defined labels and then add X-Labels to the chart. (see also ChartXLabelStartValue 777)

method void VPE.ChartDataAddXLabel(
    pointer *hData*,
    string *XLabel*
)

pointer hData
handle to ChartData object 714

string XLabel
string of label to add

Remarks:
You can control if, how and on what position y-labels are drawn with the following properties:

- ChartXLabelState 771
- ChartXGridStep 766
- ChartXLabelStep 775

Example:

```
hData: Pointer
hData = Doc.ChartDataCreate(2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

```
Doc.ChartDataAddValue(hData, 0, 10)
Doc.ChartDataAddValue(hData, 0, 20)
Doc.ChartDataAddValue(hData, 0, 30)
Doc.ChartDataAddValue(hData, 0, 40)

Doc.ChartDataAddValue(hData, 1, 5)
Doc.ChartDataAddValue(hData, 1, 10)
Doc.ChartDataAddValue(hData, 1, 15)
Doc.ChartDataAddValue(hData, 1, 20)

Doc.ChartDataAddLegend(hData, "Apples")
Doc.ChartDataAddLegend(hData, "Bananas")
```

Now the internal data table of the Chart Data 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

```
Doc.ChartXLabelState = VCHART_LABEL_USER
Doc.ChartDataAddXLabel(hData, "1. Quarter")
Doc.ChartDataAddXLabel(hData, "2. Quarter")
Doc.ChartDataAddXLabel(hData, "3. Quarter")
Doc.ChartDataAddXLabel(hData, "4. Quarter")
```

the internal data table of the Chart Data 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

25.24 ChartDataAddYLabel

[VCL only]

Adds a new Y-Label to the chart. By default, the y-axis is labeled automatically. With ChartYLabelState 778 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.

```
method void VPE.ChartDataAddYLabel(
        pointer hData,
        string YLabel
)
```

pointer hData
handle to ChartData object 714

string YLabel
string of label to add

Remarks:
You can control if, how and on what position y-labels are drawn with the following properties:

- ChartYLabelState 778
- ChartYGridStep 767
- ChartYLabelStep 781
- ChartYLabelDivisor 782
- ChartDataSetMinimum 742
- ChartDataSetMaximum 743

25.25 ChartDataSetColor

[VCL only]

By default, VPE assigns itself default colors to each data column. With this method you can specify individual colors for each column.

```
method void VPE.ChartDataSetColor(
        pointer hData,
        long Column,
        Color Color
)
```

pointer hData
  handle to ChartData object 714

long Column
  column the color shall be assigned to

Color Color
  any of the "COLOR_xyz" constants described in Programmer's Manual
  or any RGB value

25.26 ChartDataSetLineStyle

[VCL only]

With this method you can specify individual line styles for each column.

method void VPE.ChartDataSetLineStyle(
  pointer *hData*,
  long *Column*,
  PenStyle [integer] *LineStyle*
)

pointer hData
  handle to ChartData object 714

long Column
  column the line style shall be assigned to

PenStyle [integer] LineStyle
  possible values are:

ActiveX / VCL	Value	Comment
psSolid	0	
psDash	1	-------
psDot	2	....
psDashDot	3	_._._._
psDashDotDot	4	_.._.._

Default:
  psSolid

Example:

```
Doc.ChartDataSetLineStyle(hData, 0, psSolid)
```

25.27 ChartDataSetHatchStyle

[VCL only]

If you create a Bar-, Area- or Pie Chart from the Chart Data, you can specify individual hatch styles for each column with this method. By default, no hatching is used.

method void VPE.ChartDataSetHatchStyle(
pointer *hData*,
long *Column*,
integer *HatchStyle*
)

pointer hData
handle to ChartData object 714

long Column
column the line style shall be assigned to

integer HatchStyle
column the hatch style shall be assigned to; possible values are:

ActiveX / VCL	Value	Comment
hsNone	-1	see HatchStyle 463 – 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:
hsNone (no hatching is used)

Example:

```
Doc.ChartDataSetHatchStyle(hData, 0, hsDiagCross)
```

25.28 ChartDataSetPointType

[VCL only]

If you create a Point Chart from the Chart Data, you can specify individual point styles for each column with this method.

method void VPE.ChartDataSetPointType(
pointer *hData*,
long *Column*,
ChartSymbol [long] *PointType*
)

pointer hData
handle to ChartData object 714

long Column
column the color shall be assigned to

ChartSymbol [long] PointType
possible values are:

ActiveX / VCL	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:

```
Doc.ChartDataSetPointType(hData, 0, VCHART_SYMBOL_SQUARE)
```

25.29 ChartDataSetMinimum

[VCL only]

VPE determines the minimum value of the numeric data stored in a Chart Data object itself. You can change the minimum value with this method.

```
method void VPE.ChartDataSetMinimum(
        pointer hData,
        double Minimum
)
```

pointer hData
handle to ChartData object 714

double Minimum
the minimum value of the numeric data in a Chart Data object

Remarks:
Because VPE compares - and maybe updates - the current determined minimum value with each value added by ChartDataAddValue() 731, you should call this method **after** you have finished adding values.

25.30 ChartDataSetMaximum

[VCL only]

VPE determines the maximum value of the numeric data stored in a Chart Data object itself. You can change the maximum value with this method.

method void VPE.ChartDataSetMaximum(
    pointer *hData*,
    double *Maximum*
)

pointer hData
  handle to ChartData object 714

double Maximum
  the maximum value of the numeric data in a Chart Data object

Remarks:
  Because VPE compares - and maybe updates - the current determined minimum value with each value added by ChartDataAddValue() 731, you should call this method **after** you have finished adding values.

25.31 ChartDataAddGap

[VCL only]

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.

```
method void VPE.ChartDataAddGap(
        pointer hData,
        long Column
)
```

pointer hData
handle to ChartData object 714

long Column
the column where to add the gap

25.32 ChartDataAddRow

[VCL only]

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).

method void VPE.ChartDataAddRow(
    pointer *hData*
)

pointer hData
    handle to ChartData object 714

25.33 ChartDataAddColumn

[VCL only]

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).

```
method void VPE.ChartDataAddColumn(
        pointer hData
)
```

pointer hData
 handle to ChartData object 714

25.34 ChartTitle

(Chart Property) Sets the title property for the next created chart object.

property string VPE.ChartTitle

write; runtime only

Possible Values:

title of the chart

25.35 ChartTitleFontName

(Chart Property) Sets the font for the title of the next created chart object.

property string VPE.ChartTitleFontName

write; runtime only

Possible Values:
name of the font

Default:
Times New Roman (Times on non-Windows platforms)

25.36 ChartTitleFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart title.

property double VPE.ChartTitleFontSizeFactor

write; runtime only

Possible Values:
0.1 <= factor <= 10

Default:
1.0

25.37 ChartSubTitle

(Chart Property) Sets the subtitle property for the next created chart object.

property string VPE.ChartSubTitle

write; runtime only

Possible Values:
subtitle of the chart

25.38 ChartSubTitleFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart's subtitle.

property double VPE.ChartSubTitleFontSizeFactor

write; runtime only

Possible Values:
0.1 <= factor <= 10

Default:
0.6

25.39 ChartFootNote

(Chart Property) Sets the footnote property for the next created chart object.

property string VPE.ChartFootNote

write; runtime only

Possible Values:

footnote of the chart

25.40 ChartFootNoteFontName

(Chart Property) Sets the font of the footnote for the next created chart object.

property string VPE.ChartFootNoteFontName

write; runtime only

Possible Values:
name of the font

Default:
Times New Roman (Times on non-Windows platforms)

25.41 ChartFootNoteFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart's footnote.

property double VPE.ChartFootNoteFontSizeFactor

write; runtime only

Possible Values:
0.1 <= factor <= 10

Default:
0.5

25.42 ChartAxesFontName

(Chart Property) Specifies the font name of the chart's x- and y-axis titles.

property string VPE.ChartAxesFontName

write; runtime only

Possible Values:
name of the font

Default:
Times New Roman (Times on non-Windows platforms)

25.43 ChartAxisTitleFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart's x- and y-axis titles.

property double VPE.ChartAxisTitleFontSizeFactor

write; runtime only

Possible Values:
0.1 <= factor <= 3

Default:
1.0

25.44 ChartLegendFontName

(Chart Property) Sets the font of the legend for the next created chart object.

property string VPE.ChartLegendFontName

write; runtime only

Possible Values:
name of the font

Default:
Times New Roman (Times on non-Windows platforms)

25.45 ChartLegendFontSizeFactor

(Chart Property) Specifies a factor for the font size of the chart's legend.

property double VPE.ChartLegendFontSizeFactor

write; runtime only

Possible Values:
possible values 0.1 <= factor <= 3

Default:
1.4

25.46 ChartLineWidthFactor

(Chart Property) Line Charts only: Specifies a factor for the line thickness.

property double VPE.ChartLineWidthFactor

write; runtime only

Possible Values:
possible values 0.1 <= factor <= 8

Default:
3.0

25.47 ChartBarWidthFactor

(Chart Property) Bar Charts only (2D and 3D): Specifies a factor for bar width.

property double VPE.ChartBarWidthFactor

write; runtime only

Possible Values:
possible values 0.2 <= factor <= 1.9

Default:
1.0

25.48 ChartRow

(Chart Property) For Pie Chart only. Selects the row of the Chart Data object which shall be used by a pie chart. A Pie Chart can only visualize one row of data.

property long VPE.ChartRow

write; runtime only

Possible Values:

the row that shall be visualized by a Pie Chart

Default:

0 (= the pie chart is created out of the first row of the Chart Data object)

Example:

ActiveX / VCL:

```
Doc.ChartRow = 0
Doc.Chart(1, 1, -18, -18, hData, VCHART_3D_PIE)
```

.NET:

```
Doc.ChartRow = 0
Doc.Chart(1, 1, -18, -18, Data, ChartType.Pie3D)
```

Creates a 3D Pie Chart from the 1st row of a Chart Data object.

ActiveX / VCL:

```
Doc.ChartRow = 1
Doc.Chart(1, 1, -18, -18, hData, VCHART_PIE)
```

.NET:

```
Doc.ChartRow = 1
Doc.Chart(1, 1, -18, -18, Data, ChartType.Pie)
```

Creates a 2D Pie Chart from the 2nd row of a Chart Data object.

25.49 ChartGridBkgColor

(Chart Property) Sets the chart grid background color for the next created chart object.

property Color VPE.ChartGridBkgColor

write; runtime only

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_BLACK

25.50 ChartGridBkgMode

(Chart Property) Sets the **chart grid** background mode for the next created chart object.

property BkgMode [integer] VPE.ChartGridBkgMode

write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VBKG_SOLID	0	Solid	solid background color
VBKG_TRANSPARENT	1	Transparent	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. Using other values from the BkgMode 451 enumeration has no effect., because the backgound mode for the whole chart object is set with BkgMode.

25.51 ChartGridType

(Chart Property) Sets the grid type for the next created chart object.

property ChartGridType [integer] VPE.ChartGridType

write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VCHART_GRID_NONE	-1	None	No Grid is drawn
VCHART_GRID_BOTH_AXIS	0	BothAxis	Grid is drawn for both axis
VCHART_GRID_X_AXIS	1	XAxis	Grid is only drawn for the x-axis
VCHART_GRID_Y_AXIS	2	YAxis	Grid is only drawn for the y-axis

Default:
VCHART_GRID_BOTH_AXIS

25.52 ChartGridColor

(Chart Property) Sets the grid color for the next created chart object.

property Color VPE.ChartGridColor

write; runtime only

Possible Values:

any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

25.53 ChartXGridStep

(Chart Property) Sets the x-grid-step for the next created chart object.

The x-grid-step determines at what position on the x-axis an x-grid-line is drawn. In addition it controls at what position a marker is drawn at the x-axis.

property int VPE.ChartXGridStep

write; runtime only

Possible Values:
the grid stepping

Default:
1

Example:

```
Doc.ChartXGridStep = 2
```

means, an x-grid line and a marker are drawn on every second position

25.54 ChartYGridStep

(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.

property double VPE.ChartYGridStep

write; runtime only

Possible Values:

the grid stepping

Default:

Auto 768

Example:

```
Doc.ChartYGridStep = 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, ...)

```
Doc.ChartYGridStep = 10
```

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, ...)

25.55 SetChartYAutoGridStep

(Chart Property) Instructs VPE to determine itself the grid stepping 767 to gain readable results depending on the scale of the chart.

```
method void VPE.SetChartYAutoGridStep(
)
```

Remarks:

This is the default behavior of VPE. You can turn the auto stepping off by setting ChartYGridStep 767 to any value.

25.56 ChartLegendPosition

(Chart Property) Sets the legend position for the next created chart object.

property ChartLegendPosition [long] VPE.ChartLegendPosition

write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VCHART_LEGENDPOS_NONE	-1	None	No legend drawn
VCHART_LEGENDPOS_RIGHT	0	Right	
VCHART_LEGENDPOS_RIGHT_TOP	1	RightTop	
VCHART_LEGENDPOS_RIGHT_BOTTOM	2	RightBottom	
VCHART_LEGENDPOS_LEFT	3	Left	
VCHART_LEGENDPOS_LEFT_TOP	4	LeftTop	
VCHART_LEGENDPOS_LEFT_BOTTOM	5	LeftBottom	
VCHART_LEGENDPOS_TOP	6	Top	
VCHART_LEGENDPOS_BOTTOM	7	Bottom	

Default:
VCHART_LEGENDPOS_LEFT_TOP

25.57 ChartLegendBorderStat

(Chart Property) Determines, whether the legend is drawn with a frame and a shadow or not for the next created chart object.

property boolean [long] VPE.ChartLegendBorderStat

write; runtime only

Possible Values:

Value	Description
True	drawn
False	invisible

Default:

True

25.58 ChartXLabelState

(Chart Property) Determines, how the x-axis labels are drawn for the next created chart object.

property ChartLabelState [integer] VPE.ChartXLabelState

write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VCHART_LABEL_NONE	-1	None	No labels are drawn
VCHART_LABEL_USER	0	User	User defined labels are drawn
VCHART_LABEL_AUTO	1	Auto	Labeling is done automatically

Default:
VCHART_LABEL_AUTO (= labeling is done automatically)

25.59 ChartPieLegendWithPercent

(Chart Property) Pie Chart only: Determines, whether the percent values of each pie element are shown in the legend.

property boolean VPE.ChartPieLegendWithPercent

write; runtime only

Possible Values:

Value	Description
True	yes, the percent values are shown
False	no, they are not shown

Default:

True: yes, the percent values are shown

25.60 ChartPieLabelType

(Chart Property) Pie Chart only: Determines, how the labels of a pie chart are drawn.

property ChartPieLabelType [integer] VPE.ChartPieLabelType

write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VCHART_PIE_LABEL_NONE	0	None	no labels are drawn
VCHART_PIE_LABEL_PERCENTAGE	1	Percentage	the percentage values are drawn
VCHART_PIE_LABEL_LEGEND	2	Legend	the legend texts are used for the labels
VCHART_PIE_LABEL_XLABELS	3	Xlabels	the user-defined x-labels are used

Default:
PIE_LABEL_PERCENTAGE

25.61 ChartXLabelFontSizeFactor

(Chart Property) Specifies a factor for the font size of the x-axis labels.

property double VPE.ChartXLabelFontSizeFactor

write; runtime only

Possible Values:
0.1 <= factor <= 3

Default:
1.5

25.62 ChartXLabelStep

(Chart Property) Sets the x-label-step for the next created chart object. The x-label-step determines the n-th x-grid-line where a label is drawn.

property long VPE.ChartXLabelStep

write; runtime only

Possible Values:
the n-th x-grid line where a label is drawn

Default:
1 (= each x-grid line a label is drawn)

Remarks:
This property is always in effect, regardless if ChartXLabelState 771 is VCHART_LABEL_USER or VCHART_LABEL_AUTO.

Example:

```
Doc.ChartXLabelStep = 1
```

means, at each x-grid line a label is drawn at the x-axis.

```
Doc.ChartXLabelStep = 2
```

means, that a label is only drawn each 2nd x-grid line.

25.63 ChartXLabelAngle

(Chart Property) Sets the drawing angle (clockwise) for the x-axis labels.

property integer VPE.ChartXLabelAngle

write; runtime only

Possible Values:

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

25.64 ChartXLabelStartValue

(Chart Property) Sets the start value for the x-labels. If the x-labeling is done automatically by VPE (i.e. ChartXLabelState 771 = VCHART_LABEL_AUTO), this is the start value where VPE will start to number each row.

property long VPE.ChartXLabelStartValue

write; runtime only

Possible Values:
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:

```
Doc.ChartXLabelStartValue = 7
```

VPE will label the x-axis with "7, 8, 9, 10, ..."

25.65 ChartYLabelState

(Chart Property) Determines, how the y-axis labels are drawn for the next created chart object.

property ChartLabelState [integer] VPE.ChartYLabelState

write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VCHART_LABEL_NONE	-1	None	No labels are drawn
VCHART_LABEL_USER	0	User	User defined labels are drawn
VCHART_LABEL_AUTO	1	Auto	Labeling is done automatically

Default:

VCHART_LABEL_AUTO

25.66 ChartYLabelFontSizeFactor

(Chart Property) Specifies a factor for the font size of the y-axis labels.

property double VPE.ChartYLabelFontSizeFactor

write; runtime only

Possible Values:
0.1 <= factor <= 3

Default:
1.5

25.67 ChartLabelsFontName

(Chart Property) Specifies the font name of the chart's x- and y-axis labels.

property string VPE.ChartLabelsFontName

write; runtime only

Possible Values:
name of the font

Default:
Arial (Helvetica on non-Windows platforms)

25.68 ChartYLabelStep

(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.

property long VPE.ChartYLabelStep

write; runtime only

Possible Values:
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 778 is VCHART_LABEL_USER or VCHART_LABEL_AUTO.

Example:

```
Doc.ChartYLabelStep = 1
```

means, at each y-grid line a label is drawn at the y-axis.

```
Doc.ChartYLabelStep = 2
```

means, that a label is only drawn each 2nd y-grid line.

25.69 ChartYLabelDivisor

(Chart Property) Sets the divisor for y-labels, if the y-labels are drawn automatically by VPE (ChartYLabelState 778 = VCHART_LABEL_AUTO).

property double VPE.ChartYLabelDivisor

write; runtime only

Possible Values:
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"

25.70 ChartGridRotation

(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.

property integer VPE.ChartGridRotation

write; runtime only

Possible Values:

Value	Description
0	0 degrees
900	90 degrees, the chart is rotated by 90 degrees clockwise

Default:
0

25.71 ChartYAxisAngle

(Chart Property) Only 3D charts. Sets the view angle of the y-axis. Values are in 1/10°.

property integer VPE.ChartYAxisAngle

write; runtime only

Possible Values:
150 <= yangle <= 780

Default:
300 (= 30 degrees)

Remarks:
With this property you can change the view perspective, it does not rotate the chart itself.

25.72 ChartXAxisAngle

(Chart Property) Only 3D charts. Sets the view angle of the x-axis. Values are in 1/10°.

property integer VPE.ChartXAxisAngle

write; runtime only

Possible Values:
150 <= xangle <= 780

Default:
450 (= 45 degrees)

Remarks:
With this property you can change the view perspective, it does not rotate the chart itself.

25.73 Chart

Creates a chart object at the given position from the given ChartData object. The type of the chart is specified in the parameter ChartType. All current chart property settings apply to the created chart.

The property TextColor 492 - which is the generic foreground color - specifies the color for the following chart elements:

- title, subtitle, footnote
- x- / y-axis titles
- x- / y-labels

method void VPE.Chart(
VpeCoord *Left*,
VpeCoord *Top*,
VpeCoord *Right*,
VpeCoord *Bottom*,
TVPEChartData [pointer] *hData*,
ChartType [integer] *ChartType*
)

VpeCoord *Left, Top, Right, Bottom*
position and dimensions of the chart

TVPEChartData [pointer] hData

VCL	handle to ChartData object
all other Control-Types	Chart Data object

ChartType [integer] ChartType
possible values are:

ActiveX / VCL	Value	Enum
VCHART_POINT	0	Point
VCHART_LINE	1	Line
VCHART_BAR	2	Bar
VCHART_STACKED_BAR_ABSOLUTE	3	StackedBarAbsolute
VCHART_STACKED_BAR_PERCENT	4	StackedBarPercent
VCHART_3D_BAR	5	Bar3D
VCHART_3D_STACKED_BAR_ABSOLUTE	6	StackedBar3DAbsolute
VCHART_3D_STACKED_BAR_PERCENT	7	StackedBar3DPercent
VCHART_PIE	8	Pie
VCHART_3D_PIE	9	Pie3D

VCHART_AREA_ABSOLUTE	10	AreaAbsolute
VCHART_AREA_PERCENT	11	AreaPercent

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 520 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 ActiveX, .NET, Java, ...:

```
dim Data as TVPEChartData
Data = Doc.ChartDataCreate(2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

```
Data.AddValue(0, 10)
Data.AddValue(0, 20)
Data.AddValue(0, 30)
Data.AddValue(0, 40)

Data.AddValue(1, 5)
Data.AddValue(1, 10)
Data.AddValue(1, 15)
Data.AddValue(1, 20)

Data.AddLegend("Apples")
Data.AddLegend("Bananas")
```

Now the internal data table of the Chart Data 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

```
Doc.ChartXLabelState = ChartLabelState.User
Data.AddXLabel("1. Quarter")
Data.AddXLabel("2. Quarter")
Data.AddXLabel("3. Quarter")
Data.AddXLabel("4. Quarter")
```

the internal data table of the Chart Data 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:

```
Doc.BkgMode = BkgMode.GradientLine
Doc.ChartTitle = "Bar Chart"
Doc.Chart(1, 1, -18, -18, Data, ChartType.Bar)
Doc.PageBreak()

Doc.ChartTitle = "3-D Bar Chart"
Doc.Chart(1, 1, -18, -18, Data, ChartType.Bar3D)
Doc.PageBreak()

Doc.ChartTitle = "3-D Pie Chart"
Doc.ChartSubTitle = "Row 1"
Doc.ChartRow = 0
Doc.Chart(1, 1, -18, -18, Data, ChartType.Pie3D)
Doc.PageBreak()

Doc.ChartTitle "3-D Pie Chart"
Doc.ChartSubTitle = "Row 2"
Doc.ChartRow = 1
Doc.Chart(1, 1, -18, -18, Data, ChartType.Pie3D)
```

Example VCL:

```
hData: Pointer
hData = Doc.ChartDataCreate(2, 4)
```

Creates a ChartData object with two columns (for example Apples and Bananas), where each column can hold 4 numeric data values (= rows).

```
Doc.ChartDataAddValue(hData, 0, 10)
Doc.ChartDataAddValue(hData, 0, 20)
Doc.ChartDataAddValue(hData, 0, 30)
Doc.ChartDataAddValue(hData, 0, 40)

Doc.ChartDataAddValue(hData, 1, 5)
Doc.ChartDataAddValue(hData, 1, 10)
Doc.ChartDataAddValue(hData, 1, 15)
Doc.ChartDataAddValue(hData, 1, 20)

Doc.ChartDataAddLegend(hData, "Apples")
Doc.ChartDataAddLegend(hData, "Bananas")
```

Now the internal data table of the Chart Data 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

```
Doc.ChartXLabelState = VCHART_LABEL_USER
Doc.ChartDataAddXLabel(hData, "1. Quarter")
Doc.ChartDataAddXLabel(hData, "2. Quarter")
Doc.ChartDataAddXLabel(hData, "3. Quarter")
Doc.ChartDataAddXLabel(hData, "4. Quarter")
```

the internal data table of the Chart Data 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:

```
Doc.BkgMode = VBKG_GRD_LINE
Doc.ChartTitle = "Bar Chart"
Doc.Chart(1, 1, -18, -18, hData, VCHART_BAR)
Doc.PageBreak()

Doc.ChartTitle = "3-D Bar Chart"
Doc.Chart(1, 1, -18, -18, hData, VCHART_3D_BAR)
Doc.PageBreak()

Doc.ChartTitle = "3-D Pie Chart"
Doc.ChartSubTitle = "Row 1"
Doc.ChartRow = 0
Doc.Chart(1, 1, -18, -18, hData, VCHART_3D_PIE)
Doc.PageBreak()

Doc.ChartTitle "3-D Pie Chart"
Doc.ChartSubTitle = "Row 2"
Doc.ChartRow = 1
Doc.Chart(1, 1, -18, -18, hData, VCHART_3D_PIE)
```

This page is intentionally left blank.

FormFields

26 FormFields

[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 "FormFields" in the Programmer's Manual

26.1 FormField

Creates a Form Field at the given position, using the number of character cells specified by CharCount 795. Accordingly, each character cell has the

width = ((x2 - x) / CharacterCount 795*) - max(*DividerPenSize 796, AltDividerPenSize 799*)*

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 FormFieldFlags 805 - 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.

method VpeCoord VPE.FormField(
VpeCoord *Left*,
VpeCoord *Top*,
VpeCoord *Right*,
VpeCoord *Bottom*,
string *Text*
)

VpeCoord *Left, Top, Right, Bottom*
position and dimensions of the FormField

string Text
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 Write(Box) was used.
- The bottom line will only be drawn, if the normal PenSize 443 (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 WriteBox() 496.
- 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 520 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:

RenderFormField 438
"FormFields" in the Programmer's Manual

26.2 CharCount

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.

property long VPE.CharCount

read / write; runtime only; also supported by TVPEObject

Possible Values:
the number of character cells drawn in a FormField

Default:
15

Remarks:
If CharCount is set to zero, the FormField will behave the same as if Write(Box) was used.

See also:
"FormFields" in the Programmer's Manual

26.3 DividerPenSize

Sets the pen size for the dividers of a FormField.

property VpeCoord **VPE.DividerPenSize**

read / write; runtime only; also supported by TVPEObject

Possible Values:
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 FormFieldFlags 805), the value for the pen size influences the computed font size.

See also:
"FormFields" in the Programmer's Manual

26.4 DividerPenColor

Sets the pen color for the dividers of a FormField.

property Color VPE.DividerPenColor

read / write; runtime only; also supported by TVPEObject

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_BLACK

See also:
"FormFields" in the Programmer's Manual

26.5 AltDividerNPosition

Sets the position where Alternative Dividers are drawn instead of normal Dividers.

property long VPE.AltDividerNPosition

read / write; runtime only; also supported by TVPEObject

Possible Values:
the position of Alternative Dividers

Default:
0 (no Alternative Dividers are drawn)

See also:
"FormFields" in the Programmer's Manual

26.6 AltDividerPenSize

Sets the pen size for the Alternative Dividers of a FormField.

property VpeCoord **VPE.AltDividerPenSize**

read / write; runtime only; also supported by TVPEObject

Possible Values:
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 FormFieldFlags 805), the value for the pen size influences the computed font size.

See also:
"FormFields" in the Programmer's Manual

26.7 AltDividerPenColor

Sets the pen color for the Alternative Dividers of a FormField.

property Color VPE.AltDividerPenColor

read / write; runtime only; also supported by TVPEObject

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_BLACK

See also:
"FormFields" in the Programmer's Manual

26.8 BottomLinePenSize

Sets the pen size for the bottom line. A bottom line is only drawn, if the global PenSize 443 (used for lines, frames, etc.) is set to zero. Otherwise a framed FormField will be drawn similar to WriteBox() 496.

property VpeCoord **VPE.BottomLinePenSize**

read / write; runtime only; also supported by TVPEObject

Possible Values:
the pen size for the bottom line

Default:
0.03 (= 0.3 mm)

See also:
"FormFields" in the Programmer's Manual

26.9 BottomLinePenColor

Sets the pen color for the bottom line. A bottom line is only drawn, if the global PenSize 443 is set to zero. Otherwise a framed FormField will be drawn similar to WriteBox() 496.

property Color VPE.BottomLinePenColor

read / write; runtime only; also supported by TVPEObject

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_BLACK

See also:
"FormFields" in the Programmer's Manual

26.10 DividerStyle

Sets the paint-style for dividers.

property DividerStyle [long] VPE.DividerStyle

read / write; runtime only; also supported by TVPEObject

Possible Values:

the style of dividers, possible values are:

ActiveX / VCL	Value	Enum	Comment
VFF_STYLE_NONE	0	StyleNone	no dividers
VFF_STYLE_1_4	1	Style_1_4	1/4 height
VFF_STYLE_1_3	2	Style_1_3	1/3 height
VFF_STYLE_1_2	3	Style_1_2	1/2 height (default)
VFF_STYLE_2_3	4	Style_2_3	2/3 height
VFF_STYLE_3_4	5	Style_3_4	3/4 height
VFF_STYLE_1_1	6	Style_1_1	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

26.11 AltDividerStyle

Sets the paint-style for Alternative Dividers.

property DividerStyle [long] VPE.AltDividerStyle

read / write; runtime only; also supported by TVPEObject

Possible Values:

the style of Alternative Dividers, possible values are:

ActiveX / VCL	Value	Enum	Comment
VFF_STYLE_NONE	0	StyleNone	no dividers
VFF_STYLE_1_4	1	Style_1_4	1/4 height
VFF_STYLE_1_3	2	Style_1_3	1/3 height
VFF_STYLE_1_2	3	Style_1_2	1/2 height (default)
VFF_STYLE_2_3	4	Style_2_3	2/3 height
VFF_STYLE_3_4	5	Style_3_4	3/4 height
VFF_STYLE_1_1	6	Style_1_1	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

26.12 FormFieldFlags

Sets additional control flags for FormFields.

property FormFieldFlags [long] VPE.FormFieldFlags

read / write; runtime only; also supported by TVPEObject

Possible Values:

possible values are any combination of:

ActiveX / VCL	Value	Enum	Comment
VFF_FLAG_DIV_FIRST	1	DivFirst	first divider is painted (only, if no frame)
VFF_FLAG_DIV_LAST	2	DivLast	last divider is painted (only, if no frame)
VFF_FLAG_ALT_FIRST	4	AltFirst	(default) first divider is painted as AltDivider (overrides _DIV_FIRST) (only, if no frame)
VFF_FLAG_ALT_LAST	8	AltLast	(default) last divider is painted as AltDivider (overrides _DIV_LAST) (only, if no frame)
VFF_FLAG_AUTO_FONTSIZE	16	AutoFontSize	(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 FormField 793 for details.

See also:

"FormFields" in the Programmer's Manual

This page is intentionally left blank.

Template Functions

27 Template Functions

[Enterprise Edition and above]

This section describes the methods and properties related to Templates.

See also:

"dycodoc Template Processing" in the Programmer's Manual

27.1 LoadTemplate

Loads a template file and creates a Template Object 816 from it in memory.

The method returns a Template Object, which is the anchor object for 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 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.

```
method TVPETemplate VPE.LoadTemplate(
        string FileName
)
```

string FileName
  the path and file name of the file to load

Returns:
  The Template Object, if the template file was found and could be loaded.
  In case of an error an exception is thrown and LastError 201 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() 810 for loading templates with an *Authenticity Key*.

See also:
  "dycodoc Template Processing" in the Programmer's Manual

27.2 LoadTemplateAuthKey

This method is identical to LoadTemplate() 809, 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 816 from it in memory.

The method returns a Template Object, which is the anchor object for 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 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.

```
method TVPETemplate VPE.LoadTemplateAuthKey(
        string FileName,
        long Key1,
        long Key2,
        long Key3,
        long Key4
)
```

string FileName
the path and file name of the file to load

long Key1, Key2, Key3, Key4
the Authenticity Key

Returns:
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.

In case of an error an exception is thrown and LastError 201 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

27.3 DumpTemplate

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.

```
method integer VPE.DumpTemplate(
        TVPETemplate objTemplate
)
```

TVPETemplate objTemplate
the Template Object 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 364 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 201 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

27.4 DumpTemplatePage

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.

```
method integer VPE.DumpTemplatePage(
        TVPETemplate objTemplate,
        long page
)
```

TVPETemplate objTemplate
the Template Object that shall be dumped

long page
The page of the template that shall be dumped, may be in the range of
0 (= first page) to PageCount 846 - 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() 811-methods do not make any differences between the AutoBreakMode 364 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 201 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

27.5 UseTemplateMargins

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.

```
method void VPE.UseTemplateMargins(
        TVPETemplate objTemplate,
        long page
)
```

long page

The page of the template whose margins shall be retrieved, may be in the range of 0 (= first page) to PageCount 846 - 1 (= last page).

See also:

"dycodoc Template Processing" in the Programmer's Manual

27.6 UseTemplateSettings

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.

```
method void VPE.UseTemplateSettings(
        TVPETemplate objTemplate,
        long page
)
```

long page

The page of the template whose page settings shall be retrieved, may be in the range of 0 (= first page) to PageCount 846 - 1 (= last page).

See also:

"dycodoc Template Processing" in the Programmer's Manual

Template Object

28 Template Object

[Enterprise Edition and above]

This section describes the methods and properties of the Template Object.

The class-name is TVPETemplate.

.NET: The TVPETemplate class is derived from the base class *Object*.

See also:

"dycodoc Template Processing" in the Programmer's Manual

28.1 ObjectHandle

This is the internal object handle of the VPE DLL for the Template Object.

Do not use.

property long TVPETemplate.ObjectHandle

read; runtime only

Possible Values:

Value	Description
NULL	not initialized
NON-NULL	it holds a reference to an internal Template Object in the VPE DLL

See also:

"dycodoc Template Processing" in the Programmer's Manual

28.2 Master

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.)

property boolean [integer] TVPETemplate.Master

read / write; runtime only

Possible Values:

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() 809)

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 the *Master* property 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

28.3 ClearFields

Clears all Fields of a template, i.e. null-values are assigned to all fields within the given template.

method void TVPETemplate.ClearFields()

Remarks:

A null-value is a special value, like in SQL databases. An empty string or an integer / double / date with the value zero are not null-values.

In string expressions a null-value is treated like an empty string. In numeric or date expressions, a null-value is treated like a zero.

See also:

"dycodoc Template Processing" in the Programmer's Manual

28.4 GetFieldIsNull

Tests whether a Field has a null-value. A field has a null-value after calling ClearFields 819 or after explicitly assigning a null-value to a Field by either calling TVPEField.SetNull 861 or TVPETemplate.SetFieldToNull 821.

method boolean TVPETemplate.GetFieldIsNull(
  string *DataSourcePrefix*,
  string *FieldName*
)

string DataSourcePrefix
  prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
  name of the Data Source Field

Returns:

Value	Description
True	the field has a null-value
False	the field has not a null-value or the field was not found

See also:
  "dycodoc Template Processing" in the Programmer's Manual

28.5 SetFieldToNull

Assigns a null-value to a field. A field has a null-value after calling ClearFields 819 or after explicitly assigning a null-value to a Field by either calling TVPEField.SetNull 861 or by calling this method.

```
method void TVPETemplate.SetFieldToNull(
        string DataSourcePrefix,
        string FieldName
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.6 GetFieldNullValueText

Returns the text which is printed for Fields that have a null-value. You can assign a special text to Fields with a null-value. By default, the null-value text is an empty string. You can specify for each field an individual null-value text, for example "n/a".

method boolean TVPETemplate.GetFieldNullValueText(
string *DataSourcePrefix*,
string *FieldName*
)

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

Returns:
The string with the null-value text of the Field.

Remarks:
In case of an error, an exception is thrown (for example, the field was not found, or out of memory)

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.7 SetFieldNullValueText

Sets the text which is printed for Fields that have a null-value. You can assign a special text to Fields with a null-value. By default, the null-value text is an empty string. You can specify for each field an individual null-value text, for example "n/a".

method boolean TVPETemplate.SetFieldNullValueText(
    string *DataSourcePrefix*,
    string *FieldName*,
    string *Text*
)

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

string Text
the string that shall be assigned to the Field's null-value text

Returns:

Value	Description
True	if the field was found and its null-value text could be set
False	otherwise

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.8 GetFieldAsString

Returns the value of a Field as string.

```
method string TVPETemplate.GetFieldAsString(
        string DataSourcePrefix,
        string FieldName
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

Returns:
The string value of the Field.

Remarks:
In case of an error, an exception is thrown (for example, the field was not found, or out of memory)

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.9 SetFieldAsString

Sets the value of a Field as string.

```
method int TVPETemplate.SetFieldAsString(
        string DataSourcePrefix,
        string FieldName,
        string Value
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

string Value
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

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.10 GetFieldAsInteger

Returns the value of a Field as integer.

```
method integer TVPETemplate.GetFieldAsInteger(
        string DataSourcePrefix,
        string FieldName
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

Returns:
The integer value of the Field.

Remarks:
In case of an error, an exception is thrown (for example, if the field contained pure text that could not be converted to an integer value)

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.11 SetFieldAsInteger

Sets the value of a Field as integer.

```
method integer TVPETemplate.SetFieldAsInteger(
        string DataSourcePrefix,
        string FieldName,
        integer Value
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

integer 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

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.12 GetFieldAsNumber

Returns the value of a Field as number (double).

```
method double TVPETemplate.GetFieldAsNumber(
        string DataSourcePrefix,
        string FieldName
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

Returns:
The double value of the Field.

Remarks:
In case of an error, an exception is thrown (for example, if the field contained pure text that could not be converted to a number value)

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.13 SetFieldAsNumber

Sets the value of a Field as number (double).

```
method integer TVPETemplate.SetFieldAsNumber(
        string DataSourcePrefix,
        string FieldName,
        double Value
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

double Value
the double 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

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.14 GetFieldAsBoolean

Returns the value of a Field as boolean.

```
method boolean [integer] TVPETemplate.GetFieldAsBoolean(
        string DataSourcePrefix,
        string FieldName
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

Returns:
The boolean value of the Field.

Remarks:
In case of an error, an exception is thrown (for example, if the field contained pure text that could not be converted to a boolean value)

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.15 SetFieldAsBoolean

Sets the value of a Field as boolean.

method integer TVPETemplate.SetFieldAsNumber(
string *DataSourcePrefix*,
string *FieldName*,
boolean *Value*
)

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

boolean Value
the boolean 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

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.16 DateTimeIsUTC

Specifies, whether VPE's date/time functions receive and return date/time values in UTC (Universal Time Coordinated) or in local time.

property boolean [integer] VPE. DateTimeIsUTC

read / write; runtime only

Possible Values:

Value	Description
True	yes, parameters and return values of VPE's date/time functions are in UTC
False	no, parameters and return values of VPE's date/time functions are in local time

Default:

False = parameters and return values of VPE's date/time functions are in local time

Example:

```
Doc.DateTimeIsUTC = true
```

See also:

"dycodoc Template Processing" in the Programmer's Manual

28.17 SetFieldAsDateTime

Sets the value of a Field as date / time.

method integer TVPETemplate.SetFieldAsDateTime(
  string *DataSourcePrefix*,
  string *FieldName*,
  integer *year*,
  integer *month*,
  integer *day*,
  integer *hour*,
  integer *minute*,
  integer *second*,
  integer *msec*
)

string DataSourcePrefix
 prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
 name of the Data Source Field

int year, month, day, hour, minute, second, msec
 the integer values, which shall be assigned to the respective values of the Field as integer
 month is from 0 - 11
 msec is in milliseconds

Returns:

Value	Description
True	if the field was found and its value could be set
False	otherwise

See also:
 "dycodoc Template Processing" in the Programmer's Manual

28.18 GetFieldAsOleDateTime

[not in .NET]

Returns the value of a Field as OLE date / time.

```
method date TVPETemplate.GetFieldAsOleDateTime(
        string DataSourcePrefix,
        string FieldName
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

Returns:
The OLE date / time value of the Field.

Remarks:
In case of an error, an exception is thrown (for example, if the field contained pure text that could not be converted to an OLE date / time value)

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.19 SetFieldAsOleDateTime

[not in .NET]

Sets the value of a Field as OLE date / time.

method integer TVPETemplate.SetFieldAsOleDateTime(
string *DataSourcePrefix*,
string *FieldName*,
date *Value*
)

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

date Value
the OLE date / time 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

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.20 GetFieldAsNetDateTime

Returns the value of a Field as .NET date / time.

method DateTime TVPETemplate.GetFieldAsNetDateTime(
    string *DataSourcePrefix*,
    string *FieldName*
)

string DataSourcePrefix
    prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
    name of the Data Source Field

Returns:
    The .NET date / time value of the Field.

Remarks:
    In case of an error, an exception is thrown (for example, if the field contained pure text that could not be converted to a .NET date / time value)

See also:
    "dycodoc Template Processing" in the Programmer's Manual

28.21 SetFieldAsNetDateTime

Sets the value of a Field as .NET date / time.

```
method integer TVPETemplate.SetFieldAsNetDateTime(
        string DataSourcePrefix,
        string FieldName,
        DateTime Value
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

DateTime Value
the .NET date / time 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

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.22 GetFieldAsJavaDateTime

Returns the value of a Field as Java date / time (number of milliseconds since midnight Jan 1, 1970).

```
method double TVPETemplate.GetFieldAsJavaDateTime(
        string DataSourcePrefix,
        string FieldName
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

Returns:
The Java date / time value of the Field.

Remarks:
In case of an error, an exception is thrown (for example, if the field contained pure text that could not be converted to an Java date / time value)

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.23 SetFieldAsJavaDateTime

Sets the value of a Field as Java date / time (number of milliseconds since midnight Jan 1, 1970).

```
method integer TVPETemplate.SetFieldAsJavaDateTime(
        string DataSourcePrefix,
        string FieldName,
        double Value
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

double Value
the Java date / time 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

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.24 GetFieldAsPythonDateTime

[Python only]

Returns the value of a Field as Python date / time.

method datetime TVPETemplate.GetFieldAsPythonDateTime(
    string *DataSourcePrefix*,
    string *FieldName*
)

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

Returns:
The Python date / time value of the Field.

Remarks:
In case of an error, an exception is thrown (for example, if the field contained pure text that could not be converted to an Python date / time value)

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.25 SetFieldAsPythonDateTime

[Python only]

Sets the value of a Field as Python date / time.

```
method integer TVPETemplate.SetFieldAsPythonDateTime(
        string DataSourcePrefix,
        string FieldName,
        datetime Value
)
```

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

datetime Value
the Python date / time 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

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.26 ClearFields

Resets all Fields of the specified template to NULL.

method void TVPETemplate.ClearFields(
)

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.27 DataSourceCount

Returns the total number of Data Sources used within the given template.

property long TVPETemplate.DataSourceCount

read; runtime only

Returns:
the total number of Data Sources used within the given template

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.28 DataSourceObject

Returns a Data Source Object.

```
method TVPEDataSource TVPETemplate.DataSourceObject(
        long DataSourceIndex
)
```

long DataSourceIndex
index into the array of available Data Sources
this value must be in the range between 0 and DataSourceCount 843 - 1

Returns:
the Data Source Object; if *DataSourceIndex* is out of range, an exception is thrown

Remarks:
The function can be used to access Data Source Objects directly.

See also:
"dycodoc Template Processing" in the Programmer's Manual
"Important Note for VPE-VCL Users" in the Programmer's Manual

28.29 FindFieldObject

Searches for a given Field and returns the Field object 856.

method TVPEField TVPETemplate.FindFieldObject(
string *DataSourcePrefix*,
string *FieldName*
)

string DataSourcePrefix
prefix of the Data Source, this is the unique name of a Data Source as defined in *dycodoc*

string FieldName
name of the Data Source Field

Returns:
a Field Object; if the field is not found the ObjectHandle of the returned object is null.

Remarks:
The function can be used to access Field Objects directly.

Example:

```
template.FindFieldObject(hTpl, "SomeTable", "Year").AsInteger = 1982;
```

See also:
"dycodoc Template Processing" in the Programmer's Manual
"Important Note for VPE-VCL Users" in the Programmer's Manual

28.30 PageCount

Returns the number of pages in the given template.

property long TVPETemplate.PageCount

read; runtime only

Returns:
the number of pages in the given template

See also:
"dycodoc Template Processing" in the Programmer's Manual

28.31 PageObject

Returns the Page Object of the specified page.

```
method TVPETemplatePage TVPETemplate.PageObject(
        long PageIndex
)
```

long PageIndex
index into the array of available Pages
this value must be in the range between 0 and PageCount 846 - 1

Returns:
the Page Object; if *PageIndex* is out of range an exception is thrown

Remarks:
The function can be used to access Page Objects directly.

See also:
"dycodoc Template Processing" in the Programmer's Manual
"Important Note for VPE-VCL Users" in the Programmer's Manual

28.32 FindVpeObject

In *dycodoc* a unique name 892 is assigned to each VPE Object 884. This function searches for a given VPE Object name in the template and returns the VPE Object.

method TVPEObject TVPETemplate.FindVpeObject(
    string *ObjectName*
)

string ObjectName
    name of the VPE Object as defined in *dycodoc*

Returns:
    a VPE Object. If the object is not found the ObjectHandle of the returned object is null.

Remarks:
    The function can be used to access VPE Objects within the template directly.

Example:

ActiveX / VCL:
```
Dim VPEObject as TVPEObject
VPEObject = template.FindVpeObject("Text1")
VPEObject.BkgMode = VBKG_SOLID
VPEObject.BkgColor = COLOR_RED
```

.NET:
```
Dim VPEObject as TVPEObject
VPEObject = template.FindVpeObject("Text1")
VPEObject.BkgMode = BkgMode.Solid
VPEObject.BkgColor = 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
    "Important Note for VPE-VCL Users" in the Programmer's Manual

DataSource Object

29 DataSource Object

[Enterprise Edition and above]

This section describes the methods and properties of the Data Source Object. A Data Source encapsulates a table, which is defined in the *dycodoc* field definitions pane.

The class-name is TVPEDataSource.

.NET: The TVPEDataSource class is derived from the base class *Object*.

See also:

"dycodoc Template Processing" in the Programmer's Manual

29.1 ObjectHandle

This is the internal object handle of the VPE DLL for the DataSource Object.

The internal Object Handle is of interest to check if its value is null, which means that it is not initialized, i.e. it has no corresponding object in the DLL. This can happen in case of errors, i.e. if the object is not existing.

property long TVPEDataSource.ObjectHandle

read; runtime only

Possible Values:

Value	Description
NULL	not initialized
NON-NULL	it holds a reference to an internal object in the VPE DLL

See also:

"dycodoc Template Processing" in the Programmer's Manual

29.2 Prefix

Returns the prefix of the Data Source, i.e. the table name as defined in *dycodoc*.

property string TVPEDataSource.Prefix

read; runtime only

Returns:

the prefix of the Data Source as defined in *dycodoc*

See also:

"dycodoc Template Processing" in the Programmer's Manual

29.3 FileName

Returns the file name of the Data Source. This is the same name as the table name.

property string TVPEDataSource.FileName

read; runtime only

Returns:
the file name of the Data Source

See also:
"dycodoc Template Processing" in the Programmer's Manual

29.4 Description

Returns the description of the Data Source as defined in *dycodoc*.

property string TVPEDataSource.Description

read; runtime only

Returns:
the description of the Data Source as defined in *dycodoc*

See also:
"dycodoc Template Processing" in the Programmer's Manual

29.5 FieldCount

Returns the total number of Fields used within the Data Source.

property long TVPEDataSource.FieldCount

read; runtime only

Returns:
the total number of Fields used within the Data Source

See also:
"dycodoc Template Processing" in the Programmer's Manual

29.6 FieldObject

Returns a Field Object.

```
method TVPEField TVPEDataSource.FieldObject(
        long FieldIndex
)
```

long FieldIndex
index into the array of available Fields
this value must be in the range between 0 and FieldCount() 855 - 1

Returns:
a Field Object

Remarks:
The function can be used to access Field Objects within the Data Source directly.

See also:
"dycodoc Template Processing" in the Programmer's Manual
"Important Note for VPE-VCL Users" in the Programmer's Manual

Field Object

30 Field Object

[Enterprise Edition and above]

This section describes the methods and properties of the Field Object. A Field Object encapsulates a single field of a Data Source / table.

The class-name is TVPEField.

.NET: The TVPEField class is derived from the base class *Object*.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.1 ObjectHandle

This is the internal object handle of the VPE DLL for the Field Object.

The internal Object Handle is of interest to check if its value is null, which means that it is not initialized, i.e. it has no corresponding object in the DLL. This can happen in case of errors, i.e. if the object is not existing.

property long TVPEField.ObjectHandle

read; runtime only

Possible Values:

Value	Description
NULL	not initialized
NON-NULL	it holds a reference to an internal object in the VPE DLL

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.2 IsNull

Tests whether a Field has a null-value. A field has a null-value after calling ClearFields 819 or after explicitly assigning a null-value to a Field by either calling TVPEField.SetNull 861 or TVPETemplate.SetFieldToNull 821.

property boolean TVPEField.IsNull

read; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldIsNull 820 and SetFieldToNull 821.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.3 SetNull

Assigns a null-value to a field. A field has a null-value after calling TVPETemplate.ClearFields 819 or after explicitly assigning a null-value to a Field by either calling TVPEField.SetNull 861 or by calling this method.

method void TVPEField.SetNull

Remarks:

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldIsNull 820 and SetFieldToNull 821.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.4 NullValueText

Sets / returns the text which is printed for Fields that have a null-value. You can assign a special text to Fields with a null-value. By default, the null-value text is an empty string. You can specify for each field an individual null-value text, for example "n/a".

property string TVPEField.NullValueText

read / write; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldNullValueText 822 and SetFieldNullValueText 823.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.5 AsString

Sets / returns the value of a Field as string.

property string TVPEField.AsString

read / write; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldAsString 824 and SetFieldAsString 825.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.6 AsInteger

Sets / returns the value of a Field as integer.

property integer TVPEField.AsInteger

read / write; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory or the field contained pure text that could not be converted to an integer value).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldAsInteger 826 and SetFieldAsInteger 827.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.7 AsNumber

Sets / returns the value of a Field as number (double).

property double TVPEField.AsNumber

read / write; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory or the field contained pure text that could not be converted to an number value).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldAsNumber 828 and SetFieldAsNumber 829.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.8 AsBoolean

Sets / returns the value of a Field as boolean.

property Boolean [integer] TVPEField.AsBoolean

read / write; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory or the field contained pure text that could not be converted to an boolean value).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldAsBoolean 830 and SetFieldAsBoolean 831.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.9 SetAsDateTime

Sets / returns the value of a Field as date / time.

method integer TVPEField.SetAsDateTime(
 integer *year*,
 integer *month*,
 integer *day*,
 integer *hour*,
 integer *minute*,
 integer *second*,
 integer *msec*
)

int year, month, day, hour, minute, second, msec
 the integer values, which shall be assigned to the respective values of the Field as integer

Returns:

Value	Description
True	if the field was found and its value could be set
False	otherwise

Remarks:
 The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See SetFieldAsDateTime 833.

See also:
 "dycodoc Template Processing" in the Programmer's Manual

30.10 AsOleDateTime

[not in .NET]

Sets / returns the value of a Field as OLE date / time.

property date TVPEField.AsOleDateTime

read / write; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory or the field contained pure text that could not be converted to an OLE date / time value).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldAsOleDateTime 834 and SetFieldAsOleDateTime 835.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.11 AsNetDateTime

Sets / returns the value of a Field as .NET date / time.

property DateTime TVPEField.AsNetDateTime

read / write; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory or the field contained pure text that could not be converted to a .NET date / time value).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldAsNetDateTime 836 and SetFieldAsNetDateTime 837.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.12 AsJavaDateTime

Sets / returns the value of a Field as Java date / time (number of milliseconds since midnight Jan 1, 1970).

property double TVPEField.AsJavaDateTime

read / write; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory or the field contained pure text that could not be converted to a Java date / time value).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldAsJavaDateTime 838 and SetFieldAsJavaDateTime 839.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.13 AsPythonDateTime

[Python only]

Sets / returns the value of a Field as Python date / time.

property double TVPEField.AsPythonDateTime

read / write; runtime only

Remarks:

In case of an error, an exception is thrown (for example out of memory or the field contained pure text that could not be converted to a Java date / time value).

The Template Object 816 itself offers simple methods to access a Field's value directly, i.e. without retrieving a Field Object first. See GetFieldAsPythonDateTime 840 and SetFieldAsPythonDateTime 841.

See also:

"dycodoc Template Processing" in the Programmer's Manual

30.14 Name

Returns the name of the Field as defined in *dycodoc*.

property string TVPEField.Name

read; runtime only

Returns:
the name of the Field as defined in *dycodoc*

See also:
"dycodoc Template Processing" in the Programmer's Manual

30.15 Description

Returns the description of the Field as defined in *dycodoc*.

property string TVPEField.Description

read; runtime only

Returns:
the description of the Field as defined in *dycodoc*

See also:
"dycodoc Template Processing" in the Programmer's Manual

30.16 DataSourceObject

Returns the Data Source Object where the Field belongs to.

property TVPEDataSource TVPEField.DataSourceObject

read; runtime only

Returns:
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
"Important Note for VPE-VCL Users" in the Programmer's Manual

Template Page Object

31 Template Page Object

[Enterprise Edition and above]

This section describes the methods and properties of the Page Object.

The class-name is TVPETemplatePage.

.NET: The TVPETemplatePage class is derived from the base class *Object*.

See also:

"dycodoc Template Processing" in the Programmer's Manual

31.1 ObjectHandle

This is the internal object handle of the VPE DLL for the TemplatePage Object.

The internal Object Handle is of interest to check if its value is null, which means that it is not initialized, i.e. it has no corresponding object in the DLL. This can happen in case of errors, i.e. if the object is not existing.

property long TVPETemplatePage.ObjectHandle

read; runtime only

Possible Values:

Value	Description
NULL	not initialized
NON-NULL	it holds a reference to an internal object in the VPE DLL

See also:

"dycodoc Template Processing" in the Programmer's Manual

31.2 PageWidth

Sets / returns the width of the specified template page.

property VpeCoord TVPETemplatePage.PageWidth

read / write; runtime only

See also:

"dycodoc Template Processing" in the Programmer's Manual

31.3 PageHeight

Sets / returns the height of the specified template page.

property VpeCoord TVPETemplatePage.PageHeight

read / write; runtime only

See also:

"dycodoc Template Processing" in the Programmer's Manual

31.4 Orientation

Sets / returns the orientation of the specified template page.

property PageOrientation [integer] TVPETemplatePage.Orientation

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VORIENT_PORTRAIT	1	Portrait	
VORIENT_LANDSCAPE	2	Landscape	

See also:

"dycodoc Template Processing" in the Programmer's Manual

31.5 PaperBin

Sets / returns the printer's input paper bin of the specified template page.

property PaperBin [integer] TVPETemplatePage.PaperBin

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VBIN_UNTOUCHED	0	Untouched	
VBIN_UPPER	1	Upper	
VBIN_ONLYONE	1	OnlyOne	
VBIN_LOWER	2	Lower	
VBIN_MIDDLE	3	Middle	
VBIN_MANUAL	4	Manual	
VBIN_ENVELOPE	5	Envelope	
VBIN_ENVMANUAL	6	EnvelopeManual	
VBIN_AUTO	7	Auto	
VBIN_TRACTOR	8	Tractor	
VBIN_SMALLFMT	9	SmallFormat	
VBIN_LARGEFMT	10	LargeFormat	
VBIN_LARGECAPACITY	11	LargeCapacity	
VBIN_CASSETTE	14	Cassette	

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

31.6 nLeftMargin, nTopMargin, nRightMargin, nBottomMargin

Sets / returns the position of the left, top, right and bottom margins of the specified template page.

See "Dynamic Positioning" and "Page Margins" in the Programmer's Manual for details.

property VpeCoord TVPETemplatePage.nLeftMargin
property VpeCoord TVPETemplatePage.nTopMargin
property VpeCoord TVPETemplatePage.nRightMargin
property VpeCoord TVPETemplatePage.nBottomMargin

read / write; runtime only

See also:

"dycodoc Template Processing" in the Programmer's Manual

31.7 VpeObjectCount

Returns the number of VPE Objects 884 in the template page.

property long TVPETemplatePage.VpeObjectCount

read; runtime only

Returns:
the number of VPE Objects in the template page

See also:
"dycodoc Template Processing" in the Programmer's Manual

31.8 VpeObject

Returns a VPE Object 884 from a template page.

```
method TVPEObject TVPETemplatePage.VpeObject(
        long Index
)
```

long Index

index into the array of available VPE Objects in the page
this value must be in the range between 0 and VpeObjectCount() 883 - 1

Returns:

a VPE Object from a template page

Remarks:

The function can be used to access VPE Objects within the template directly.

See also:

"dycodoc Template Processing" in the Programmer's Manual
"Important Note for VPE-VCL Users" in the Programmer's Manual

VPE Object

32 VPE Object

[Enterprise Edition and above]

This section describes the methods and properties of the VPE Object.

The class-name is TVPEObject.

.NET: The TVPEObject class is derived from the base class *Object*.

In addition the VPE Object has the following properties and methods:

PenSize 443, PenStyle 444, PenColor 445, SetPen 441, NoPen 442

BkgMode 451, TransparentMode 462, BkgColor 452, BkgGradientStartColor 453, BkgGradientEndColor 454, BkgGradientTriColor 455, BkgGradientMiddleColorPosition 456, BkgGradientMiddleColor 457, BkgGradientRotation 458, HatchStyle 463, HatchColor 464, CornerRadius 465

FontName 477, FontSize 478, CharSet 481, TextAlignment 486, TextBold 487, TextUnderline 488, TextUnderlined 489, TextStrikeOut 491, TextItalic 490, TextColor 492, SelectFont 476, SetFont 475, SetFontAttr 484

Rotation 398

PictureType 524, PicturePage 528, PictureEmbedInDoc 529, PictureKeepAspect 530, PictureBestFit 531, PictureX2YResolution 533, PictureDrawExact 534

SetBarcodeParms 545, BarcodeAlignment 548, BarcodeMainTextParms 546, BarcodeAddTextParms 547, BarcodeAutoChecksum 549, BarcodeThinBar 550, BarcodeThickBar 551

Bar2DAlignment 557

DataMatrixEncodingFormat 558
DataMatrixEccType 559
DataMatrixRows 560
DataMatrixColumns 561
DataMatrixMirror 562
DataMatrixBorder 563

QRCodeVersion 566
QRCodeEccLevel 567
QRCodeMode 568
QRCodeBorder 569

PDF417ErrorLevel 578
PDF417Rows 579
PDF417Columns 580

AztecFlags 584
AztecControl 585
AztecMenu 586
AztecMultipleSymbols 587
AztecID 588

Viewable 399, Printable 400, Streamable 402, Shadowed 403

UDOlParam 649

CharCount 795, DividerPenSize 796, DividerPenColor 797, AltDividerNPosition 798, AltDividerPenSize 799, AltDividerPenColor 800, BottomLinePenSize 801, BottomLinePenColor 802, DividerStyle 803, AltDividerStyle 804, FormFieldFlags 805

CheckmarkColor 911

See also:

"dycodoc Template Processing" in the Programmer's Manual

32.1 ObjectHandle

This is the internal object handle of the VPE DLL for the VPE Object.

The internal Object Handle is of interest to check if its value is null, which means that it is not initialized, i.e. it has no corresponding object in the DLL. This can happen in case of errors, i.e. if the object is not existing.

property long TVPEObject.ObjectHandle

read; runtime only

Possible Values:

Value	Description
NULL	not initialized
NON-NULL	it holds a reference to an internal object in the VPE DLL

See also:

"dycodoc Template Processing" in the Programmer's Manual

32.2 Kind

By using this function, you can identify the type of an object.

property Kind [long] TVPEObject.Kind

read; runtime only

Returns:

Possible return codes are:

ActiveX / VCL	Value	Enum	Comment
VOBJID_NULL	16	Null	does not occur
VOBJID_LINE	17	Line	
VOBJID_POLYLINE	18	Polyline	
VOBJID_FRAME	19	Frame	
VOBJID_TEXT	20	Text	
VOBJID_PICTURE	21	Picture	
VOBJID_BARCODE	22	Barcode	
VOBJID_ELLIPSE	23	Ellipse	
VOBJID_PIE	24	Pie	
VOBJID_POLYGON	25	Polygon	
VOBJID_RTF	26	RTF	
VOBJID_CHART	27	Chart	
VOBJID_UDO	28	UDO	
VOBJID_FORMFIELD	29	FormField	
VOBJID_RESERVED	30	----------	does not occur
VOBJID_DOCDATA	31	DocData	VPE document handle
VOBJID_CTRL_FORMFIELD	32	CtrlFormField	
VOBJID_CTRL_CHECKBOX	33	CtrlCheckbox	
VOBJID_CTRL_RADIOBUTTON	34	CtrlRadioButton	
VOBJID_CTRL_RADIOBUTTONGROUP	35	CtrlRadioButtonGroup	
VOBJID_RESERVED2	36	----------	does not occur
VOBJID_RESERVED3	37	----------	does not occur
VOBJID_DATA_MATRIX	38	DataMatrix	
VOBJID_MAXI_CODE	39	MaxiCode	

VOBJID_PDF417	40	Pdf417	
VOBJID_AZTEC	41	Aztec	
VOBJID_QRCODE	42	QRCode	

Example:

ActiveX - VB 6:

```
' Returns the type name of a given VPE Object-ID:
Private Function GetObjectKindName(obj As TVPEObject) As String
  Select Case obj.Kind
       Case VOBJID_LINE
            GetObjectKindName = "Line"
       Case VOBJID_POLYLINE
            GetObjectKindName = "Polyline"
       Case VOBJID_FRAME
            GetObjectKindName = "Frame"
       Case VOBJID_TEXT
            GetObjectKindName = "Text"
       Case VOBJID_PICTURE
            GetObjectKindName = "Picture"
       Case VOBJID_BARCODE
            GetObjectKindName = "Barcode"
       Case VOBJID_ELLIPSE
            GetObjectKindName = "Ellipse"
       Case VOBJID_PIE
            GetObjectKindName = "Pie"
       Case VOBJID_POLYGON
            GetObjectKindName = "Polygon"
       Case VOBJID_RTF
            GetObjectKindName = "RTF"
       Case VOBJID_CHART
            GetObjectKindName = "Chart"
       Case VOBJID_UDO
            GetObjectKindName = "UDO"
       Case VOBJID_FORMFIELD
            GetObjectKindName = "FormField"
       Case VOBJID_DOCDATA
            GetObjectKindName = "Document Data"
       Case VOBJID_CTRL_FORMFIELD
            GetObjectKindName = "FormField Control"
       Case VOBJID_CTRL_CHECKBOX
            GetObjectKindName = "Checkbox Control"
       Case VOBJID_CTRL_RADIOBUTTON
            GetObjectKindName = "RadioButton Control"
       Case VOBJID_CTRL_RADIOBUTTONGROUP
            GetObjectKindName = "RadioButton Group Control"
       Case VOBJID_DATA_MATRIX
            GetObjectKindName = "DataMatrix Barcode"
       Case VOBJID_MAXI_CODE
            GetObjectKindName = "MaxiCode Barcode"
       Case VOBJID_PDF417
            GetObjectKindName = "PDF417 Barcode"
       Case VOBJID_AZTEC
            GetObjectKindName = "Aztec Barcode"
       Case VOBJID_QRCODE
            GetObjectKindName = "QRCode Barcode"
       Case Else
            GetObjectKindName = "NULL"
  End Select
End Function
```

See also:

"dycodoc Template Processing" in the Programmer's Manual

32.3 Name

In *dycodoc* a unique name is assigned to each VPE Object. This function returns the name of a VPE Object as defined in *dycodoc*.

property string TVPEObject.Name

read; runtime only

Remarks:

This property does only return correct values for VPE Objects Field object 856 and Control Objects that reside in a template, and which have been inserted into the document using DumpTemplate() 811 / DumpTemplatePage() 812. Otherwise this property returns an empty string.
You may access this property also for a VPE object that has a NULL-Handle, in this case an empty string is returned.

See also:

"dycodoc Template Processing" in the Programmer's Manual

32.4 Text

Returns the text content of a Text object.

property string TVPEObject.Text

read; runtime only

Remarks:

This function does not support RTF objects.

The returned string does not contain Fields nor the values of Fields.
See ResolvedText 894.

See also:

"dycodoc Template Processing" in the Programmer's Manual

32.5 ResolvedText

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.

property string TVPEObject.ResolvedText

read; runtime only

Remarks:
This function does not support RTF objects.

See also:
"dycodoc Template Processing" in the Programmer's Manual

32.6 nLeft, nTop, nRight, nBottom

Returns the coordinates, i.e. the position, of the specified VPE Object.

property VpeCoord TVPEObject.nLeft
property VpeCoord TVPEObject.nTop
property VpeCoord TVPEObject.nRight
property VpeCoord TVPEObject.nBottom

read; runtime only

Remarks:

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() 447) require normalized coordinates.

See also:

"dycodoc Template Processing" in the Programmer's Manual

32.7 PictureFileName

Sets / returns the file name of a Picture 520 Object.

property string TVPEObject.PictureFileName

read / write; runtime only

Remarks:

Writing to this property (i.e. assigning a value to it) does only work for Picture Objects that reside in a template. If you specify a different object, the function does nothing.

See also:

"dycodoc Template Processing" in the Programmer's Manual

32.8 TemplateObject

Returns the template object where the given object belongs to.

property TVPETemplate TVPEObject.TemplateObject

read; runtime only

Remarks:

This property does only work for VPE Objects that reside in a template, for Field object 856 and Control Objects. If you read this property from a different object, an exception is thrown.

See also:

"dycodoc Template Processing" in the Programmer's Manual
"Important Note for VPE-VCL Users" in the Programmer's Manual

32.9 InsertedVpeObjectCount

After a call to DumpTemplate() 811 or DumpTemplatePage() 812 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.

property long TVPEObject.InsertedVpeObjectCount

read; runtime only

Returns:
the number of VPE Objects in the document that have been created from the provided template VPE Object

Remarks:
This property does only work for VPE Objects that reside in a template.

See also:
"dycodoc Template Processing" in the Programmer's Manual

32.10 InsertedVpeObject

Returns the n-th VPE Object in the document that has been created from the provided template VPE Object.

method TVPEObject TVPEObject.InsertedVpeObject (
    long *Index*
)

long Index
    index into the array of available VPE objects in the VPE Document
    this value must be in the range between 0 and InsertedVpeObjectCount 898 - 1

Returns:
    The n-th VPE Object in the document that has been created from the provided template VPE Object. If *Index* is out of range an exception is thrown.

Remarks:
    This method does only work for VPE Objects that reside in a template.

See also:
    "dycodoc Template Processing" in the Programmer's Manual
    "Important Note for VPE-VCL Users" in the Programmer's Manual

32.11 InsertedVpeObjectPageNo

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.

```
method long TVPEObject.InsertedVpeObjectPageNo(
        long Index
)
```

long Index

index into the array of available VPE objects in the VPE Document
this value must be in the range between 0 and InsertedVpeObjectCount 898 - 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 method does only work for VPE Objects that reside in a template.

See also:

"dycodoc Template Processing" in the Programmer's Manual

32.12 NextObject

Returns the succeeding VPE Object in the document.

property TVPEObject TVPEObject.NextObject

read; runtime only

Returns:

The succeeding VPE Object in the document. If there is no succeeding object the ObjectHandle 888 of the returned object is null.

Remarks:

This property allows in conjunction with the method FirstObject 410 to iterate through all objects of a page / document.

This function does only work for VPE Objects that reside in a VPE Document.

See also:

"dycodoc Template Processing" in the Programmer's Manual
"Important Note for VPE-VCL Users" in the Programmer's Manual

This page is intentionally left blank.

Interactive Objects

33 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.

33.1 FormFieldControl

Inserts either an *Interactive FormField* or an *Interactive Text* control into the document. The value of the property CharCount 795 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 793.

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).

```
method TVPEObject VPE.FormFieldControl(
        VpeCoord Left,
        VpeCoord Top,
        VpeCoord Right,
        VpeCoord Bottom,
        string Text
)
```

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

string 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 884 which represents the control. This VPE Object can be used later in your code to, for example, 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 474, Rich Text 612 and Picture 520 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 #13 + #10. (ASCII Code 13 plus ASCII Code 10)

Example:

```
// Insert an Interactive Text Control, where an unlimited number of
// characters may be entered:
Dim Control1 as TVPEObject
Dim Control2 as TVPEObject
Dim Control3 as TVPEObject
Doc.CharCount = 0
Control1 = Doc.FormFieldControl(1, 1, -9, VFREE,
           "Hello" + #10 + #13 + "World!")

// Insert an Interactive Text Control, limited to max. 14 characters:
Doc.CharCount = -14
Control2 = Doc.FormFieldControl(1, 2, -9, VFREE,
           "Hello" + #10 + #13 + "World!")

// Insert an Interactive Form Field Control, limited to max. 14
characters:
Doc.CharCount = 14
Control3 = Doc.FormFieldControl(1, 3, -9, VFREE, "Hello World!")
```

.NET:
use Doc.nFree 392 instead of VFREE in the example above

See also:
"Interactive Documents" in the Programmer's Manual.

33.2 Checkbox

Inserts a Checkbox into the document.

method TVPEObject VPE.Checkbox(
    VpeCoord *Left*,
    VpeCoord *Top*,
    VpeCoord *Right*,
    VpeCoord *Bottom*
)

VpeCoord *Left, Top, Right, Bottom*
  position and dimensions

Returns:
  The VPE Object 884 which represents the control. This VPE Object can be used later in your code to, for example, 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 474, Rich Text 612 and Picture 520 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:

```
Dim Control as TVPEObject
Control = Doc.Checkbox(1, 1, -0.5, -0.5)
```

See also:
  "Interactive Documents" in the Programmer's Manual.

33.3 RadioButtonGroup

Inserts a Radiobutton Group into the document. It is required that you insert a Radiobutton Group into the document before inserting Radiobuttons 909. 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.

```
method TVPEObject VPE.RadioButtonGroup(
        integer DefaultValue
)
```

integer DefaultValue

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 884 which represents the control.
The Object is required to associate Radiobuttons with it.
In addition the VPE Object can be used later in your code to, for example, 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 RadioButton 909

See also:

"Interactive Documents" in the Programmer's Manual.

33.4 RadioButton

Inserts a Radiobutton into the document. Before inserting a Radiobutton, you need to insert a Radiobutton Group 908 into the document.

method TVPEObject VPE.RadioButton(
TVPEObject *RadioButtonGroup*,
VpeCoord *Left*,
VpeCoord *Top*,
VpeCoord *Right*,
VpeCoord *Bottom*,
integer *Value*
)

TVPEObject RadioButtonGroup
the Radiobutton Group Object you want to associate the Radiobutton with

VpeCoord *Left, Top, Right, Bottom*
position and dimensions

integer Value
the value the Radiobutton shall represent within its group

Returns:
The VPE Object 884 which represents the control. This VPE Object can be used later in your code to, for example, 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 474, Rich Text 612 and Picture 520 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
Dim GroupObj as TVPEObject
GroupObj = Doc.RadioButtonGroup(2)

// Create three Radiobuttons and associate each with the group
Doc.RadioButton(GroupObj, 1, 1, -0.5, -0.5, 1)
Doc.Print(Doc.nRight + 15, VTOP, "lives in forrest")

Doc.RadioButton(GroupObj, VLEFT, Doc.nBottom + 0.5, -0.5, -0.5, 2)
Doc.Print(Doc.nRight + 15, VTOP, "lives on boat")

Doc.RadioButton(GroupObj, VLEFT, Doc.nBottom + 0.5, -0.5, -0.5, 3)
Doc.Print(Doc.nRight + 15, VTOP, "lives in house")
```

.NET:
use Doc.nTop and Doc.nLeft 388 instead of VTOP and VLEFT in the example above

See also:

"Interactive Documents" in the Programmer's Manual.

33.5 CheckmarkColor

Sets the checkmark color used for Checkboxes 907 and Radiobuttons 909.

property Color VPE.CheckmarkColor

read / write; runtime only

Possible Values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_BLACK

See also:
"Interactive Documents" in the Programmer's Manual.

33.6 DocContainsControls

Determines, whether the document contains controls (i.e. Interactive Objects) or not.

property boolean VPE.ContainsControls

read; runtime only

Returns:

Value	Description
True	yes, the document contains controls
False	no

See also:

"Interactive Documents" in the Programmer's Manual.

33.7 ControlsModified

VPE keeps track, if the content (or value) of any control in the document has been modified. This property allows to read and explicitly set or reset the modification state of the document.

property boolean VPE.ControlsModified

read / write; runtime only

Possible Values:

Value	Description
True	the document is modified
False	the document is not modified

See also:

"Interactive Documents" in the Programmer's Manual.

33.8 Interaction

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 SwapFileName 195. 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 OpenDoc() 189) and read the document file into memory using ReadDoc() 229.

property boolean VPE.Interaction

read / write; runtime only

Possible Values:

Value	Description
True	enable the whole document for editing
False	disable the whole document for editing

Default:
False

See also:
"Interactive Documents" in the Programmer's Manual.

33.9 SetFocusToFirstControl

The focus is set to the very first control in the document (i.e. the control with the lowest TAB-ID).

```
method boolean VPE.SetFocusToFirstControl(
)
```

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.

33.10 SetFocusControlByName

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 892 to each object. The name can be supplied to this method in order to set the focus to a named control.

method boolean VPE.SetFocusControlByName(
    TVPETemplate *objTemplate*,
    string *Name*
)

TVPETemplate objTemplate
    Template Object of the template which was dumped into the VPE Document

string 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:
Doc.SetFocusControlByName(objTemplate, "Street")
```

See also:
    "Interactive Documents" in the Programmer's Manual.

33.11 SetFocusControl

Sets the focus to a control.

method boolean VPE.SetFocusControl(
TVPEObject *objControl*
)

TVPEObject objControl
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:

```
Dim objControl As TVPEObject
objControl = Doc.FormFieldControl(1, 1, -9, VFREE, "Hello World!")
Doc.SetFocusControl(objControl);
```

.NET:
use Doc.nFree 392 instead of VFREE in the example above

See also:
"Interactive Documents" in the Programmer's Manual.

33.12 GetFocusControl

Returns the VPE Object which currently owns the focus.

method long VPE.GetFocusControl(
)

Returns:

Returns the Control Handle of the control which currently has the focus.
If no control owns the focus, the ObjectHandle of the returned object is null.

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 an empty
string
// is returned)

Dim control_name As String
control_name = Doc.GetFocusControl.Name
```

See also:

"Interactive Documents" in the Programmer's Manual.

33.13 FindControl

Searches by name the VPE Object (a control) which was inserted with DumpTemplate() 811 / DumpTemplatePage() 812 into the current document. Names 892 can only assigned to objects using dycodoc and templates.

This method belongs to the TVPETemplate Object!
Use a Template Object 816 of a template which has already been dumped into the VPE Document.

method TVPEObject VPE.FindControl(
string *Name*
)

string Name
the name of the control as assigned in dycodoc

Returns:
Returns the VPE Object of the control with the given name.
If no control with the given name was found, the ObjectHandle of the returned object is null.

Example:

```
// Retrieves the Control Handle of the object named "Street"
// and disables the control
objTemplate.FindControl("Street").ControlEnabled = False
```

See also:
"Interactive Documents" in the Programmer's Manual.

33.14 ControlEnabled

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.

This method belongs to TVPEObject.

property boolean VPE.ControlEnabled

read / write; runtime only

Possible Values:

Value	Description
True	the control is enabled for editing
False	the control is disabled for editing

Remarks:

You may access this property also for a VPE object that has a NULL-Handle, in this case a write access does nothing and a read access returns False.

Example:

```
// Retrieves the Control Handle of the object named "Street"
// and enables the control
objTemplate.FindControl("Street").ControlEnabled = True
```

See also:

"Interactive Documents" in the Programmer's Manual.

33.15 ControlTabIndex

Returns / modifies 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 FormFieldControl() 905, Checkbox() 907, 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.

This property belongs to TVPEObject.

property long VPE.ControlTabIndex

read / write; runtime only

Possible Values:

The Tab-Index of the specified control. If this property is used for a TVPEObject which does not represent a control, it will return null für reading and it will do nothing if you assign a value to it.

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.

You may access this property also for a VPE object that has a NULL-Handle, in this case a write access does nothing and a read access returns zero.

See also:

"Interactive Documents" in the Programmer's Manual.

33.16 ControlGroupObject

Returns the Group Object at which the specified control belongs to.

This property belongs to TVPEObject.

property TVPEObject VPE.ControlGroupObject

read; runtime only

Returns:

The Group Object the specified control belongs to.
If the specified control does not belong to a group, or if this property is used for a TVPEObject which does not represent a control, the ObjectHandle of the returned object is null.

Remarks:

You may access this property also for a VPE object that has a NULL-Handle, in this case an object with a NULL-Handle is returned.

See also:

"Interactive Documents" in the Programmer's Manual.

33.17 ControlFieldObject

If the specified control was created from a template using dycodoc and DumpTemplate() 811, its value might be associated with a field. In such case this property returns the associated Field object 856.

This property belongs to TVPEObject.

property TVPEField VPE.ControlFieldObject

read; runtime only

Returns:

The Field Object which is associated with the specified control.
If the specified control is not associated with a Field, or if this property is used for a TVPEObject which does not represent a control, the ObjectHandle of the returned object is null.

Remarks:

You may access this property also for a VPE object that has a NULL-Handle, in this case an object with a NULL-Handle is returned.

See also:

"Interactive Documents" in the Programmer's Manual.

33.18 ControlTplVpeObject

If the specified control in the document was created from a template using dycodoc and DumpTemplate() 811, 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.

This property belongs to TVPEObject.

property TVPEObject VPE.ControlTplVpeObject

read; runtime only

Returns:

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, the ObjectHandle of the returned object is null.

Remarks:

The property can be used to access VPE Objects within the template directly. Normally, there is no need to use this property.

You may access this property also for a VPE object that has a NULL-Handle, in this case an object with a NULL-Handle is returned.

See also:

"Interactive Documents" in the Programmer's Manual.

33.19 ControlAsString

Returns / sets the value of the specified control as string.

This property belongs to TVPEObject.

property string VPE.ControlAsString

read / write; runtime only

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 AsString 863.

In case of an error, an exception is thrown (for example, out of memory).

Example:

```
Dim RecipientCompany As String
Dim objControl As TVPEObject
objControl = Doc.FormFieldControl(1, 1, -9, VFREE, "Hello World!")
RecipientCompany = objControl.ControlAsString
objControl.ControlAsString = "IDEAL Software"
```

.NET:
use Doc.nFree 392 instead of VFREE in the example above

See also:

"Interactive Documents" in the Programmer's Manual.

33.20 ControlAsInteger

Returns / sets the value of the specified control as integer.

This property belongs to TVPEObject.

property string VPE.ControlAsInteger

read / write; runtime only

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 AsInteger 864.

In case of an error, an exception is thrown (for example out of memory or the field contained pure text that could not be converted to an integer value).

Example:

```
Dim value As Integer
Dim objControl As TVPEObject
objControl = Doc.FormFieldControl(1, 1, -9, VFREE, "123")
value = objControl.ControlAsInteger
objControl.ControlAsInteger = 456
```

.NET:
use Doc.nFree 392 instead of VFREE in the example above

See also:

"Interactive Documents" in the Programmer's Manual.

PDF Export

34 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 WriteDoc() 225.

For important details about PDF Export and the features provided by VPE, please see the Programmer's Manual, chapter "The PDF Export Module".

34.1 PDFVersion

Sets / returns the current version compatibility for exported PDF documents.

property PDFVersion [long] VPE.PDFVersion

read / write; runtime only

Possible Values:

ActiveX / VCL	Value	Enum	Comment
VPE_PDF_VERSION_1300	1300	Version1300	PDF Version 1.3 (Acrobat 4)
VPE_PDF_VERSION_1400	1400	Version1400	PDF Version 1.4 (Acrobat 5)
VPE_PDF_VERSION_1700	1700	Version1700	PDF Version 1.7

Default:

VPE_PDF_VERSION_1300

Remarks:

The version compatibility is especially important for the encryption key length 946 (see also Encryption 945). When you change this property, the encryption key length is set automatically to 40-bits for VPE_PDF_VERSION_1300 and to 128-bits for VPE_PDF_VERSION_1400 or higher.

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.

34.2 Author

property string VPE.Author

write; runtime only

Default:
not set (i.e. empty)

34.3 Title

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 Caption 236 property.

property string VPE.Title

write; runtime only

Default:

the value of the property Caption

34.4 Subject

Sets the Subject of the document. This value is displayed in Acrobat Reader's Document Properties dialog box.

property string VPE.Subject

write; runtime only

Default:
not set (i.e. empty)

34.5 Keywords

Sets the Keywords for the document. This value is displayed in Acrobat Reader's Document Properties dialog box.

property string VPE.Keywords

write; runtime only

Default:
not set (i.e. empty)

34.6 Creator

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.

property string VPE.Creator

write; runtime only

Default:
not set (i.e. empty)

34.7 EmbedAllFonts

[Not supported by the Community Edition]

Sets / returns the current setting 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.

property boolean VPE.EmbedAllFonts

read / write; runtime only

Possible values:

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 SetFontControl() 942.

VPE does only embed True-Type fonts, it will not embed PostScript fonts.

You can also subset embedded fonts. Subsetting means that VPE builds and embeds on-the-fly a new font, which does only contain the characters used in the current document. This creates in regular smaller fonts and therefore smaller PDF files. See SubsetAllFonts 941 for details.

34.8 DocExportPictureResolution

Sets / returns 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.

property integer VPE.DocExportPictureResolution

read / write; runtime only

Possible values:
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 DocExportPictureQuality 937 for details).

Non-Windows platforms: this option requires the Professional Edition or higher. For lower editions no down-scaling is performed.

34.9 DocExportPictureQuality

[VPE Professional Edition and above]

Sets / returns the current setting for the quality of images exported to non-VPE documents (like PDF, HTML, etc.).

property DocExportPictureQuality [integer] VPE.DocExportPictureQuality

read / write; runtime only

Possible values:

ActiveX / VCL	Value	Enum	Comment
PICEXP_QUALITY_NORMAL	0	Normal	embedded images will have normal quality [default]
PICEXP_QUALITY_HIGH	1	High	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 DocExportPictureResolution 936), 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.

34.10 FastWebView

[VPE Professional Edition and above]

Sets / 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.

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 UseTempFiles 940 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.

property boolean VPE.FastWebView

read / write; runtime only

Possible values:

Value	Description
True	create linearized PDF documents for Fast Web View

False	create normal documents

Default:
False

34.11 UseTempFiles

[VPE Professional Edition and above]

Sets / returns the current mode of operation, specifiying whether temporary swap files are used or not. This property is only in effect, if Fast Web View 938 is activated.

property boolean VPE.UseTempFiles

read / write; runtime only

Possible values:

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.

34.12 SubsetAllFonts

[VPE Professional Edition and above]

Sets the current mode for font subsetting. Only fonts that are embedded, can be subsetted (see EmbedAllFonts 935).

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.

property boolean VPE.SubsetAllFonts

read / write; runtime only

Possible values:

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 SetFontControl() 942.

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.

Example:

The following code activates font embedding, which is required for font-subsetting, and then activates font-subsetting.

```
VPE.EmbedAllFonts  = True
VPE.SubsetAllFonts = True
```

34.13 SetFontControl

[VPE Professional Edition and above]

Allows to set several parameters for an individual font. This function overrides the settings of EmbedAllFonts 935 and SubsetAllFonts 941.

method void VPE.SetFontControl(
    string *FontName*,
    string *SubstFontName*,
    boolean *DoEmbed*,
    boolean *DoSubset*
)

string FontName
  the font you want to set the parameters for

string SubstFontName
  the font with which you want to substitute the original font with

boolean DoEmbed

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

boolean DoSubset

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 ResetFontControl() 944.

Embedding for specific fonts can only be turned off, if global embedding is turned on (see EmbedAllFonts 935).

Subsetting for specific fonts can only be turned off, if global subsetting is turned on (see SubsetAllFonts 941).

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 SetFontSubstitution() 479, 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:

```
VPE.SetFontControl("Arial", "Helvetica", False, False)
VPE.SetFontControl("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.

Example 2:

```
VPE.EmbedAllFonts = True
VPE.SubsetAllFonts = True
VPE.SetFontControl("Arial", "Arial", True, False);
VPE.SetFontControl("Times New Roman", "Times New Roman", True, True);
VPE.SetFontControl("Verdana", "Verdana", False, False);
```

In the code above, no font substitution is used.
For Arial embedding is enabled and subsetting is disabled.
For Times New Roman embedding and subsetting is enabled.
For Verdana embedding and subsetting is disabled.

Note that setting EmbedAllFonts = True and SetSubsetAllFonts = True is required to be able to specify embedding and subsetting for specific fonts.

34.14 ResetFontControl

[VPE Professional Edition and above]

With each call to SetFontControl() 942, VPE adds internally a new font control structure to a list. ResetFontControl() clears the whole list, so all individual font control settings are deleted.

```
method void VPE.ResetFontControl(
)
```

34.15 Encryption

[VPE Professional Edition and above]

Activates or deactivates encryption for exported PDF documents.

property Encryption [integer] VPE.Encryption

read / write; runtime only

Possible values:

ActiveX / VCL	Value	Enum	Comment
DOC_ENCRYPT_NONE	0	Normal	no encryption [default]
DOC_ENCRYPT_STREAM	1	Stream	use stream encryption (compatible to Acrobat 4 and higher)

Default:
DOC_ENCRYPT_NONE

Remarks:
See EncryptionKeyLength 946, UserPassword 947, OwnerPassword 948 and Protection 949 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.

34.16 EncryptionKeyLength

[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 945.

property integer VPE.EncryptionKeyLength

read / write; runtime only

Possible values:

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 929 is VPE_PDF_ACROBAT_4 (default)

128 if PDFVersion is VPE_PDF_ACROBAT_5

Remarks:

The User Password 947 as well as the Owner Password 948 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 929, the encryption key length is adjusted accordingly and any previously assigned value is overwritten.

34.17 UserPassword

[VPE Professional Edition and above]

Sets the user password for the document. In order to be able to open an encrypted document (see Encryption 945), 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 OwnerPassword 948).

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 Protection 949).

property string VPE.UserPassword

read / write; runtime only

Possible values:

A string which may contain any characters. Up to 32 characters of the password are used for generating the encryption key (see EncryptionKeyLength 946).

Default:

not set (i.e. empty)

34.18 OwnerPassword

[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 UserPassword 947 and Protection 949).

property string VPE.OwnerPassword

read / write; runtime only

Possible values:

A string which may contain any characters. Up to 32 characters of the password are used for generating the encryption key (see EncryptionKeyLength 946).

Default:

not set (i.e. empty)

34.19 Protection

[VPE Professional Edition and above]

Sets the document protection flags. Document protection is only in effect, if encryption 945 is activated.

property Protection [long] VPE.Protection

read / write; runtime only

Possible values:

ActiveX / VCL	Value	Enum	Comment
PDF_ALLOW_NONE	0	PDFAllowNone	full protection, no flag is raised
PDF_ALLOW_PRINT	4	PDFAllowPrint	allow to print the document (see also the flag PDF_ALLOW_HIQ_PRINT)
PDF_ALLOW_MODIFY	8	PDFAllowModify	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	PDFAllowCopy	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	PDFAllowTaIff	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	PDFAllowFillIff	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	PDFAllowExtract	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	PDFAllowAssemble	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	PDFAllowHiQualityPrint	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	PDFAllowAll	no protection, all flags are raised

Default:
PDF_ALLOW_NONE

34.20 SetBookmarkDestination

[VPE Professional Edition and above]

Sets the destination for subsequently added bookmarks.

method void VPE.SetBookmarkDestination(
    BookmarkDestination [integer] *BookmarkDestination*,
    long *Left*,
    long *Top*,
    long *Right*,
    long *Bottom*,
    double *ZoomFactor*
)

BookmarkDestination [integer] BookmarkDestination
Bookmark Type, possible values are:

ActiveX / VCL Comment	**Value**	**Enum**
VBOOKMARK_DEST_NONE No Destination	**0**	None
VBOOKMARK_DEST_LTZ 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.	**1**	LeftTopZoom
VBOOKMARK_DEST_FIT 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.	**2**	Fit
VBOOKMARK_DEST_FITH 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.	**3**	FitHorizontally
VBOOKMARK_DEST_FITV	**4**	FitVertically

			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**	FitRectangle	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**	FitBounding	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**	FitBoundingHorizontally	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**	FitBoundingVertically	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.

long Left, long Top, long Right, long Bottom
position and dimensions

double ZoomFactor
zoom factor, e.g.: 0.5 = 50%; 1.0 = 100%; 2.0 = 200%; etc.

Default:
VBOOKMARK_DEST_LTZ

34.21 BookmarkStyle

[VPE Professional Edition and above]

Sets / returns the current bookmark style.

property BookmarkStyle [integer] VPE.BookmarkStyle

read / write; runtime only

Possible values:

any combination of the following flags, the values can be combined by adding there values:

ActiveX / VCL	Value	Enum	Comment
VBOOKMARK_STYLE_NONE	0	None	[default]
VBOOKMARK_STYLE_BOLD	1	Bold	PDF 1.4 or higher
VBOOKMARK_STYLE_ITALIC	2	Italic	PDF 1.4 or higher
VBOOKMARK_STYLE_OPEN	4	Open	

Default:

VBOOKMARK_STYLE_NONE

Example:

ActiveX / VCL:

```
// Bold and Italic styles will only work with PDF 1.4, activate it:
VPE.PDFVersion = VPE_PDF_ACROBAT_5

// Change the bookmark style to bold and italic:
VPE.BookmarkStyle = VBOOKMARK_STYLE_BOLD + VBOOKMARK_STYLE_ITALIC
```

.NET:

```
// Bold and Italic styles will only work with PDF 1.4, activate it:
VPE.PDFVersion = PDFVersion.Acrobat5

// Change the bookmark style to bold and italic:
VPE.BookmarkStyle = BookmarkStyle.Bold + BookmarkStyle.Italic
```

34.22 BookmarkColor

[VPE Professional Edition and above]

Sets the current bookmark color. Bookmark colors are supported by Acrobat Reader 5.0 (PDF v1.4) or higher.

property Color VPE.BookmarkColor

read / write; runtime only

Possible values:
any of the "COLOR_xyz" constants described in Programmer's Manual
or ActiveX / VCL: any RGB value
or .NET: any member of the .NET Color structure

Default:
COLOR_BLACK

34.23 AddBookmark

[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 bookmark will have the currently assigned destination, style and color (see SetBookmarkDestination() 951, BookmarkStyle 953, BookmarkColor 954).

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 *AddBookmark* method returns a handle to the newly created bookmark, which can be used as *parent* for subsequent calls.

method TVPEBookmark [pointer] VPE.AddBookmark(
    TVPEBookmark [pointer] *Parent*,
    string *Title*
)

TVPEBookmark [pointer] Parent
  the parent bookmark or null for a top-level bookmark

string Title
  the title of the bookmark

Returns:
  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 SwapFileName 195.

Example:

VCL:

```
// Bold and Italic styles will work only with PDF 1.4, activate it:
VPE.PDFVersion = VPE_PDF_ACROBAT_5

// Output some text on the current page:
VPE.VpePrint(1, 1, "Introduction")
```

```
// Set the style for newly added bookmarks to bold, italic, open:
VPE.BookmarkStyle = VBOOKMARK_STYLE_BOLD + VBOOKMARK_STYLE_ITALIC +
                    VBOOKMARK_STYLE_OPEN

// Set the destination type for newly added bookmarks:
VPE.SetBookmarkDestination(VBOOKMARK_DEST_FIT, 0, 0, 0, 0, 1)

// Add a new bookmark to the top-level of the hierachy:
Pointer ParentBookmark = VPE.AddBookmark(0, "Introduction")

// Append a new page to the VPE document:
VPE.PageBreak();

// Output some text on the current page:
VPE.VpePrint(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:
VPE.AddBookmark(ParentBookmark, "Bookmarks explained")
```

ActiveX, .NET, Java, ...:

```
// Bold and Italic styles will work only with PDF 1.4, activate it:
VPE.PDFVersion = PDFVersion.Acrobat5

// Output some text on the current page:
VPE.Print(1, 1, "Introduction")

// VB .NET
// Set the style for newly added bookmarks to bold, italic, open:
VPE.BookmarkStyle = BookmarkStyle.Bold + BookmarkStyle.Italic +
                    BookmarkStyle.Open

// C# (must use | operator)
// Set the style for newly added bookmarks to bold, italic, open:
VPE.BookmarkStyle = BookmarkStyle.Bold | BookmarkStyle.Italic |
                    BookmarkStyle.Open

// Set the destination type for newly added bookmarks:
VPE.SetBookmarkDestination(BookmarkDestination.Fit, 0, 0, 0, 0, 1)

// Add a new bookmark to the top-level of the hierachy:
// (for Visual Basic .NET use "Nothing" instead of "null")
TVPEBookmark ParentBookmark;
ParentBookmark = VPE.AddBookmark(null, "Introduction")

// Append a new page to the VPE document:
VPE.PageBreak();

// Output some text on the current page:
VPE.Print(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:
VPE.AddBookmark(ParentBookmark, "Bookmarks explained")
```

For Python use "None" instead of “null” in the call to *AddBookmark()*.

For Visual Basic 6.0 use “Nothing” instead of “null” in the call to *AddBookmark()*.

34.24 PDFALevel

[VPE Professional Edition and above]

VPE supports the export to the PDF/A format for long-term archival, according to ISO standard ISO 19005-1:2005, PDF/A-1b.

property PDFALevel [long] VPE. PDFALevel

write; runtime only

Possible values:

ActiveX / VCL	Value	Enum	Comment
VPE_PDF_A_LEVEL_NONE	0	LevelNone	No PDF/A document is created. [default]
VPE_PDF_A_LEVEL_1B	1	Level1B	A PDF/A-1b document shall be created.
VPE_PDF_A_LEVEL_3B	3	Level3B	A PDF/A-3b document shall be created.

Default:

VPE_PDF_A_LEVEL_NONE

Remarks:

The PDF/A standard enforces several other document properties. When you set this property to VPE_PDF_A_LEVEL_nB, VPE sets the following properties automatically:

for VPE_PDF_A_LEVEL_1B: PDFVersion = VPE_PDF_VERSION_1400
for VPE_PDF_A_LEVEL_3B: PDFVersion = VPE_PDF_VERSION_1700
EmbedAllFonts = true
SubsetAllFonts = true
Encryption = DOC_ENCRYPT_NONE

By default, font subsetting is enabled. If you disable it, and you are using symbolic fonts, like Windings or Webdings, you must enable subsetting at least for those fonts (see SetFontControl 942), because the PDF/A standard requires that symbolic fonts only contain one CMAP table. But the fonts come in regular with multiple CMAP tables. The font subsetter of VPE strips the unwanted CMAP tables.

Furthermore you need to add a Color Profile to the PDF document, see the method AddColorProfile() 959.

Example:

```
Doc.PDFALevel = VPE_PDF_A_LEVEL_3B

Doc.AddColorProfile(
   "GTS_PDFA1",
   "sRGB_IEC61966-2-1",
   "Custom",
   "http://www.color.org",
   "sRGB_IEC61966-2-1",
   "sRGB_IEC61966-2-1")

Doc.WriteDoc("test.pdf")
```

34.25 AddColorProfile

[VPE Professional Edition and above]

Adds a Color Profile to the exported PDF document. A Color Profile is required when exporting to the PDF/A document format for long-term archival

```
method void VPE.AddColorProfile(
        string Subtype,
        string OutputCondition,
        string OutputConditionIdentifier,
        string RegistryName,
        string Info,
        string FileName
)
```

string Subtype
the sub-type of the color profile. See the PDF specification for details.

string OutputCondition
the output condition for the color profile. Use "GTS_PDFA1", see the PDF specification for details.

string OutputConditionIdentifier
the output condition identifier of the color profile. Use "Custom", see the PDF specification for details.

string RegistryName
the registry name of the color profile. See the PDF specification for details.

string Info
the info dictionary entry of the color profile. See the PDF specification for details.

string FileName
the path and file name of the color profile file. VPE reads this file from hard disk end embeds it into the PDF document. VPE does not verify or convert the given file.

For convenience, VPE provides two built-in color profiles:
"sRGB_IEC61966-2-1" and "AdobeRGB1998"
If you specify either of both strings as FileName parameter, VPE will embed this profile from its internal memory. No external files are required.

Remarks:

VPE does not convert images to the given Color Profile. When adding images to a VPE document, you need to ensure that they match the Color Profile. Special care must be taken when using JPEG images: VPE embeds JPEG images in their original binary file format into PDF streams. That means, JPEG images in the CMYK format are embedded as CMYK into the document. Make sure they are converted to the given Color Profile, before importing them into a VPE document.

Example:

```
Doc.PDFALevel = VPE_PDF_A_LEVEL_1B

Doc.AddColorProfile(
   "GTS_PDFA1",
   "sRGB_IEC61966-2-1",
   "Custom",
   "http://www.color.org",
   "sRGB_IEC61966-2-1",
   "sRGB_IEC61966-2-1")

Doc.WriteDoc("test.pdf")
```

34.26 PDFEmbedFile

[VPE Professional Edition and above]

Embeds a file into the generated PDF document.

Using PDF viewer applications like Acrobat Reader or Foxit PDF Reader, users can extract embedded files to their hard drive.

method void VPE.PDFEmbedFile(
    string *FileName,*
    string *InternalName,*
    string *Description,*
    string *MimeType,*
    string *AFRelation*
)

string FileName
full path and file name of file on hard-disk

string InternalName
as used inside PDF, if e-invoice, must be "factur-x.xml", for XRechnung: "xrechnung.xml"
should only contain ASCII characters 33 - 126 and no slash "/" or backslash "\"

string Description
any descriptive text

string MimeType
MIME Type, e.g. "text/xml"

string AFRelation
see /AFRelationship in PDF Filespec dict in PDF spec and ZUGfERD / Factur-X spec

Remarks:
Linearized PDF (Fast Web View) is always turned off when embedding files.

Sets LastError.

Example:

```
Doc.PDFEmbedFile(
  "c:\\data\\test3.html",
  "test.html",
  "third test performed today",
  "text/html",
  "Data")

Doc.WriteDoc("test.pdf")
```

34.27 PDFClearEmbeddedFiles

[VPE Professional Edition and above]

Clears the list of embedded files.

method void VPE.PDFClearEmbeddedFiles()

Example:

```
Doc.PDFEmbedFile(
   "c:\\data\\test3.html",
   "test.html",
   "third test performed today",
   "text/html",
   "Data")

Doc.WriteDoc("test.pdf")

Doc.PDFClearEmbeddedFiles()
```

34.28 SetFacturXParams

[VPE Professional Edition and above]

Sets parameters for an embedded Factur-X, ZUGfERD, XRechnung file.

method void VPE.SetFacturXParams(
string *InternalName,*
string *DocumentType,*
string *Version,*
string *ConformanceLevel*
)

string InternalName
as used inside PDF, if e-invoice, must be "factur-x.xml", for XRechnung: "xrechnung.xml" should only contain ASCII characters 33 - 126 and no slash "/" or backslash "\"

string DocumentType
always "INVOICE" (see ZUGfERD / Factur-X spec)

string Version
"1.0" (see ZUGfERD / Factur-X spec)

string ConformanceLevel
MINIMUM, BASIC WL, BASIC, EN 16931, EXTENDED, XRECHNUNG (see ZUGfERD / Factur-X spec)

Example:

In the following example, a Factur-X / ZUGfERD conformant PDF file is generated:

```
Doc.SetPDFALevel(VPE_PDF_A_LEVEL_3B)

Doc.AddColorProfile(
   "GTS_PDFA1",
   "sRGB_IEC61966-2-1",
   "Custom",
   "http://www.color.org",
   "sRGB_IEC61966-2-1",
   "sRGB_IEC61966-2-1")

Doc.PDFEmbedFile(
   "c:\\data\\factur-x-0327.xml",
   "factur-x.xml",
   "electronic invoice",
   "text/xml",
   "Alternative")

// note that InternalName is identical to the InternalName used for
the call to PDFEmbedFile()
Doc.SetFacturXParams("factur-x.xml", "INVOICE", "1.0", "EN 16931")

Doc.WriteDoc("test.pdf")
```

34.29 ExtIntDA

Used internally for diagnostic tasks. **Do not use!**

method long VPE.ExtIntDA(
long *i*,
long *s*
)

HTML Export

35 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 WriteDoc() 225.

The following properties are written as (meta-)tags to generated HTML documents:

Title 931 as <title> tag

Creator 934 as "generator" meta-tag

Author 930 as "author" meta-tag

Keywords 933 as "keywords" meta-tag

Subject 932 as "description" meta-tag

In addition, the settings for DocExportPictureQuality 937 and DocExportPictureResolution 936 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 936, PictureExportColorDepth 666 and PictureExportDither 667.

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".

35.1 HtmlScale

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.

property double VPE. HtmlScale

read / write; runtime only

Possible Values:
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

35.2 HtmlWordSpacing

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.

property double VPE. HtmlWordSpacing

read / write; runtime only

Possible Values:

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

35.3 HtmlRtfLineSpacing

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.

property boolean VPE. HtmlRtfLineSpacing

read / write; runtime only

Possible Values:

Value	Description
True	shrink the line spacing for viewing with Mozilla browsers
False	do not change the line spacing

Default:

False

35.4 HtmlCopyImages

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.

property boolean VPE. HtmlCopyImages

read / write; runtime only

Possible Values:

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.

35.5 HtmlPageBorders

If true, borders are added at page boundaries, i.e. a frame is drawn at the page boundaries.

property boolean VPE. HtmlPageBorders

read / write; runtime only

Possible Values:

Value	Description
True	add borders at page boundaries
False	do not draw borders at page boundaries

Default:

False

35.6 HtmlPageSeparators

If true, pages are separated by horizontal rules.

property boolean VPE. HtmlPageSeparators

read / write; runtime only

Possible Values:

Value	Description
True	separate pages by horizontal rules
False	do not separate pages by horizontal rules

Default:
False

XML Export

36 XML Export

This section describes the methods and properties related to XML Export.

A VPE Document is exported to the XML file format by calling the method WriteDoc() 225
.

The XML Document has a following hierarchic structure.

- VPE
  - DOC
    - FileDictionary
      - Image
    - Page
      - VpeObjects (i.e. Line, PolyLine, Picture …)

36.1 XMLPictureExport

Specifies the mode for handling images of generated XML documents.

If set to true, VPE copies / creates images in destination directory named "<XML 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 BMP, TIF, GIF, PCX, JPG, PNG and ICO, the original image is copied. Otherwise a compressed PNG is created from the original. The file path is written to the XML stream in the object block of the picture. The FileDictionary block of the XML stream is empty.

If set to false, pictures are encoded into Base-64. The Base-64 code will be written into the FileDictionary block of the XML stream with the relative path as the key. The encoded image will be embedded only once within the XML stream.

property boolean VPE. XmlPictureExport

read / write; runtime only

Possible Values:

Value	Description
True	Copy / create external images
False	Encode images into Base-64 and embed them within the XML stream.

Default:
False

Remarks:
This property should be set to false, if exporting XML to a memory stream.

ODT Export

37 ODT Export

This section describes the methods and properties related to ODT Export. OpenDocument Text is a free and open document standard, which is published as an ISO/IEC international standard, ISO/IEC 26300:2006 Open Document Format for Office Applications (OpenDocument) v1.0. Published ODF standards meet the common definitions of an open standard, meaning they are freely available and implementable.

ODT files can be imported by Microsoft Office 2003 and later through add-ons.

A VPE Document is exported to the ODT file format by calling the method WriteDoc() 225.

The ODT Document is a zip-File, which includes a group of xml-files and picture files. The following example shows the structure of an exported ODT Document.

- ODT
  - META-INF\manifest.xml
  - Pictures\000000000001.png
  - Meta.xml
  - Settings.xml
  - Content.xml
  - Styles.xml

META-INF\manifest.xml

Manifest.xml lists the all files in this archives format again. If this file is missing, this document will not be represented correctly.

Folder “Pictures”

The used images are stored in the folder Pictures as copies.

Meta.xml

The brief information of the ODT Document are saved here. For example, the generator, the create time of this document, etc.

Settings.xml

The document-specific attitudes are saved in settings.xml.

Styles.xml

All used document formats are stored in styles.xml. For example, fonts, line styles, paragraph styles, and page layout, etc.

Content.xml

The file content.xml contains text contents of the document.

Please see this webpage (http://en.wikipedia.org/wiki/OpenDocument) for more information of the Open Document.

37.1 OdtTextWidthScale

Specifies the width scaling for text. It represents a value in percent. Normally, this value should be set to 1.0.

This is a global value, which applies to all exported text objects.

property double VPE.OdtTextWidthScale

read / write; runtime only

Possible Values:

The text width scale of the generated ODT document. If the value is below 1.0, characters are positioned more closely. When the scale is above 1.0, characters are expanded.

Default:

1.0, which equals 100% of the original text width

37.2 OdtTextPtSizeScale

Specifies the scaling of the point size for text.

This is a global value, which applies to all exported text objects.

property double VPE.OdtTextPtSizeScale

read / write; runtime only

Possible Values:

The scale of the text point size of the generated ODT document. If the value is below 1.0, text is shrinked. When the scale is above 1.0, text is expanded.

Default:

0.98, which equals 98% of the original text point size

37.3 OdtLineHeight

Specifies the scale of the text line height. It represents a value in percent. Normally, this value should be set to 1.0.

This is a global value, which applies to all exported text objects.

property double VPE.OdtLineHeight

read / write; runtime only

Possible Values:

The text line height of the generated ODT document. If the value is below 1.0, the distance between two lines of text is shrinked. If the scale is above 1.0, the distance between two lines of text is expanded.

Default:

1.0, which equals 100% of the original line distance.

37.4 OdtAutoTextboxHeight

Open Office and VPE render text differently. Therefore exported ODT documents do not look 100% identically to VPE documents. Sometimes a text box of Open Office is not capable of displaying all the text it contains, although VPE can. With this parameter you can specify, whether a text box in ODT is drawn with the same height as in VPE, or automatically with a compatible height. If this property is set to true, the text box height specified in VPE is used as the minimal height of the exported text box.

This is a global value, which applies to all exported text objects.

property bool VPE.OdtTextboxHeight

read / write; runtime only

Possible Values:

Value	Description
True	Export text boxes with compatible height, so that all the text can be displayed.
False	Text boxes are exported with the same height as specified in VPE.

Default:

True

37.5 OdtPositionProtect

VPE can draw text boxes as high as a whole page. If such a text box is exported to an ODT document, the position of the text box might be moved by Open Office automatically. In order to protect the positions of exported text boxes, you can set this property to true.

This is a global value, which applies to all exported text objects.

property bool VPE.OdtPositionProtect

read / write; runtime only

Possible Values:

Value	Description
True	The position of exported text boxes is protected.
False	The position of exported text boxes is not protected.

Default:

True

DLL vs ActiveX / VCL

38 DLL vs ActiveX / VCL

This section is for users that are working with the DLL and want to use the ActiveX / VCL, or vice versa.

Instead of properties and methods the DLL offers a set of functions. The events are the same (notification messages sent via "SendMessage()").

The DLL functions all start with the prefix "Vpe" and require a document handle. Since the ActiveX and VCL use the syntax "<Object> . <Method>" the document handle is not required and the prefix "Vpe" is left out.

Example:

DLL:
```
VpeLine( hDoc, x, y, x2, y2 )
```

Control:
```
<Object>.Line( x, y, x2, y2 )
```

(Visual Basic has problems with the keywords "Print, Write, Line and Scale". VB doesn't recognize that these are methods and properties of an ActiveX. So we implemented them twice: in VB use "VpePrint, VpeWrite, VpeLine and VpeScale".)

All DLL functions, which just set or get a single value, are defined as properties in the ActiveX / VCL.

Example:

DLL:
```
VpeSetPenSize( hDoc, 10 )
```

Control:
```
<Object>.PenSize = 10
```

Example:

DLL:
```
ps = VpeGetPenSize( hDoc )
```

Control:
```
ps = <Object>.PenSize
```

- B -