OCX Development

This version of VisualWriter is designed for developers creating 16- or 32-bit OCX applications in Visual Basic 4.0, Visual C++ 4.0, Access95, or other environments that support OCX containers.

VisualWriter provides one OCX for use in developing 16-bit applications (VW16.OCX) and another OCX for use in developing 32-bit applications (VW32.OCX.)

Visual Basic was chosen for most of the code examples in this help because of its wide acceptance and ease of use. If you are developing applications in Visual C++, refer to the section Using VisualWriter in Visual C++.

VisualWriters major features include:

- **Flexible implementation**. With VisualWriter, you can add a simple text editor to your application or a full-featured word processor with abilities to create, format, and edit text documents that rival commercial word processors.
- **Auxiliary controls.** VisualWriter provides tools for creating button bars, status bars, and rulers that work with your VisualWriter text controls.
- **RTF support**. In addition to VisualWriters native document format, VisualWriter can load and save text documents in rich text format (RTF).
- **Multiple control support**. You can create applications that feature multiple VisualWriter controls within a form. In addition, you can use a VisualWriter control as a container for other VisualWriter controls.
- **Spell checking with VisualSpeller**. VisualWriter provides two properties that allow you to integrate the full power of VisualSpellers robust spell checking into your text application.
- **Image embedding.** TIFF, BMP, and WMF images can be included in your text documents.
- **Database access.** VisualWriter can be used as a bound control, allowing interaction with Access databases.

Company Commitment

The phenomenal increase in computing power in recent years has given rise to applications that are increasingly complex. This trend has made it all but impossible to develop applications from raw code. Instead, developers need high-quality tools if they are to build world class applications quickly and efficiently. Visual Components, Inc. was formed in February 1993, to supply those tools. Our guiding principles are:

Integrity is our highest concern

Customers are the reason we exist

Quality is not available for compromise

Employee involvement is our basic business model

Getting Technical Support

The Visual Components technical support staff can help you with any problem you encounter installing or using VisualWriter. If you need assistance, contact Visual Components in any of the following ways:

By telephone. You can contact our technical support staff at (913)599-6500 on weekdays between 8:30 a.m. and 5:30 p.m., central time.

By FAX. You can contact us by FAX at (913)599-6597.

On the World Wide Web. Contact us at:

```
www.visualcomp.com
```

Via BBS. You can contact us through our 24-hour bulletin board service at (913)599-6713.

By mail. Address your correspondence to:

Technical Support Department
Visual Components, Inc.
15721 College Blvd.
Lenexa, Kansas 66219

In Europe, contact:

Visual Components Europe
Lenexa House
11 Eldon Way
Paddock Wood, Kent
England TN12 6BE
Tel: +44 1892 834343
Fax: +44 1892 835843
BBS: +44 1892 835579

Adding the OCX to Your Application

The process you use to add an OCX to your application varies slightly from one development environment to another. In most cases it consists of:

- Adding the OCX control to your project.
- Selecting the controls tool from the tool bar and drawing the control on a form or in a window.

Consult your development environment documentation for specific steps to add a control to your application. If you are developing applications in Visual C++, refer to the section Using VisualWriter in Visual C++.

Distributing VisualWriter Applications

Please read the license agreement that was shipped with this package. You are bound by the licensing agreement contained in that document.

Redistributing Files

You can use all the files accompanying this product for development of an application. You can redistribute the run time version of the software according to the terms of the license agreement.

The following list of files are important. Read the paragraph below to learn what each file is used for.

VisualWriter can only be used in design mode if the license file VW32.LIC is present in the same directory as the custom control file VW32.OCX. This license file is not required for the operation of compiled programs (EXE files), and may not be distributed with your applications.

	16-bit Files	**32-bit Files**
1	VW16.OCX	VW32.OCX
	VW.HLP	VW.HLP
2	OC25.DLL	MFC40.DLL
		MSVCRT40.DLL
3	OLE2DISP.DLL	OLEAUT32.DLL
	TYPELIB.DLL	
	COMPOBJ.DLL	
4	KERNEL.DLL	KERNEL32.DLL
	USER.DLL	USER32.DLL
	GDI.DLL	GDI32.DLL
	WIN87EM.DLL	
	KEYBOARD.DLL	

This first group of files are the VisualWriter redistributable files. These files may be in the Windows System directory, on the computers Path, or for WinNT and Win95, in the directory specified by your applications Per Application Path key in the Registry Database.

The second group of files are Microsoft redistributable files necessary to for this OCX to operate. Your programming environment should have installed and registered these files. If they were not, or they were an older version, the VisualWriter installer updated and registered them. These files were then copied to ReDist16 and ReDist32 in the installation directory, depending on which installation options were chosen. You may run the installer again and install only these files.

The third group of files are installed when support for OLE/ODBC is installed. You should verify that these files are installed on your clients system and refer to Microsoft redistribution policy if you need to redistribute them. Some of these files need to be registered before use. You can use REGSVR.EXE or REGSVR32.EXE to register them. The result code from REGSVR(32) will indicate whether the file needs to be registered. The fourth group of files should be present on any system running Windows. You should not remove or update these files. They are included only to form a complete list of files needed to use this OCX.

VisualWriter Properties, Events and Methods

The following section contains an alphabetical reference of all the properties, events, and methods for VisualWriter. Included in this section are the standard Visual Basic properties, events and methods used by VisualWriter.

The Visual Basic properties, events and methods have been documented according to the information currently available in the Visual Basic reference manual. For more information on these Visual Basic properties and events, refer to your Visual Basic documentation.

The table below lists all the properties, events and methods for VisualWriter.

Align Property	Alignment Property	BackColor Property
BackStyle Property	BaseLine Property	BorderStyle Property
ButtonBarHandle Property	CanRedo Property	CanUndo Property
Change Event	Click Event	Clip Method
ClipChildren Property	ClipSiblings Property	ControlChars Property
CurrentPages Property	DataChanged Property	DataField Property
DataFormat Property	DataSource Property	DataText Property
DblClick Event	DragDrop Event	DragIcon Property
DragMode Property	DragOver Event	EditMode Property
Enabled Property	Error Event	FieldChangeable Property
FieldChanged Event	FieldClicked Event	FieldCreated Event
FieldCurrent Property	FieldDblClicked Event	FieldDelete Method
FieldDeleteable Property	FieldDeleted Event	FieldEnd Property
FieldInsert Method	FieldPosX Property	FieldPosY Property
FieldSetCursor Event	FieldStart Property	FieldText Property
FindReplace Method	FontBold Property	FontDialog Method
FontItalic Property	FontName Property	FontSize Property
FontStrikethru Property	FontUnderline Property	ForeColor Property
FormatSelection Property	FrameDistance Property	FrameLineWidth Property
FrameStyle Property	GotFocus Event	Height Property
HelpContextID Property	HideSelection Property	HScroll Event
hWnd Property	IndentB Property	IndentFL Property
IndentL Property	IndentR Property	IndentT Property
Index Property	InsertionMode Property	KeyDown, KeyUp Events
KeyPress Event	Language Property	Left Property
LineSpacing	LineSpacingT Property	Load Method
LostFocus Event	MouseDown, MouseUp Events	MouseMove Event
MousePointer Property	Move Event	Move Method
Name Property	NextWindow Property	ObjectClicked Event
ObjectCreated Event	ObjectCurrent Property	ObjectDblClicked Event

Align Property

Description

This is a standard Visual Basic property.

Determines whether a text control can appear in any size anywhere on a form or whether it appears at the top or bottom of the form automatically sized to fit the forms width.

Syntax

[form.]*VisualWriter.***Align** [= *numericexpression*]

Remarks

The Align property settings are:

Setting	Description
0	None size and location can be set at design time or in code. Ignored if the text control is on an MDI form.
1	Top text control is at the top of the form and its width is equal to the forms ScaleWidth property setting.
2	Bottom text control is at the bottom of the form and its width is equal to the forms ScaleWidth property setting.

Data Type

Integer

Alignment Property

Description

Specifies the alignment of text in VisualWriter.

Syntax

[form.]*VisualWriter.***Alignment** [= *alignment*]

Remarks

The Alignment property settings are:

Setting	Description
0	Text is left aligned. (Default)
1	Text is right aligned.
2	Text is centered.
3	Text is justified.
4	This value cannot be assigned to the property. Its purpose is to indicate that the selected text contains paragraphs which have different types of alignment.

If the FormatSelection property has previously been set to True, changing the Alignment property affects only the currently selected paragaph. If FormatSelection has been set to False, the setting applies to the entire control, in which case a value of 4 does not occur.

Data Type

Integer

BackColor Property

Description

This is a standard Visual Basic property.

Determines the background color of a text control.

Syntax

[form.]*VisualWriter.***BackColor** [= *color*]

Remarks

Visual Basic uses the Microsoft Windows environment RGB scheme for colors. Each property has the following range of settings:

Range of Settings	Description
Normal RGB colors	Colors specified by using the color palette, or by using the RGB or QBColor functions in code.
System default colors	Colors specified with system color constants from CONSTANT.TXT, a Visual Basic file that specifies system defaults. The Windows environment substitutes the users choices as specified in the users Control Panel settings.

Data Type

Long

See Also

ForeColor Property

BackStyle Property

Description

This is a standard Visual Basic property.

Determines whether the background of a text control is transparent or opaque.

Syntax

[form.]*VisualWriter.***BackStyle** [= *numericexpression*]

Remarks

The BackStyle Property settings are:

Setting	Description
0	Transparent background color and any graphics are visible behind the control.
1	(Default) Opaque the controls BackColor fills the control and obscures any color or graphics behind it.

Data Type

Integer

See Also

ForeColor property

BaseLine Property

Description
Specifies the baseline alignment for selected text. A negative value is used to specify a subscript offset, a positive value for superscript. VisualWriter limits the baseline alignment to 960 twips in both directions.

Syntax
[form.]*VisualWriter*.**BaseLine** [= *baseline alignment*]

Data Type
Integer

BorderStyle Property

Description

This is a standard Visual Basic property.

Determines the border *style* for a text control. Read-only at run-time.

Syntax

[form.]*VisualWriter.***BorderStyle** [= *style*]

Remarks

The BorderStyle property settings are:

Settings	Description
0	(Default for image and label) None
1	(Default for text control and text box) Fixed Single.

Data Type

Integer

ButtonBarHandle Property

Description
Specifies the button bar control to be used with VisualWriter. Not available at design time.

Syntax

[form.]*VisualWriter.***ButtonBarHandle** [= *ButtonBar.hWnd*]

Data Type
Integer

See also
RulerHandle Property

StatusBarHandle Property

CanRedo Property

Description

Informs whether an operation can be redone using the Redo method. Not available at design time; read-only at run time.

Syntax

[form.]*VisualWriter.***CanRedo**

Remarks

The CanRedo property has one of the following values:

Setting	Description
0	Nothing to be undone.
10	The next undo operation deletes inserted text.
11	The next undo operation inserts deleted text.
12	The next undo operation resets the last formatting operation.

See also

CanUndo Property

Undo Method

Redo Method

Example

See MDIDEMO sample program.

CanUndo Property

Description

Informs whether an operation can be undone using the Undo method. Not available at design time; read-only at run time.

Syntax

[form.]*VisualWriter.***CanUndo**

Remarks

The CanUndo property has one of the following values:

Setting	Description
0	Nothing to be undone.
1	The next undo operation deletes inserted text.
2	The next undo operation inserts deleted text.
3	The next undo operation resets the last formatting operation.

See also

CanRedo Property

Undo Method

Redo Method

Example

See MDIDEMO sample program.

Change Event

Description

This is a standard Visual Basic event.

Indicates that the contents of a control have changed. This event occurs when a DDE link updates data, when a user changes the text, or when you change the **Text** property setting through code.

Syntax

Private Sub *object*_**Change**([*index* As Integer])

The Change event syntax has these parts:

Part	Description
index	An integer that uniquely identifies a control if its in a control array.

See also

KeyDown Event

KeyUp Event

KeyPress Event

LostFocus Event

Text Property

Click Event

Description

This is a standard Visual Basic event.

Occurs when the user presses and then releases a mouse button over an object. It can also occur when the value of a control is changed.

Syntax

Private Sub Form_**Click** ()

Private Sub *object*_**Click** ([*index* As Integer])

The Click event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if its in a control array.

Remarks

Typically, you attach a Click event procedure to a **CommandButton** control, **Menu** object, or **PictureBox** control to carry out commands and command-like actions. For the other applicable controls, use this event to trigger actions in response to a change in the control.

See also

DblClick Event

MouseDown, MouseUp Events

Clip Method

Description

Performs VisualWriter clipboard actions.

Syntax

[form.]*VisualWriter.***Clip** action

Remarks

The parameter can have one of the following values:

Setting	Description
1	Cut out the selected text and copy it to the clipboard.
2	Copy the selected text to the clipboard.
3	Paste text from the clipboard.
4	Clear the selection.

Data Type

Integer

Example

This example copies the selected text from the control named VisualWriter1 to the clipboard when the user selects the Edit/Copy menu item:

```
Private Sub mnuEdit_Copy_Click ()

    VisualWriter1.Clip 2

End Sub
```

ClipChildren Property

Description
The ClipChildren Property is only used for text controls which act as a container for other text controls or embedded objects. When ClipChildren set to True, the areas occupied by the child controls are excluded from the update area.

Syntax
[form.]*VisualWriter.***ClipChildren** [= *boolean*]

Remarks
The ClipChildren Property settings are:

Setting	Description
True	Exclude areas which are occupied by child controls from the update area.
False	Update areas which are occupied by child controls (Default).

Data Type
Boolean

See also
ClipSiblings Property

Example
See Forms2 sample program.

ClipSiblings Property

Description

The ClipSiblings property determines the clipping behavior of each of the child controls which belong to a common container control. It must be set to False if the program is to allow transparent text controls to overlap other text controls.

Syntax

[form.]*VisualWriter.***ClipSiblings** [= *boolean*]

Remarks

The ClipSiblings property settings are:

Setting	Description
True	Excludes those areas occupied by other child controls from the update area . (Default).
False	Updates areas which are occupied by other child controls.

Data Type

Boolean

See also

ClipChildren Property

Example

See Forms2 sample program.

ControlChars Property

Description

Specifies if control characters are visible.

Syntax

[form.]*VisualWriter.***ControlChars** [= *boolean*]

Remarks

The ControlChars Property settings are:

Setting	Description
True	Control characters, like space or paragraph break, are made visible.
False	Control characters are invisible.

Data Type

Boolean

CurrentPages Property

Description

Specifies the number of pages contained in VisualWriter. Not available at design time; read-only at run time.

Syntax

[form.]*VisualWriter.***CurrentPages**

Remarks

The value of this property depends on the size of the text as well as on the settings of the PageHeight, PageWidth and PageMargin*x* Properties.

Data Type

Long

See also

PageHeight Property

PageWidth Property

PageMarginB Property

PageMarginL Property

PageMarginR Property

PageMarginT Property

PrintDevice Property

PrintPage Method

Example

See PrintPage Method example.

DataChanged Property

Description

This is a standard Visual Basic property.

Indicates that data in the text control has been changed by some process other than getting data from the current record.

Syntax

[form.]*VisualWriter.***DataChanged** [= {*True*|*False*}]

Remarks

The DataChanged Property settings are:

Setting	Description
True	The data currently in the control is not the same as in the current record.
False	(Default) The data currently in the control, if any, is the same as the data in the current record.

Data Type

Integer (Boolean)

See also

Change event

DataField Property

Description

This is a standard Visual Basic property.

Binds a control to a Field.

Syntax

[form.]*VisualWriter.***DataField** [= *fieldname*]

Remarks

Bound controls provide access to specific data in your database. Each bound control typically displays the value of a different field in the current record.

Data Type

String

See also

DataSource property

DataFormat Property

Description

When using VisualWriter as a bound control, this property specifies if the data which is exchanged with a database is text or binary data.

Syntax

[form.]*VisualWriter.***DataFormat** [= *format*]

Setting	Description
0 - Text	Data is stored as text.
1 - Binary	Text and formatting information are stored in VisualWriter's own binary format.

Data Type

Integer

DataSource Property

Description

This is a standard Visual Basic property.

Determines the data control through which the current control is bound to a database.

Syntax

[form.]*VisualWriter.***DataSource**

Remarks

To bind a control to a field in a database at run time, you must specify a data control in this property at design time. To complete the connection with a database, you must also provide the name of a Field object in the DataField Property. Unlike the DataSource Property, the DataField Property setting can be provided at run time.

DataText Property

Description
Returns all of the text in the VisualWriter control.

Syntax
[form.]*VisualWriter.***DataText** [= *String*]

Data Type
String

See Also
DataFormat property

DblClick Event

Description

This is a standard Visual Basic event.

Occurs when the user presses and releases a mouse button and then presses and releases it again over an object.

Syntax

Private Sub Form_**DblClick** ( )

Private Sub *object*_**DblClick**(*index* As Integer)

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	Identifies the control if its in a control array.

See also

Click Event

MouseDown, MouseUp Events

DragDrop Event

Description

This is a standard Visual Basic event.

Occurs when a drag-and-drop operation is completed as a result of dragging a control over a form or control and releasing the mouse button or using the Drag method with its *action* argument set to 2 (Drop).

Syntax

Private Sub Form_**DragDrop** (*source* As Control, *x* As Single, *y* As Single)

Private Sub MDIForm_**DragDrop** (*source* As Control, *x* As Single, *y* As Single)

Private Sub *object*_**DragDrop** ([*index* As Integer,]*source* As Control, *x* As Single, *y* As Single)

The DragDrop syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if its in a control array.
source	The control being dragged. You can include properties and methods with this argumentfor example, Source.Visible = 0.
x,y	A number that specifies the current horizontal (x) and vertical (y) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the targets coordinate system as set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties.

Remarks

Use a DragDrop event procedure to control what happens after a drag operation is completed. For example, you can move the source control to a new location or copy a file from one location to another.

See also

DragIcon Property

DragMode Property

DragOver Event

MouseDown, MouseUp Events

MouseMove Event

DragIcon Property

Description

This is a standard Visual Basic property.

Determines the icon to be displayed as the pointer in a drag-and-drop operation.

Syntax

[form.]*VisualWriter.***DragIcon** [= *icon*]

Remarks

The DragIcon Property settings are:

Setting	Description
(*none*)	(Default) An arrow pointer inside a rectangle.
icon	A custom mouse pointer. You specify the icon by loading it using the Properties window at design time. You can also use the LoadPicture function at run time. The file you load must have the .ICO file-name extension and format.

Data Type

Integer

See also

DragDrop event

DragMode Property

DragMode Property

Description

This is a standard Visual Basic property.

Determines manual or automatic dragging mode for a drag-and-drop operation.

Syntax

[form.]*VisualWriter.***DragMode** [= *mode*]

Remarks

The DragMode property settings are:

Setting	Description
0	(Default) Manual; requires using the Drag method to initiate dragging on the source control.
1	Automatic; clicking the source control automatically initiates dragging.

Data Type

Integer

See also

DragDrop event

DragOver event

DragOver Event

Description

This is a standard Visual Basic event.

Occurs when a drag-and-drop operation is in progress. You can use this event to monitor the mouse pointer as it enters, leaves, or rests directly over a valid target. The mouse pointer position determines the target object that receives this event.

Syntax

Private Sub Form_**DragOver** (*source* As Control, *x* As Single, *y* As Single, *state* As Integer)

Private Sub MDIForm_**DragOver** (*source* As Control, *x* As Single, *y* As Single, *state* As Integer)

Private Sub *object*_**DragOver** ([*index* As Integer,]*source* As Control, *x* As Single, *y* As Single, *state* As Integer)

The DragOver event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if its in a control array.
source	The control being dragged. You can include properties and methods with this argument - for example, Source.Visible = False.
x,y	A number that specifies the current horizontal (x) and vertical (y) position of the mouse pointer within the target form or control. These coordinates are always expressed in terms of the targets coordinate system as set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties.
state	An integer that corresponds to the transition state of the control being dragged in relation to a target form or control: 0 = Enter (source control is being dragged within the range of a target). 1 = Leave (source control is being dragged out of the range of a target). 2 = Over (source control has moved from one position in the target to another).

Remarks

Use a DragDrop event procedure to determine what happens after dragging is initiated and before a control drops onto a target. For example, you can verify a valid targe range by highlighting the target (set the **BackColor** or **ForeColor** property from code) or by displaying a special drag pointer (set the **DragIcon** or **MousePointer** property from code).

Use the *state* argument to determine actions at key transition points. For example, you might highlight a possible target when *state* is set to 0 (Enter) and restore the objects previous appearance when *state* is set ot 1 (Leave).

When an object receives a DragOver event when *state* is set to 0 (Enter):

- If the source control is dropped on the object, that object receives a DragDrop event.
- If the source control isnt dropped on the object, that object receives another DragOver event when *state* is set to 1 (Leave).

See also

DragDrop Event

DragIcon Property

DragMode Property

MouseDown, MouseUp Events

MouseMove Event

EditMode Property

Description

Specifies whether VisualWriter operates in edit mode or in one of the two read-only modes.

Syntax

[form.]*VisualWriter.***EditMode** [= *mode*]

Remarks

The Property settings are:

Setting	Description
0	Edit mode. This mode can be used to edit and display text. The cursor is the text I-beam cursor.
1	Read-only mode. This mode can be used to display and select text. The cursor is the standard arrow cursor.
2	This mode can be used to display text only. Text input and selecting text with the mouse or the keyboard is not possible. The cursor is the standard arrow cursor.

Data Type

Integer

Enabled Property

Description

This is a standard Visual Basic property.

Determines whether the form or control can respond to user-generated events.

Syntax

[form.]*VisualWriter.***Enabled** [= *boolean*]

Remarks

The enabled property settings are:

Setting	Description
True	(Default) Allows the object to respond to events.
False	Prevents the object from responding to events.

Data Type

Integer (Boolean)

See also

Visible property

Error Event

Description

Occurs when VisualWriter reports an error.

Syntax

Sub VisualWriter1_**Error** (*Number* As Integer,*Description* As String, *Scode* As Error, *Source* As String, *HelpFile* As String, *HelpID* As Long, *CancelDisplay* As Boolean)

Remarks

The Error event has the following parameters:

Parameter	Description
Number	The error number.
Description	The error string.
SCode	OLE Status Code.
Source	Name of the module which caused the error.
HelpFile	Help file name.
HelpID	Help context ID.
CancelDisplay	Can be set to True if the application is to display the error string. If False, the control will display a message box showing the error string.

FieldChangeable Property

Description

Specifies if the contents of a marked text field can be changed by the user. The field number must have previously been determined with the FieldCurrent property. Not available at design time.

Syntax

[form.]*VisualWriter.***FieldChangeable** [= *boolean*]

Remarks

The FieldChangeable property settings are:

Setting	Description
True	The text which is contained in the field can be changed.
False	The text can not be changed.

Data Type

Boolean

See also

FieldDeleteable Property

FieldChanged Event

Description
Occurs when the text of a marked text field has been changed.

Syntax

Sub VisualWriter_**FieldChanged**(ByVal *Index* As Integer)

Remarks
The value of the FieldCurrentPropertyis updated with the field number of the marked text field for which the event has occured.

See also
FieldClicked Event

FieldCreated Event

FieldDblClicked Event

FieldDeleted Event

FieldSetCursor Event

FieldClicked Event

Description

Occurs when a marked text field has been clicked on.

Syntax

Sub VisualWriter_**FieldClicked**(ByVal *Index* As Integer)

Remarks

The value of the FieldCurrentPropertyis updated with the field number of the marked text field for which the event has occured.

See also

FieldChanged Event

FieldCreated Event

FieldDblClicked Event

FieldDeleted Event

FieldSetCursor Event

FieldCreated Event

Description

Occurs when a marked text field has been pasted from the clipboard.

Syntax

Sub VisualWriter_**FieldCreated**(ByVal *Index* As Integer)

Remarks

The value of the FieldCurrent Property is updated with the field number of the marked text field for which the event has occured.

See also

FieldChanged Event

FieldClicked Event

FieldDblClicked Event

FieldDeleted Event

FieldSetCursor Event

FieldCurrent Property

Description

Specifies the current marked text field for FieldChangeable, FieldDelete, FieldDeleteable, FieldPosX, FieldPosY, and FieldText. Not available at design time.

Syntax

[form.]*VisualWriter.***FieldCurrent** [= *field number*]

Data Type

Integer

Example

The example creates a marked text field with a text content of New Field and afterwards changes the text to Hello:

```
Sub Create()

       Dim FieldNumber As Integer

       'Create a marked text field and store its number

       VisualWriter.FieldInsert "New Field"

       FieldNumber = VisualWriter.FieldCurrent

       ..

       'Change the text

       VisualWriter.FieldCurrent = FieldNumber

       VisualWriter.FieldText = "Hello"

End Sub
```

FieldDblClicked Event

Description
Occurs when a marked text field has been double-clicked on.

Syntax

Sub VisualWriter_**FieldDblClicked**(ByVal *Index* As Integer)

Remarks
The value of the FieldCurrent Property is updated with the field number of the marked text field for which the event has occured.

See also

FieldChanged Event

FieldClicked Event

FieldCreated Event

FieldDeleted Event

FieldSetCursor Event

FieldDelete Method

Description

Deletes the marked text field specified by the FieldCurrent Property or changes it to simple text.

Syntax

[form.]*VisualWriter.***FieldDelete** [= *boolean*]

Remarks

The FieldDelete property settings are:

Setting	Description
True	The marked text field is completely deleted.
False	The marked text field is deleted, but its text contents is preserved.

Data Type

Boolean

FieldDeleteable Property

Description

Specifies whether a marked text field can be deleted by the user. The field number must have previously been determined with the FieldCurrent Property. Not available at design time.

Syntax

[form.]*VisualWriter.***FieldDeleteable** [= *boolean*]

Remarks

The FieldDeleteable property settings are:

Setting	Description
True	The field can be deleted.
False	The text can not be deleted.

Data Type

Boolean

See also

FieldChangeable Property

FieldDeleted Event

Description

Occurs when a marked text field has been deleted.

Syntax

Sub VisualWriter**_FieldDeleted**(ByVal *Index* As Integer)

Remarks

The value of the FieldCurrent Property is updated with the field number of the marked text field for which the event has occured.

See also

FieldChanged Event

FieldClicked Event

FieldCreatedEvent

FieldDblClicked Event

FieldSetCursor Event

FieldEnd Property

Description

Specifies the end position of a marked text field. The field number must have previously been determined with the FieldCurrent Property. Not available at design time; read-only at run time.

Syntax

[form.]*VisualWriter.***FieldEnd**

Data Type

Long

See also

FieldStart Property

FieldInsert Method

Description

Inserts a new marked text field at the current caret position.

Syntax

[form.]*VisualWriter.***FieldInsert** = *FieldText*

Remarks

Selected text can be converted to a marked text field by using an empty string as *FieldText*. Inserting a marked text field changes the value of the FieldCurrent Property to the number of the newly created field.

Data Type

String

Example

See description of the FieldCurrent Property

FieldPosX Property

Description

Specifies the position of a marked text field. The field number must have previously been determined with the FieldCurrent Property. Not available at design time; read-only at run time.

Syntax

[form.]*VisualWriter.***FieldPosX**

Remarks

The property value is the distance in horizontal direction between the left border of the marked text field and the left border of theVisualWriter. It is not affected by the scrollbar positions.

Data Type

Long

FieldPosY Property

Description

Specifies the position of a marked text field. The field number must have previously been determined with the FieldCurrent Property. Not available at design time; read-only at run time.

Syntax

[form.]*VisualWriter.***FieldPosY**

Remarks

The property value is the distance in vertical direction between the upper left corner of the marked text field and the upper left corner of the text. It is not affected by the scrollbar positions.

Data Type

Long

FieldSetCursor Event

Description
Occurs when the cursor is moved over a marked text field.

Syntax

Sub VisualWriter_**FieldSetCursor**(ByVal *Index* As Integer)

Remarks
This is the only field related event that does not change the value of the FieldCurrent Property.

See also
FieldChanged Event

FieldClicked Event

FieldCreated Event

FieldDblClicked Event

FieldDeleted Event

FieldStart Property

Description

Specifies the start position of a marked text field. The field number must have previously been determined with the FieldCurrent Property. Not available at design time; read-only at run time.

Syntax

[form.]*VisualWriter.***FieldStart**

Data Type

Long

See also

FieldEnd Property

FieldText Property

Description

Specifies the text which is contained within a marked text field. The field number must have previously been determined with the FieldCurrent Property. Not available at design time.

Syntax

[form.]*VisualWriter.***FieldText**

Data Type

String

FindReplace Method

Description
Displays a Find or Replace dialog box.

Syntax

[form.]*VisualWriter.***FindReplace** = *type of dialog*

Remarks
The property settings are:

Setting	Description
1	Display a Find dialog box.
2	Display a Replace dialog box.

Data Type
Integer

FontBold Property

Description

Determines the font style.

Syntax

[form.]*VisualWriter.***FontBold** [= *style*]

Remarks

At design time, this property works like the standard FontBold property. At runtime, if the FormatSelection property has been set to True, then the following settings affect only the selected text. The settings are:

Setting	Description
0	The characters are not bold.
1	The characters are bold.
2	Indicates that the selected text contains bold and non-bold characters.

Data Type

Integer

See also

FontItalic Property

FontStrikethru Property

FontUnderline Property

Font Dialog Method

FormatSelection Property

FontDialog Method

Description

Invokes VisualWriters built-in font dialog box and, after the user has closed the dialog box, specifies whether he has changed something.

Syntax

[form.]*VisualWriter.***FontDialog**

Remarks

The changes made in the dialog box apply to the currently selected text. The method returns one of the following values:

Return Value	Description
True	The user has changed one or more attributes.
False	The formatting remains unchanged.

Data Type

Boolean

FontItalic Property

Description
Determines the font style.

Syntax

[form.]*VisualWriter.***FontItalic** [= *style*]

Remarks
At design time, this property works like the standard FontItalic property. At runtime, if the FormatSelection property has been set to True, then the following settings affect only the selected text. The settings are:

Setting	Description
0	The characters are not italic.
1	The characters are italic.
2	Indicates that the selected text contains italic and non-italic characters.

Data Type
Integer

See also
FontBold Property

FontStrikethru Property

FontUnderline Property

FontDialog Method

FormatSelection Property

FontName Property

Description

This is a standard Visual Basic property.

Determines the font used to display text in a control.

Syntax

[form.]*VisualWriter.***FontName** [= *font*]

Remarks

The default for this property is determined by the system. Fonts available with Visual Basic vary according to your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which fonts exist.

In general, you should change FontName before setting size and style attributes with the FontSize, FontBold, FontItalic, FontStrikethru, FontTransparent, and FontUnderline properties.

Data Type

String

See also

FontItalic Property

FontStrikethru Property

FontSize Property

Description

This is a standard Visual Basic property.

Returns or sets the size of the font to be used for text displayed in a control.

Note The **FontSize** property is included for use with the **CommonDialog** control and for compability with earlier versions of Visual Basic. For additional functionality, use the new **Font** object properties (not available for the **CommonDialog** control).

Syntax

*object.***FontSize** [=*points*]

The **FontSize** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
points	A numeric expression specifying the font size to use, in points.

Remarks

Use this property to format text in the font size you want. The default is determined by the system. To change the default, specify the size of the font in points.

The maximum value for FontSize is 2160 points.

Note Fonts available with Visual Basic vary depending on your system configuration, display devices, and printing devices. Font-related properties can be set only to values for which fonts exist.

In general, you should change the **FontName** property before you set size and style attributes with the **FontSize**, **FontBold**, **FontItalic**, **FontStrikethru**, and **FontUnderline** properties. However, when you set TrueType fonts to smaller than 8 points, you should set the point size with the **FontSize** property, then set the **FontName** property, and then set the size again with the **FontSize** property. The Microsoft Windows operating environment uses a different font for TrueType fonts that are smaller than 8 points.

See also

FontBold Property

FontItalic Property

FontStrikethru Property

FontUnderline Property

FontCount Property

FontName Property

Fonts Property

Example

This example prints text on your form in two different point sizes with each click of the mouse. To try this example, paste the code into the Declarations section of a form. Press F5 to run the program, and then click on the form.

```
Private Sub Form_Click ( )

   FontSize = 24              Set FontSize.

   Print This is 24-point type.     Print large type.

   FontSize =  8              Set FontSize.

   Print This is 8-point type.      Print small font.

End Sub
```

FontStrikethru Property

Description

Determines the font style.

Syntax

[form.]*VisualWriter.***Strikethru** [= *style*]

Remarks

At design time, this property works like the standard FontStrikethru property. At runtime, if the FormatSelection property has been set to True, then the following settings affect only the selected text. The settings are:

Setting	Description
0	The characters are not struck through.
1	The characters are struck through.
2	Indicates that the selected text contains struck through and non-struck through characters.

Data Type

Integer

See also

FontBold Property

FontItalic Property

FontStrikethru Property

FontUnderline Property

FontDialog Method

FormatSelection Property

FontUnderline Property

Description

Determines the font style.

Syntax

[form.]*VisualWriter.***FontUnderline** [= *style*]

Remarks

At design time, this property works like the standard FontUnderline property. At runtime, if the FormatSelection property has been set to True, then the following settings affect only the selected text. The settings are:

Setting	Description
0	The characters are not underlined.
1	The characters are underlined.
2	Indicates that the selected text contains underlined and non-underlined characters.

Data Type

Integer

See also

FontBold Property

FontItalic Property

FontStrikethru Property

FontDialog Method

FormatSelection Property

ForeColor Property

Description

This is a standard Visual Basic property.

Determines the foreground color of an object.

Syntax

[form.]*VisualWriter.***ForeColor** [= *color*]

Remarks

Visual Basic uses the Microsoft Windows environment RGB scheme for colors. Each property has the following range of settings:

Range of Settings	Description
Normal RGB colors	Colors specified by using the color palette, or by using the RGB or QBColor functions in code.
System default colors	Colors specified with system color constants from CONSTANT.TXT, a Visual Basic file that specifies system defaults. The Windows environment substitutes the users choices as specified in the users Control Panel settings.

Data Type

Long

See also

BackColor Property

FormatSelection Property

Description

Specifies if character and paragraph formatting properties apply to the whole text or to a particular selection only.

Syntax

[form.]*VisualWriter.***FormatSelection**

Remarks

The properties which are affected are Alignment, BaseLine, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, LineSpacing, LineSpacingT, ForeColor, and TextBkColor.

Setting	Description
True	The formatting properties only apply to selected text. This mode works only at run time, because at design time it is not possible to select text.
False	The formatting properties apply to the whole text. This is the default mode.

Data Type

Boolean

FrameDistance Property

Description

Specifies the distance between text and paragraph frame for the currently selected paragraph(s). Not available at design time.

Syntax

[form.]*VisualWriter.***FrameDistance** [= *distance*]

Remarks

The property value is set to -1 if the user selects two or more paragraphs which have different frame distance settings.

Data Type

Integer

FrameLineWidth Property

Description

Specifies the line widths of the currently selected paragraph's frames. Not available at design time.

Syntax

[form.]*VisualWriter.***FrameLineWidth** [= *line width*]

Remarks

The property value is set to 0 if the user selects two or more paragraphs which have different line width settings.

Data Type

Integer

FrameStyle Property

Description

Specifies the style of the currently selected paragraph's frames. Not available at design time.

Syntax

[form.]*VisualWriter.***FrameStyle** [= *Style Flags*]

Remarks

The property value can be a combination of the following flags:

Setting	Description
BF_LEFTLINE (&H1)	Draws a left frame line.
BF_RIGHTLINE (&H2)	Draws a right frame line.
BF_TOPLINE (&H4)	Draws a top frame line.
BF_BOTTOMLINE (&H8)	Draws a bottom frame line.
BF_TABLINES (&H10)	Draws a vertical line at each tab position.
BF_SINGLE (&H20)	Draws a single line.
BF_DOUBLE (&H40)	Draws a double line.
BF_BOXCONNECT (&H80)	Draws a double line.
BF_NOLEFTLINE (&H100)	Resets an existing left line.
BF_NORIGHTLINE (&H200)	Resets an existing right line.
BF_NOTOPLINE (&H400)	Resets an existing top line.
BF_NOBOTTOMLINE (&H800)	Resets an existing bottom line.
BF_NOTABLINES (&H1000)	Resets existing tabulator lines.

The property value is set to -1 if the user selects two or more paragraphs which have different frame style settings.

Data Type

Integer

GotFocus Event

Description

This is a standard Visual Basic event.

Occurs when an object receives the focus, either by user action, such as tabbing to or clicking the object, or by changing the focus in code using the **SetFocus** method. A form receives the focus only when all visible controls are disabled.

Syntax

Private Sub Form_**GotFocus** ( )

Private Sub *object*_**GotFocus** ([*index* As Integer])

The GotFocus event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if its in a control array.

Remarks

Typically, you use a GotFocus event procedure to specify the actions that occur when a control or form first receives the focus. For example, by attaching a GotFocus event procedure to each control on a form, you can guide the user by displaying brief instructions or status bar messages. You can also provide visual cues by enabling, disabling, or showing other controls that depend on the control that has the focus.

Note An object can receive the focus only if its Enabled and Visible properties are set to True. To customize the kekyboard interface in Visual Basic for moving the focus, set the tab order or specify access keys for controls on a form.

See also

ActiveControl Property

ActiveForm Property

LostFocus Event

SetFocus Method

TabIndex Property

TabStop Property

Height Property

Description

This is a standard Visual Basic property.

Determines the dimensions of an object.

Syntax

[form.]*VisualWriter.***Height** [= *numericexpression*]

Remarks

Measurements are calculated as follows:

- **Form** the external height and width of the form, including the borders and title bar.
- **Control** measured from the center of the controls border so that controls with different border widths align correctly. These properties use the scale units of a controls container.

For a form or control, the values for these properties change as the object is sized by the user or by code. Maximum limits of these properties for all objects are system-dependent.

Data Type

Single

See also

Width Property

Left Property

Top Property

Move Method

HelpContextID Property

Description

This is a standard Visual Basic property.

Determines an associated context number for an object.

Syntax

[form.]*VisualWriter.***HelpContextID** [= *setting*]

Remarks

For context-sensitive Help on an object in your application, you must assign the same context number to both the object and to the associated Help topic when you compile your Help file.

If youve created a Windows environment Help file for your application and set the applications HelpFile property, when a user presses the F1 key, Visual Basic automatically calls Help and requests the topic identified by the current context number.

The current context number is the value of HelpContextID for the object that has the focus. If HelpContextID is 0, then Visual Basic looks in the HelpContextID of the objects container, and then that objects container, and so on. If a nonzero current context number cant be found, the F1 key is ignored.

Data Type

Long integer

HideSelection Property

Description
Specifies whether a text selection is to be hidden when the VisualWriter window is not active.

Syntax
[form.]*VisualWriter.***HideSelection** [= *boolean*]

Remarks
The HideSelection property settings are:

Setting	Description
True	The selection is hidden when the VisualWriter window becomes inactive.
False	The selection stays visible.

Data Type
Boolean

HScroll Event

Description

Occurs when the horizontal scroll position has been changed.

Syntax

Sub VisualWriter_**HScroll**()

See also

VScroll Event

hWnd Property

Description

This is a standard Visual Basic property.

Specifies the handle to a form or control.

Syntax

[form.]*VisualWriter.***hWnd**

Remarks

The Windows environment identifies each form and control in an application by assigning it a handle, or hWnd. The hWnd is used with Windows API calls. Many Windows environment functions require the hWnd of the current window as an argument.

Data Type

Integer

IndentB Property

Description
Determine the bottom indent (in twips) for a paragraph or a selected range of paragraphs.

Syntax

[form.]*VisualWriter.***IndentB** [= *indent*]

Remarks
If a number of paragraphs have been selected which have different settings for one of the indents, the value of this indent is INDENT_NOCOMMON (&H8000). The first line indent can be negative.

Data Type
Integer

IndentFL Property

Description
Determine the first line indent (in twips) for a paragraph or a selected range of paragraphs.

Syntax

[form.]*VisualWriter.***IndentFL** [= *indent*]

Remarks
If a number of paragraphs have been selected which have different settings for one of the indents, the value of this indent is INDENT_NOCOMMON (&H8000). The first line indent can be negative.

Data Type
Integer

IndentL Property

Description
Determine the left indent (in twips) for a paragraph or a selected range of paragraphs.

Syntax
[form.]*VisualWriter.***IndentL** [= *indent*]

Remarks
If a number of paragraphs have been selected which have different settings for one of the indents, the value of this indent is INDENT_NOCOMMON (&H8000). The first line indent can be negative.

Data Type
Integer

IndentR Property

Description
Determine the right indent (in twips) for a paragraph or a selected range of paragraphs.

Syntax

[form.]*VisualWriter.***IndentR** [= *indent*]

Remarks
If a number of paragraphs have been selected which have different settings for one of the indents, the value of this indent is INDENT_NOCOMMON (&H8000). The first line indent can be negative.

Data Type
Integer

IndentT Property

Description

Determine the top indent (in twips) for a paragraph or a selected range of paragraphs.

Syntax

[form.]*VisualWriter.***IndentT** [= *indent*]

Remarks

If a number of paragraphs have been selected which have different settings for one of the indents, the value of this indent is INDENT_NOCOMMON (&H8000). The first line indent can be negative.

Data Type

Integer

Index Property

Description

This is a standard Visual Basic property.

Specifies the number that uniquely identifies a control in a control array.

Syntax

[form.]**VisualWriter**[(***integer***)] **.Index**

Remarks

The Index Property settings are:

Setting	Description
No value	(Default) Not part of an array.
0 to 32,767	Part of an array. Specify an integer greater than or equal to 0 to assign a control to a control array. You can specify the same name for two or more controls through the Name property. Visual Basic automatically assigns the first unique index available within the control array.

Data Type

Integer

See also

Tag Property

InsertionMode Property

Description
Specifies insert or overwrite mode.

Syntax
[form.]*VisualWriter.***InsertionMode** [= *boolean*]

Remarks
The InsertionMode settings are:

Setting	Description
True	Insert mode.
False	Overwrite mode.

Data Type
Boolean

KeyDown, KeyUp Events

Description

These are standard Visual Basic properties.

Occur when the user presses (KeyDown) or releases (KeyUp) a key while an object has the focus. (To interpret ANSI characters, use the KeyPress event.)

Syntax

Private Sub *Form*_**KeyDown** (*keycode* As Integer, *shift* As Integer)

Private Sub *object*_**KeyDown** ([*index* As Integer,]*keycode* As Integer, *shift* As Integer)

Private Sub *Form*_**KeyUp** (*keycode* As Integer, *shift* As Integer)

Private Sub *object*_**KeyUp** ([*index* As Integer,]*keycode* As Integer, *shift* As Integer)

The KeyDown and KeyUp event syntaxes have these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if its in a control array.
keycode	A key code, such as **vbKeyF1** (the F1 key) or **vbKeyHome** (the HOME key). To specify key codes, use the constants in the Visual Basic object library in the Object Browser.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT are pressed, the value of *shift* is 6.

Remarks

For both events, the object with the focus receives all keystrokes. A form can have a focus only if it has no visible and enabled controls. Although the KeyDown and KeyUp events can apply to most keys, they are most often used for:

- Extended character keys such as function keys.
- Navigation keys.
- Combinations of keys with standard keyboard modifiers.
- Distinguishing between the numeric keypad and regular number keys.

Use the KeyDown and KeyUp event procedures if you need to respond to both the pressing and releasing of a key.

KeyDown and KeyUp arent invoked for:

- The ENTER key if the form has a CommandButton control with the Default property set to True.
- The ESC key if the form has a CommandButton control with the Cancel property set to True.
- The TAB key.

KeyDown and KeyUp interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key) and *shift*, which indicates the state of *shift+key* and therefore returns either A or a.

If you need to test for the shift argument, you can declare constants that define the bits within the argument by using constants listed in the Visual Basic object library in the Object Browser. The shift constants have the following values:

Constant	Value	Description
vbShiftMask	1	SHIFT key bit mask.
vbCtrlMask	2	CTRL key bit mask.
vbAltMask	4	ALT key bit mask.

The constants act as bit masks that you can use to test for any combination of keys. Place the constants at the procedure level or in the Delcarations section of a module and use this syntax:

Const Constantname = expression

You test for a condition by first assigning each result to a temporary integer variable and then comparing shift to a bit mask. Use the And operator with the shift argument to test whether the condition is greater than 0, indicating that the modifier was pressed, as in this Example

```
ShiftDown = (Shift And vbShiftMask) > 0
```

In a procedure, you can test for any combination of conditions, as in this Example

```
If ShiftDown And CtrlDown Then
```

Note If the **KeyPreview** property is set to **True**, a form receives these events before controls on the form receive the events. Use the **KeyPreview** property to create global keyboard-handling routines.

KeyPress Event

Description

This is a standard Visual Basic property.

Occurs when the user presses and releases an ANSI key.

Syntax

Private Sub **Form_KeyPress** (keyascii As Integer)

Private Sub ***object*_KeyPress** ([*index* As Integer,]*keyascii* As Integer)

The KeyPress event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if its in a control array.
keyascii	An integer that returns a standard numeric ANSI keycode. *Keyascii* is passed by reference; changing it sends a different character to the object. Changing *keyascii* to 0 cancels the keystroke so the object receives no character.

Remarks

The object with the focus receives the event. A form can receive the event only if it has no visible and enabled controls or if the **KeyPreview** property is set to **True**. A KeyPress event can involve any printable keyboard character, the CTRL key combined with a character from the standard alphabet or one of a few special characters, and the ENTER or BACKSPACE key. A KeyPress event procedure is useful for intercepting keystrokes entered in a **TextBox** or **ComboBox** control. It enables you to immediately test keystrokes for validity or to format characters as theyre typed. Changing the value of the *keyascii* argument changes the character displayed.

You can convert the *keyascii* argument into a character by using the expression:

```
Chr (KeyAscii)
```

You can then perform string operations and translate the character back to an ANSI number that the control can interpret by using the expression:

```
KeyAscii = Asc (char)
```

Use KeyDown and KeyUp event procedures to handle any keystroke not recognized by KeyPress, such as function keys, editing keys, navigation keys, and any combinations of these with keyboard modifiers. Unlike the KeyDown and KeyUp events, KeyPress doesnt indicate the physical state of the keyboard; instead, it passes a character.

KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters. KeyDown and KeyUp interpret the uppercase and lowercase of each character by means of two arguments: *keycode*, which indicates the physical key (thus returning A and a as the same key), and *shift*, which indicates the state of *shift*+*key* and therefore returns either A or a.

If the **KeyPreview** property is set to **True**, a form receives the event before controls on the form receive the event. Use the **KeyPreview** property to create global keyboard-handling routines.

Note The ANSI number for the keyboard combination of CTRL+@ is 0. Because Visual Basic recognizes a keyascii value of 0 as a zero-length string (), avoid using CTRL+@ in your applications.

See also

Asc Function	Change Event
Character Set (0-127)	Character Set (128-255)
Chr Function	KeyDown, KeyUp Events
KeyPreview Property	SendKeys Statement

Example

This example converts text entered into a **TextBox** control to uppercase. To try this example, paste the code into the Declarations section of a form that contains a **TextBox**, and then press F5 and enter something into the **TextBox**.

```
Private Sub Text1_KeyPress (KeyAscii As Integer)

   Char = Chr (KeyAscii)

   KeyAscii = Asc (Ucase (Char))

End Sub
```

Language Property

Description

Determines the language in which VisualWriter displays dialog boxes and error messages. Not available at design time.

Syntax

[form.]*VisualWriter.***Language** = *Country code*

Remarks

The default language is determined by the 'iCountry=' setting in WIN.INI.

Setting	Description
34	Spanish
49, 41, 43	German
else	English

Data Type

Boolean

Left Property

Description

This is a standard Visual Basic property.

Determines the distance between the internal left edge of an object and the left edge of its container.

Syntax

[form.]*VisualWriter.***Left** [= *x*]

Remarks

You can specify a single-precision number. Use Left and Top properties and the Height and Width properties for operations based on an objects external dimensions, such as moving or resizing.

Data Type

Single

See also

Top Property

Move Method

LineSpacing

Description
Specifies the line spacing for the currently selected paragraphs as a percentage of the font size.

Syntax

[form.]*VisualWriter.***LineSpacing** [= *line spacing*]

Data Type
Integer

LineSpacingT Property

Description
Specifies the line spacing for the currently selected paragraphs in twips.

Syntax

[form.]*VisualWriter.***LineSpacingT** [= *line spacing*]

Data Type
Integer

Load Method

Description

Loads the contents of a text control with all text and format information from a file which has previously been saved using the Save method.

Syntax

[form.]*VisualWriter.***Load FileName**, *offset*

Remarks

You can store more than one text controls data in a single file. The *offset* parameter determines the file position from where the text control's data is read.

The Load method will only load previously saved VisualWriter files. To read an ASCII text file from another word processor, see TextImport Method. To read an RTF file, see RTFImport Method.

Data Type

FileName: String

Offset: Long

See also

Save Method

Example

This example opens a file and loads its contents into VisualWriter1:

```
Private Sub mnuFile_Load_Click()

       On Error Resume Next

       ' Create an "Open File" dialog box

       CommonDialog1.Filter = "TX Demo (*.tx)|*.tx"

       CommonDialog1.DialogTitle = "Open"

       CommonDialog1.Flags = cdlOFNFileMustExist Or _
                 cdlOFNHideReadOnly

       CommonDialog1.CancelError = True

       CommonDialog1.ShowOpen

       If Err Then Exit Sub

       ' Pass the filename to the text control

       VisualWriter1.Load CommonDialog1.filename, 0

End Sub
```

LostFocus Event

Description

This is a standard Visual Basic event.

Occurs when an object loses the focus, either by user action, such as tabbing to or clicking another object, or by changing the focus in code using the **SetFocus** method.

Syntax

Private Sub Form_LostFocus ( )

Private Sub *object*_**LostFocus** ([*index* As Integer])

The LostFocus event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if its in a control array.

Remarks

A LostFocus event procedure is primarily useful for verification and validation updates. Using LostFocus can cause validation to take place as the user moves the focus from the control. Another use for this type of event procedure is enabling, disabling, hiding, and displaying other objects as in a GotFocus event procedure. You can also reverse or change conditions that you set up in the objects GotFocus event procedure.

See also

ActiveControl Property

ActiveForm Property

GotFocus Event

SetFocus Method

TabIndex Property

TabStop Property

Example

This example changes the color of a **TextBox** control when it receives or loses the focus (selected with the mouse or TAB key) and displays the appropriate text in the **Label** control. To try this example, paste the code into the Declarations section of a form that contains two **TextBox** controls and a **Label** control, and then press F5 and move the focus between Text1 and Text2.

```
Private Sub Text1_GotFocus ( )

   Show focus with red.

   Text1.BackColor = RGB (255, 0, 0)

   Label1.Caption = Text1 has the focus.

End Sub
```

```
Private Sub Text1_LostFocus ( )

   Show loss of focus with blue.

   Text1.BackColor = RGB (0, 0, 255)

   Label1.Caption = Text doesnt have the focus.

End Sub
```

MouseDown, MouseUp Events

Description

These are standard Visual Basic properties.

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button.

Syntax

Private Sub *Form*_**MouseDown** (*button* As Integer, *shift* As Integer, *x* As Single, *y* As Single)

Private Sub *MDIForm*_**MouseDown** (*button* As Integer, *shift* As Integer, *x* As Single, *y* As Single)

Private Sub *object*_**MouseDown** ([*index* As Integer,]*button* As Integer, *shift* As Integer, *x* As Single, *y* As Single)

Private Sub *Form*_**MouseUp** (*button* As Integer, *shift* As Integer, *x* As Single, *y* As Single)

Private Sub *MDIForm*_**MouseUp** (*button* As Integer, *shift* As Integer, *x* As Single, *y* As Single)

Private Sub *object*_**MouseUp** ([*index* As Integer,]*button* As Integer, *shift* As Integer, *x* As Single, *y* As Single)

The MouseDown and MouseUp event syntaxes have these parts:

Part	Description
object	Returns an object expression that evaluates to an object in the Applies To list.
index	Returns an integer that uniquely identifies a control if its in a control array.
button	Returns an integer that identifies the button that was pressed (MouseDown) or released (MouseUp) to cause the event. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. Only one of the bits is set, indicating the button that caused the event.
shift	Returns an integer that corresponds to the state of the SHIFT, CTRL, and ALT keys when the button specified in the *button* argument is pressed or released. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were presssed, the value of *shift* would be 6.
x, y	Returns a number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Remarks

Use a MouseDown or MouseUp event procedure to specify actions that will occur when a given mouse button is pressed or released. Unlike the Click and DblClick events, MouseDown and MouseUp events enable you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keyboard modifiers.

The following applies to both Click and DblClick events:

- If a mouse button is pressed while the pointer is over a form or control, that object captures the mouse and receives all mouse event up to and including the last MouseUp event. This implies tha the x, y mouse-pointer coordinates returned by a mouse event may not always be in the internal area of the object that receives them.
- If mouse buttons are pressed in succession, the object that captures the mouse after the first press receives all mouse events until all buttons are released.

If you need to test for the button or shift arguments, you can use constants listed in the Visual Basic object library in the Object Browser to define the bits within the argument:

Constant (Button)	Value	Description
vbLeftButton	1	Left button is pressed.
vbRightButton	2	Right button is pressed.
vbMiddleButton	4	Middle button is pressed.

Constant (Button)	Value	Description
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is passed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field value for each combination.

Note You can use a MouseMove event procedure to respond to an event caused by moving the mouse. The button argument for MouseDown and MouseUp differs from the button argument used for MouseMove. For MouseDown and MouseUp, the button argument indicates exactly one button per event; for MouseMove, it indicates the current state of all buttons.

See also

Click Event

DblClick Event

MouseMove Event

MousePointer Property

MouseMove Event

Description

This is a standard Visual Basic event.

Occurs when the user moves the mouse.

Syntax

Private Sub *Form*_**MouseMove** (*button* As Integer, *shift* As Integer, *x* As Single, *y* As Single)

Private Sub *MDIForm*_**MouseMove** (*button* As Integer, *shift* As Integer, *x* As Single, *y* As Single)

Private Sub *object*_**MouseMove** ([*index* As Integer,]*button* As Integer, *shift* As Integer, *x* As Single, *y* As Single)

The MouseMove event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
index	An integer that uniquely identifies a control if its in a control array.
button	An integer that corresponds to the state of the mouse buttons in which a bit is set if the button is down. The *button* argument is a bit field with bits corresponding to the left button (bit 0), right button (bit 1), and middle button (bit 2). These bits correspond to the values 1, 2, and 4, respectively. It indicates the complete state of the mouse buttons; some, all, or none of these three bits can be set, indicating that some, all, or none of the buttons are pressed.
shift	An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys. A bit is set if the key is down. The *shift* argument is a bit field with the least-significant bits corresponding to the SHIFT key (bit 0), the CTRL key (bit 1), and the ALT key (bit 2). These bits correspond to the values 1, 2, and 4, respectively. The *shift* argument indicates the state of these keys. Some, all, or none of the bits can be set, indicating that some, all, or none of the keys are pressed. For example, if both CTRL and ALT were presssed, the value of *shift* would be 6.
x, y	A number that specifies the current location of the mouse pointer. The *x* and *y* values are always expressed in terms of the coordinate system set by the **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties of the object.

Remarks

The MouseMove event is generated continually as the mouse pointer moves across objects. Unless another object has captured the mouse, an object recognizes a MouseMove event whenever the mouse position is within its borders.

If you need to test for the *button* or *shift* arguments, you can use constants listed in the Visual Basic object library in the Object Browser to define the bits within the argument:

Constant (Button)	Value	Description
vbLeftButton	1	Left button is pressed.
vbRightButton	2	Right button is pressed.
vbMiddleButton	4	Middle button is pressed.

Constant (Button)	Value	Description
vbShiftMask	1	SHIFT key is pressed.
vbCtrlMask	2	CTRL key is pressed.
vbAltMask	4	ALT key is pressed.

The constants then act as bit masks you can use to test for any combination of buttons without having to figure out the unique bit field values for each combination.

You test for a condition by first assigning each result to a temporary integer variable and then comparing the *button* or *shift* arguments to a bit mask. Use the And operator with each argument to test if the condition is greater than zero, indicating the key or button is pressed, as in this Example

```
LeftDown = (Button And vbLeftButton) > 0

CtrlDown = (Shift And vbCtrlMask) > 0
```

Then, in a procedure, you can test for any combination of conditions, as in this Example

```
If LeftDown And CtrlDown Then
```

Note You can use MouseDown and MouseUp event procedures to respond to events caused by pressing and releasing mouse buttons.

The button argument for MouseMove differs from the button argument for MouseDown and MouseUp. For MouseMove, the button argument indicates the current state of all buttons; a single MouseMove can indicate that some, all or no buttons are pressed. For Mousedown and MouseUp, the button argument indicates exactly one button per event.

Any time you move a window inside a MouseMove event, it can cause a cascading event. MouseMove events are generated when the window moves underneath the pointer. A MouseMove event can be generated even if the mouse is perfectly stationary.

See also

Click Event

DblClick Event

MouseDown, MouseUp Events

MousePointer Property

MousePointer Property

Description

This is a standard Visual Basic property.

Determines the type of mouse pointer displayed when the mouse is over a particular part of a form or control at run time.

Syntax

[form.]*VisualWriter.***MousePointer** [= *setting*]

Remarks

The MousePointer Property settings are:

Setting	Description
0	(Default) Shape determined by control
1	Arrow
2	Cross (cross-hair pointer)
3	I-Beam
4	Icon (small square within a square)
5	Size (four-pointed arrow pointing north, south, east, west)
6	Size NE SW (double arrow pointing northeast and southwest)
7	Size N S (double arrow pointing north and south)
8	Size NW SE (double arrow pointing northwest and southeast)
9	Size W E (double arrow pointing west and east)
10	Up Arrow
11	Hourglass (wait)

The MousePointer property controls the shape of the mouse pointer. This property is useful when you want to indicate changes in functionality as the mouse pointer passes over controls on a form or dialog box. The hourglass setting (11) is useful for indicating that the user should wait for a process or operation to finish.

Data Type

Integer

See also

MouseMove event

Move Event

Description
Occurs when VisualWriter has been moved with the mouse while depressing the ALT key.

Syntax
Sub VisualWriter_**Move**

See also
Size Event

SizeMode Property

Move Method

Description

This is a standard Visual Basic method.

Moves an **MDIForm**, **Form**, or control. Doesnt support named arguments.

Syntax

*VisualWriter.***Move** *left, top, width, height*

The Move method syntax has these parts:

Part	Description
object	Optional. An object expression that evaluates to an object in the Applies To list. If *object* is omitted, the form with the focus is assumed to be *object*.
left	Required. Single-precision value indicating the horizontal coordinate (x-axis) for the left edge of *object*.
top	Optional. Single-precision value indicating the vertical coordinate (y-axis) for the top edge of *object*.
width	Optional. Single-precision value indicating the new width of *object*.
height	Optional. Single-precision value indicating the new height of *object*.

Remarks

Only the *left* argument is required. However, to specify any other arguments, you must specify all arguments that appear in the syntax before the argument you want to specify. For example, you cant specify *width* without specifying *left* and *top*. Any trailing arguments that are unspecified remain unchanged.

For forms and controls in a **Frame** control, the coordinate system is always in twips. Moving a form on the screen or moving a control in a **Frame** is always relative to the origin (0,0), which is the upper-left corner. When moving a control on a **Form** object or in a **PictureBox** (or an MDI child form on an **MDIForm** object), the coordinate system of the container object is used. The coordinate system or unit of measure is set with the **ScaleMode** property at design time. You can change the coordinate system at run time with the **Scale** method.

See also

Height Property

Left Property

ScaleHeight Property

ScaleLeft Property

ScaleMode Property

ScaleWidth Property

Top Property

Width Property

Name Property

Description

This is a standard Visual Basic property.

Specifies the name used in code to identify a form, control, or data access object.

Syntax

[form.]*VisualWriter*.**Name**

Remarks

An objects Name Property must start with a letter and can be a maximum of 40 characters. It can include numbers and underscore characters but cant include punctuation or spaces. Forms cant have the same name as another global object such as Clipboard, Screen or App. However, a name can be the same as a reserved word, property name, or the name of another object, but this can create conflicts in your code. Not available at run time.

Data Type

String

See also

Index property

Text property

NextWindow Property

Description

Indicates the next window in that will get focus.

Syntax

[form.]*VisualWriter.***NextWindow** [= *OLE_HANDLE*]

See Also

ObjectNext method

ObjectClicked Event

Description
Occurs when an object has been clicked on.

Syntax

> Sub VisualWriter_**ObjectClicked**(ByVal *ObjectIndex* As Integer)

See also
ObjectInsertAsChar Method

ObjectInsertFixed Method

ObjectCreated Event

Description
Occurs when an object is inserted.

Syntax

Sub VisualWriter_**ObjectCreated** (ByVal *ObjectIndex* As Integer)

See Also
ObjectInsertAsChar property

ObjectCurrent Property

Description

Specifies the current object for the properties: ObjectAttr and ObjectFilename. The value is automatically updated when an object is inserted or when you click on an object. Not available at design time.

Syntax

[form.]*VisualWriter.***ObjectCurrent** [= *object number*]

Data Type

Integer

ObjectDblClicked Event

Description

Occurs when an object has been double-clicked on. For more information, see VisualWriter Objects.

Syntax

Sub VisualWriter_**ObjectDblClicked**(ByVal *ObjectIndex* As Integer)

See also

ObjectInsertAsChar Method

ObjectInsertFixed Method

ObjectDelete Method

Description

This method deletes an object.

Syntax

[form.]*VisualWriter.***ObjectDelete** *ObjectNumber*

Data Type

Integer

See also

ObjectInsertAsChar Method

ObjectInsertFixed Method

ObjectDeleted Event

Description
Occurs when an object has been deleted

Syntax

Sub VisualWriter_**ObjectDeleted** (ByVal *ObjectIndex* As Integer)

See also
ObjectInsertAsChar Method

ObjectInsertFixed Method

ObjectDistance Property

Description

Specifies the distance (in twips) between an object and the text that flows around it.

Syntax

[form.]*VisualWriter.***ObjectDistance**(i) [= *distance*]

Remarks

This property can only be used with objects which have been inserted using the ObjectInsertFixed method. Otherwise an Error event is generated. The property array's members are:

Setting	Description
ObjectDistance(1)	Left distance.
ObjectDistance(2)	Top distance.
ObjectDistance(3)	Right distance.
ObjectDistance(4)	Bottom distance.

Data Type

Array of 4 integer values

See also

ObjectInsertAsChar Method

ObjectInsertFixed Method

ObjectInsertAsChar Method

Description

This method embeds a new object or image which is handled like a single character in the text.

Syntax

[form.]*VisualWriter.***ObjectInsertAsChar** *hWnd, FileName, TextPos, ScaleX, ScaleY, ImageDisplayMode, ImageSaveMode*

Remarks

The methods parameters are:

Parameter	Data Type	Description
hWnd	Integer	Specifies an externally created window that represents the object to be inserted. This parameter can be zero if the FileName parameter specifies a file name containing an image. In this case VisualWriter creates an Image-Control window and handles this window internally.
FileName	String	Specifies the full DOS path name of a file that contains an image. This parameter can be zero if *hWnd* specifies an externally created window.
TextPos	Long	This parameter specifies the text position where the object is to be inserted. If *lTextPos* is -1 the object is inserted at the current input position.
ScaleX	Integer	Specifies a horizontal scaling factor as a percentage. It must be a value between 10 and 250.
ScaleY	Integer	Specifies a vertical scaling factor as a percentage. It must be a value between 10 and 250.
DisplayMode	Integer	Specifies the display status of the object or image.
SaveMode	Integer	Specifies the saved format of the object or image.

See also

ObjectInsertFixed Method

ObjectInsertFixed Method

Description

This method embeds a new object or image at a fixed position. The text flows around the object.

Syntax

[form.]*VisualWriter.***ObjectInsertFixed** *hWnd, FileName, PosX, PosY, ScaleX, ScaleY, ImageDisplayMode, ImageSaveMode, SizeMode, TextFlow, DistanceL, DistanceT, DistanceR, DistanceB*

Remarks

The methods parameters are:

Parameter	Data Type	Description
hWnd	Integer	Specifies an externally created window that represents the object to be inserted. This parameter can be zero if the FileName parameter specifies a file name containing an image. In this case VisualWriter creates an Image-Control window and handles this window internally.
FileName	String	Specifies the full DOS path name of a file that contains an image. This parameter can be zero if *hWnd* specifies an externally created window.
PosX	Long	Specifies the objects horizontal position in twentieths of a point.
PosY	Long	Specifies the objects vertical position in twentieths of a point.
ScaleX	Integer	Specifies a horizontal scaling factor as a percentage. It must be a value between 10 and 250.
ScaleY	Integer	Specifies a vertical scaling factor as a percentage. It must be a value between 10 and 250.
DisplayMode	Integer	Specifies the display status of the object or image.
SaveMode	Integer	Specifies the saved format of the object or image.
SizeMode	Integer	See ObjectSizeModeProperty.
TextFlow	Integer	See ObjectTextFlowProperty.
DistanceL	Integer	See ObjectDistanceProperty.
DistanceT	Integer	See ObjectDistanceProperty.
DistanceR,	Integer	See ObjectDistanceProperty.

DistanceB	Integer	See ObjectDistanceProperty.

See also

ObjectInsertAsChar Method

ObjectMoved Event

Description
Occurs when an embedded object has been moved with the mouse while depressing the ALT key.

Syntax

Sub VisualWriter_**ObjectMoved**(ByVal *ObjectIndex* As Integer)

See also
ObjectSized Event

ObjectNext Method

Description

Moves focus to the next object in the text control.

Syntax

[form.]*VisualWriter.***ObjectNext** (*ObjectNumber* As Integer, *ObjectGroup* As Integer) As Integer

Remarks

ObjectNumber indicates the number of total objects and will return the value equal to *ObjectNumber* +1. *ObjectGroup* can have a value of 0 if you do not create an object group.

Data Type

Integer

See Also

NextWindow property

ObjectScaleX Property

Description

Specifies the object's scaling factor as a percentage in the range of 10 to 400%. The object must have previously been selected with the ObjectCurrent property.

Syntax

[form.]*VisualWriter.***ObjectScaleX** [= *factor*]

Data Type

Integer

See also

ObjectDistance Property

ObjectInsertFixed Method

ObjectScaleY Property

ObjectScaleY Property

Description

Specifies the object's scaling factor as a percentage in the range of 10 to 400%. The object must have previously been selected with the ObjectCurrent property.

Syntax

[form.]*VisualWriter.***ObjectScaleY** [= *factor*]

Data Type

Integer

See also

ObjectDistance Property

ObjectInsertFixed Method

ObjectScaleX Property

ObjectSized Event

Description
Occurs when an embedded object has been resized with the mouse while depressing the ALT key.

Syntax
Sub VisualWriter_**ObjectSized**()

See also
ObjectMoved Event

ObjectSizeMode Property

Description

Specifies whether an inserted object can be moved or resized at runtime. If the Moveable option is selected, the control can be moved on the background by depressing the ALT key and then dragging the control with the mouse. If the Sizeable option is selected and the ALT key is depressed, the borders of the control can be dragged.

Syntax

[form.]*VisualWriter.***ObjectSizeMode** = *mode*

Remarks

The property settings are:

Setting	Description
0 - Fixed	The object cannot be moved or resized. (Default).
1 - Moveable	The object window can be moved.
2 - Sizeable	The object window can be resized.
3 - Move andSizeable	Both 1 and 2.

Data Type

Integer

See also

ObjectInsertFixed Method

ObjectTextFlow Property

Description

Informs about the way in which text flows around an embedded object. Not available at design time; read-only at run time.

Syntax

[form.]*VisualWriter.***ObjectTextFlow** [= *mode*]

Remarks

The property settings are:

Setting	Description
1	The object has been inserted 'as character' using the ObjectInsertAsChar method.
2	The object has been inserted as fixed object. The text overwrites the object.
3	The object has been inserted as fixed object. The text stops at the top and continues at the bottom of the object.
4	Same as 3, but empty areas at the left and right side are filled.

Data Type

Integer

See also

ObjectInsertFixed Method

PageHeight Property

Description

Specifies the height of the printer page.

Syntax

[form.]*VisualWriter.***PageHeight** [= *height*]

Remarks

The height of the actual printed area is PageHeight minus PageMarginB minus PageMarginT. The maximum value depends on the capabilities of the selected printer and must not exceed 32767 twips. (Twips is the default scale in Visual Basic. One twip is a twentieth of a Point. There are 1,440 twips to one inch).

If PageHeight is 0, the Height property is used instead. This setting can be used to place several controls without scrollbars on a page. The PageMarginT property then determines the vertical position of the control.

Data Type

Long

See also

PageWidth Property

PageMarginx Properties

PrintDevice Property

PrintPage Method

Example

See PrintPageMethod example.

PageMarginB Property

Description
Specifies the bottom margin on the printed page.

Syntax

[form.]*VisualWriter.***PageMarginB** [= *margin*]

Remarks
The maximum value depends on the setting of the PageHeight property.

Data Type
Long

See also
PageHeight Property

PageMarginL Property

PrintDevice Property

PrintPage Method

Example
See PrintPage Method PrintPageMethod example.

PageMarginL Property

Description
Specifies the left margin on the printed page.

Syntax

[form.]*VisualWriter.***PageMarginL** [= *margin*]

Remarks
The maximum value depends on the setting of the PageWidth property.

Data Type
Long

See also
PageWidth Property

PageMarginR Property

PrintDevice Property

PrintPage Method

PageHeight Property

Example
See PrintPageMethod example.

PageMarginR Property

Description
Specifies the right margin on the printed page.

Syntax

[form.]*VisualWriter.***PageMarginR** [= *margin*]

Remarks
The maximum value depends on the setting of the PageWidth property.

Data Type
Long

See also
PageWidth Property

PageMarginT Property

PrintDevice Property

PrintPage Method

PageHeight Property

Example
See PrintPage Method example.

PageMarginT Property

Description
Specifies the top margin on the printed page.

Syntax

[form.]*VisualWriter.***PageMarginT** [= *margin*]

Remarks
The maximum value depends on the setting of the PageHeight property.

Data Type
Long

See also
PageWidth Property

PageMarginL Property

PrintDevice Property

PrintPage Method

PageHeight Property

Example
See PrintPage Method example.

PageWidth Property

Description
Specifies the width of the printed page.

Syntax

[form.]*VisualWriter.***PageWidth** [= *height*]

Remarks
The width of the actual printed area is PageWidth minus PageMarginR minus PageMarginL. The maximum value depends on the capabilities of the selected printer and must not exceed 32767 twips. (Twips is the default scale in Visual Basic. There are 1,440 twips to one inch).

If PageWidth is 0, the Width property is used instead. This setting can be used to place several controls without scrollbars on a page. The PageMarginL property then determines the horizontal position of the control.

Data Type
Long

See also
PageMarginx Properties

PrintDevice Property

PrintPage Method

PageHeight Property

Example
See PrintPage Method example.

ParagraphDialog Method

Description

Invokes VisualWriter's built-in paragraph attributes dialog box and, after the user has closed the dialog box, specifies whether he has changed something. Not available at design time; read-only at run time.

Syntax

[form.]*VisualWriter.***ParagraphDialog**.

Remarks

The changes made in the dialog box apply to the currently selected text. The method returns one of the following values:

Return Value	Description
True	The user has changed one or more attibutes.
False	The formatting remains unchanged.

Data Type

Boolean

Parent Property

Description

This is a standard Visual Basic property.

Specifies the form on which a control is located.

Syntax

[form.]*VisualWriter.***Parent**

Remarks

Use the Parent property to access the properties, methods, or controls of a controls parent form   for example, MyButton.Parent.MousePointer = 4.

Data Type

Form

PosChange Event

Description

Occurs when the current character input position has been changed.

Syntax

```
Sub VisualWriter_PosChange()
```

PrintColors Property

Description

Specifies if text colors are printed as colors or in black.

Syntax

[form.]*VisualWriter.***PrinterColors** [= *boolean*]

Data Type

Boolean

PrintDevice Property

Description
Specifies the printer device context for TextContol's built-in print function. Not available at design time.

Syntax

[form.]*VisualWriter.***PrintDevice** [= *device context handle*]

Data Type
Integer

See also
PageMarginx Properties

PrintDevice Property

PrintPage Method

PageHeight Property

Example
See PrintPage Method example.

PrintOffset Property

Description

Determines if VisualWriter will start printing exactly at the top left corner of the page, or, like the graphics methods in Visual Basic, at the default printer's printable area. Not available at design time.

Syntax

[form.]*VisualWriter.***PrintOffset** = *boolean*

Remarks

The property settings are:

Setting	Description
True	Use printing offset.
False	Do not use printing offset (Default).

Data Type

Boolean

PrintPage Method

Description
Prints a page of text on the default printer.

Syntax

[form.]*VisualWriter.***PrintPage** *PageNumber*

Remarks
Prior to using this method VisualWriter's output device must be selected using the PrintDevice property.

Data Type
Integer

See also
PageMarginx Properties
PrintDevice Property
PrintPage Method
PageHeight Property

Example
This example shows how to print the contents of VisualWriter on the default printer:

```
Sub mnuFile_Print_Click ()

       Dim wPages, No

       Printer.Print

       wPages = VisualWriter1.CurrentPages

       For No = 1 To wPages

          VisualWriter1.PrintDevice = Printer.hDC

          VisualWriter1.PrintPage No

          Printer.NewPage

       Next No

       Printer.EndDoc

End Sub
```

PrintZoom Property

Description
Specifies the zoom factor for the printer. The value is specified as a percentage in the range of 10-400.

Syntax

[form.]*VisualWriter.***PrintZoom** [= *zoom*]

Data Type
Integer

See also
ZoomFactor Property

Redo Method

Description

The Redo method can be used to redo the last VisualWriter operation.

Syntax

[form.]*VisualWriter.***Redo**

See also

Undo Method

CanUndo Property

CanRedo Property

Refresh Method

Description

This is a standard Visual Basic Method.

Forces an immediate repaint or update of a control.

Syntax

> *VisualWriter*.Refresh

Remarks

Use this method to force a complete repaint of the control.

RTFExport Method

Description

Writes the contents of VisualWriter to a file using the Rich Text Format.

Syntax

[form.]*VisualWriter.***RTFExport** *filename*

Remarks

RTF (Rich Text Format) is one of the most common interchange formats for text documents. Most word processors available for Windows are able to read and write RTF files.

Data Type

String

See also

RTFImport Method

RTFImport Method

Description

Loads the contents of an RTF file into VisualWriter. The text is inserted at the current caret position.

Syntax

[form.]*VisualWriter.***RTFImport** *filename*

Remarks

RTF (Rich Text Format) is one of the most common interchange formats for text documents. Most word processors available for Windows are able to read and write RTF files.

Data Type

String

See also

RTFExport Method

RTFSelText Property

Description

This property works much like the standard SelText Property. The SelStart and SelLength Properties can be used to specify a text selection which is to be copied to a string or inserted from a string. The difference between SelText and RTFSelText is that with the SelText Property, text is stored without formatting information in the ANSI format, while RTFSelText uses Rich Text Format to preserve all of the formatting attributes.

Syntax

[form.]*VisualWriter.***RTFSelText** [= *string*]

Remarks

The text selection that is copied to or inserted from a string must be of type RTF. The RTF string will include the RTF header information which identifies the formatting attributes for the string. RTF (Rich Text Format) is one of the most common interchange formats for text documents. Most word processors available for Windows are able to read and write RTF files.

Data Type

String

See also

RTFImport Method

RulerHandle Property

Description
Specifies the ruler control to be used with VisualWriter. Not available at design time.

Syntax

[form.]*VisualWriter.***RulerHandle** *[= Ruler.hWnd]*

Data Type
Integer (window handle)

See also
StatusBarHandle Property

Save Method

Description

Saves the contents of a text control with all its text and format information in a file.

Syntax

[form.]*VisualWriter.***Save** *filename, offset*

Remarks

You can store more than one text controls data in a single file. The *offset* parameter determines the file position to where the text controls data is written. Also, a file header can be written by the Visual Basic program before the Save method is used. An example of writing a file header can be found in the VisualWriter MDI demo source code.

A file that has been saved using the Save method can only be loaded into VisualWriter via the Load Method.

Data Type

Filename: String
Offset: Integer

See also

Load Method

ScrollBars Property

Description

This is a standard Visual Basic property.

Returns or sets a value indicating whether an object has horizontal or vertical scroll bars. Read only at run time.

Syntax

*VisualWriter.***ScrollBars**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

For an MDIForm object, the ScrollBars property settings are:

Setting	Description
True	(Default) The form has a horizontal or vertical scroll bar, or both.
False	The form has no scroll bars.

For a Grid or TextBox control, the ScrollBars property settings are:

Setting	Description
0	(Default) None
1	Horizontal
2	Vertical
3	Both

Remarks

For a **TextBox** control with setting 1 (Horizontal), 2 (Vertical), or 3 (Both), you must set the **MultiLine** property to **True**.

At run time, the Microsoft Windows operating environment automatically implements a standard keyboard interface to allow navigation in TextBox controls with the arrow keys (UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW), the HOME and END keys, and so on.

Scroll bars are displayed on an object only if its contents extend beyond the objects borders. For example, in an **MDIForm** object, if part of a child form is hidden behind the border of the parent MDI form, a horizontal scroll bar (**HScrollBar** control) is displayed. Similarly, a vertical scroll bar (**VScrollBar** control) is displayed on a **Grid** control when it cant display all of its rows; a vertical scroll bar appears on a **TextBox** control when it cant display all of its lines of text. If **ScrollBars** is set to **False**, the object wont have scroll bars, regardless of its contents.

ScrollPosX Property

Description
Specifies the position of the horizontal scrollbar. Not available at design time.

Syntax

[form.]*VisualWriter.***ScrollPosX**

Data Type
Long

See also
ScrollPosY Property

HScroll Event

VScroll Event

ScrollPosY Property

Description
Specifies the position of the vertical scrollbar. Not available at design time.

Syntax

[form.]*VisualWriter.***ScrollPosY**

Data Type
Long

See also
ScrollPosX Property

HScroll Event

VScroll Event

SelLength Property

Description

This is a standard Visual Basic property.

Determines the number of characters selected.

Syntax

[form.]*VisualWriter.***SelLength** [= *length*]

Remarks

For SelLength, the valid range of settings is 0 to text length   the total number of characters in the edit area of a text box.

Use this property for tasks such as setting the insertion point, establishing an insertion range, selecting substrings in a control, or clearing text. Used in conjunction with the Clipboard, this property is useful for copy, cut, and paste operations.

Data Type

Long

See also

Text property

SelStart Property

Description

This is a standard Visual Basic property.

Determines the starting point of selected text.

Syntax

[form.]*VisualWriter.***SelStart** [= *index*]

Remarks

For SelStart, the valid range of settings is 0 to text length   the total number of characters in the edit area of a text box.

Use this property for tasks such as setting the insertion point, establishing an insertion range, selecting substrings in a control, or clearing text. Used in conjunction with the Clipboard, this property is useful for copy, cut, and paste operations.

Data Type

Long

See also

Text property

SelText Property

Description

This is a standard Visual Basic property.

Determines the string containing the currently selected text.

Syntax

[form.]*VisualWriter.***SelText** [= *stringexpression*]

Remarks

Use this property for tasks such as setting the insertion point, establishing an insertion range, selecting substrings in a control, or clearing text. Used in conjunction with the Clipboard, this property is useful for copy, cut, and paste operations.

Data Type

String

See also

Text property

SetFocus Method

Description

This is a standard Visual Basic method.

Moves the focus to the specified control or form.

Syntax

*VisualWriter.***SetFocus**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

The object must be a **Form** object, **MDIForm** object, or control that can receive the focus. After invoking the **SetFocus** method, any user input is directed to the specified form or control.

You can only move the focus to a visible form or control. Because a form and controls on a form arent visible until the forms Load event has finished, you cant use the **SetFocus** method to move the focus to the form being loaded in its own Load event unless you first use the **Show** method to show the form before the Form_Load event procedure is finished.

You also cant move the focus to a form or control if the **Enabled** Property is set to **False**. If the **Enabled** property has been set to **False** at design time, you must first set it to **True** before it can receive the focus using the **SetFocus** method.

See also

Enabled Property

Load Statement

Show Method

Size Event

Description

Occurs when VisualWriter has been resized with the mouse while depressing the ALT key.

Syntax

Sub VisualWriter_**Size**

See also

Move Event

SizeMode Property

SizeMode Property

Description

Specifies whether the VisualWriter window can be moved or resized at runtime, in the way it can at design time. If the Moveable option is selected, the control can be moved on the background by depressing the ALT key and then dragging the control with the mouse. If the Sizeable option is selected and the ALT key is depressed, the borders of the control can be dragged.

Syntax

[form.]*VisualWriter.***SizeMode** = *mode*

Remarks

The property settings are:

Setting	Description
0 - Fixed	VisualWriter window cannot be moved or resized. (Default).
1 - Moveable	VisualWriter window can be moved.
2 - Sizeable	VisualWriter window can be resized.
3 - Move and Sizeable	Both 1 and 2.

Data Type

Integer

StatusBarHandle Property

Description
Specifies the status bar control to be used with VisualWriter. Not available at design time.

Syntax

[form.]*VisualWriter.***StatusBarHandle** *[= StatusBar.hWnd]*

Data Type
Integer (window handle)

See also
ButtonBarHandle Property

TabCurrent Property

Description

Specifies the current tab number for the properties TabPos and TabType. Not available at design time.

Syntax

[form.]*VisualWriter.***TabCurrent** [= *tab number*]

Remarks

VisualWriter supports up to 14 tabs for each paragraph. The tabs are numbered 1 to 14.

Data Type

Integer

See also

TabPos Property

TabType Property

Example

This example moves the first tab to a new position 1 inch from the left border and changes it to a decimal tab:

```
VisualWriter1.TabCurrent = 1

VisualWriter1.TabType = 4

VisualWriter1.TabPos = 1440

       The next example changes all the tabs to be right aligned at 1/2 inch
       gradations:

' Delete all tabs

for n=14 to 1 step -1

       VisualWriter1.TabCurrent = n

       VisualWriter1.TabPos = 0

next n

' Create new tabs

for n=1 to 14

       VisualWriter1.TabCurrent = n

       VisualWriter1.TabPos = n*720

       if (VisualWriter1.TabPos > 0) then VisualWriter1.TabType = 2

next n
```

VisualWriter sorts the tabs in ascending order whenever you change the position of a tab, so a tab's number can change when it it moved. In this case, the TabCurrent property is updated to reflect the change.

Tabs outside of the page are automatically set to zero.

TabIndex Property

Description

This is a standard Visual Basic property.

Determines the tab order of a control within its parent form.

Syntax

[form.]*VisualWriter.***TabIndex** [= *index*]

Remarks

The valid range is any integer from 0 to (n-1), where n is the number of controls on the form that have a TabIndex Property. Assigning a TabIndex value of less than 0 generates an error.

Data Type

Integer

See also

TabStop property

ZOrder method

TabKey Property

Description

Determines if the Tab key is used to move the focus to the next control or to insert Tabs in VisualWriter which currently has the focus.

Syntax

[form.]*VisualWriter.***TabKey** [= *boolean*]

Remarks

Valid settings are:

Setting	Description
True	Inserts a Tab in VisualWriter. (Default)
False	The focus is moved to the next control.

Data Type

Boolean

TabPos Property

Description

Determines the position (in twips) of a tab. The tab number must have previously been determined with the TabCurrent property.

Syntax

[form.]*VisualWriter.***TabPos** [= *position*]

Data Type

Long

See also

TabCurrent Property

TabType Property

TabStop Property

Description

This is a standard Visual Basic property.

Determines whether a user can use the TAB key to set the focus to a control.

Syntax

[form.]*VisualWriter.***TabStop** [= *boolean*]

Remarks

The TabStop property settings are:

Setting	Description
True	(Default) Designates the control as a tab stop.
False	Bypasses the control when the user is tabbing, although the control still holds its place in the actual tab order, as determined by the TabIndex Property.

Type

Integer (Boolean)

See also

TabIndex Property

TabType Property

Description

Determines the tab type. The tab number must have previously been determined with the TabCurrent property .

Syntax

[form.]*VisualWriter.***TabType** [= *type*]

Remarks

Valid settings are:

Setting	Description
1	Left tab.
2	Right tab.
3	Centered tab.
4	Decimal tab.

Data Type

Integer

See also

TabCurrent Property

TabPos Property

Tag Property

Description

This is a standard Visual Basic property.

Stores any extra data needed for your program.

Syntax

[form.]*VisualWriter.***Tag** [= *expression*]

Remarks

By default, the Tag property is set to an empty string. ("")

You can use this property to assign an identification string to an object without affecting any of its other property settings or causing side effects. The Tag Property is useful when you need to check the identity of a control that is passed as a variable to a procedure.

Data Type

String

See also

Name Property

Text Property

Description

This is a standard Visual Basic property.

Does the following:

- Returns or sets the text contained in the edit areaComboBox control (Style property set to 0 [Dropdown Combo] or to 1 [Simple Combo]) and TextBox control.
- Returns the selected item in the list box; the value returned is always equivalent to the value returned by the expression List (ListIndex). Read-only at design time; read-only at run timeComboBox control (Style property set to 2 [Dropdown List] and ListBox control.
- Returns or sets the text contained in a cell or range of cells. Not available at design timeGrid control.

Syntax

object.**Text** [=*string*]

The Text property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
string	A string expression specifying text.

Remarks

At design time only, the defaults for the **Text** property are:

- **ComboBox** and **TextBox** controlsthe controls **Name** property.
- **ListBox** controla zero-length string ().

For a **ComboBox** with the **Style** property set to 0 (Dropdown Combo) or to 1 (Simple Combo) or for a **TextBox**, this property is useful for reading the actual string contained in the edit area of the control. For a **ComboBox** or **ListBox** control with the **Style** property set to 2 (Dropdown List), you can use the **Text** property to determine the currently selected item.

The **Text** setting for a **TextBox** control is limited to 2048 characters unless the **MultiLine** property is **True**, in which case the limit is about 32K.

For a **Grid** control, you can add text to a single cell by setting the **Text** property. This property applies to the cell defined by the current values of the **Grid** controls **Row** and **Col** properties.

You can use the **Text** and **FillStyle** properties to add the same text to a highlighted range of cells. When **FillStyle** = 0, the text assigned to the **Text** property is added only to the cell defined by the current **Row** and **Col** property values. When **FillStyle** = 1, the text is added to all cells whose **CellSelected** property setting is **True**.

You can also use the **Clip** property to fill a range of cells. For example, you might want to paste a large block of information from the **Clipboard** into a **Grid** control.

See also

SelLength Property

SelStart Property

SelText Property

TextBkColor Property

Description

Determines the background color for selected text.

Syntax

[form.]*VisualWriter.*__TextBkColor__ [= *RGB value*]

Remarks

The TextBkColor property applies only to the currently selected text. The BackColor standard property can be used to set the window background color.

Data Type

Long

TextExport Method

Description
Writes the selected text to a file.

Syntax

[form.]*VisualWriter.***TextExport** (*filename)*

Data Type
String

See also
TextImport Method

RTFExport Method

Load Method

TextImport Method

Description
Loads text in ASCII format and inserts it at the current caret position.

Syntax

[form.]*VisualWriter.***TextImport** (*filename)*

Data Type
String

See also
TextExport Method

RTFImport Method

Load Method

Top Property

Description

This is a standard Visual Basic property.

Determines the distance between the internal top edge of an object and the top edge of its container.

Syntax

[form.]*VisualWriter.***Top** [= *y*]

Remarks

You can specify a single-precision number. Use Left and Top properties and the Height and Width properties for operations based on an objects external dimensions, such as moving or resizing.

Data Type

Single

See also

Left property

Move method

Undo Method

Description

The Undo method can be used to undo the last VisualWriter operation.

Syntax

[form.]*VisualWriter.***Undo**

See also

Redo Method

CanUndo Property

CanRedo Property

ViewMode Property

Description

Determines the mode in which VisualWriter displays the document pages.

Syntax

[form.]*VisualWriter.*ViewMode = *mode*

Remarks

Valid ViewMode settings are:

Setting	Description
0 (default)	Do not display page borders. This was the only available mode in earlier versions of VisualWriter.
1	Display the document pages with page margins and show the page number in the status bar.

Data Type

Integer

Visible Property

Description

This is a standard Visual Basic property.

Returns or sets a value indicating whether an object is visible or hidden.

Syntax

object.**Visible** [=*boolean*]

The **Visible** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
boolean	A Boolean expression specifying whether the object is visible or hidden.

The settings for *boolean* are:

Setting	Description
True	(Default) Object is visible.
False	Object is hidden.

Remarks

To hide an object at startup, set the Visible property to False at design time. Setting this property in code enables you to hide and later redisplay a control at run time in response to a particular event.

Note Using the **Show** or **Hide** method on a form is the same as setting the forms **Visible** property in code to **True** or **False**, respectively.

See also

Hide Method

Load Statement

Show Method

Example

This example creates animation using two PictureBox controls. To try this example, paste the code into the Declarations section of a form that contains two icon-sized PictureBox controls. Set the Name property to FileCab for both PictureBox controls to create an array, and then press F5 and click the picture to view the animation.

```
Private Sub Form_Load ( )

   Dim I                      Declare variable.

   FileCab(0).BorderStyle = 0       Set BorderStyle.

   FileCab(1).BorderStyle = 0

   Load icons into picture boxes.

   FileCab(1).Picture = LoadPicture (ICONS\OFFICE\FILES03B.ICO)
   FileCab(0).Picture = LoadPicture (ICONS\OFFICE\FILES03A.ICO)

   For I = 0 To 1

          FileCab(I).Move 400. 400   Place graphics at same spot.

   Next I

   FileCab(1).Visible = False       Set to invisible.

   FileCab(0).Visible =  True       Set to visible.

End Sub

Private Sub FileCab_Click (Index As Integer)

   Dim I                            Declare variable.

   For I = 0 To 1

          Switch the visibility for both graphics.

          FileCab(I).Visible = Not FileCab(I).Visible

   Next I

End Sub
```

VScroll Event

Description
Occurs when the vertical scroll position has been changed.

Syntax

Sub VisualWriter_**VScroll**()

See also
HScroll Event

VTSpellCheck Method

Description

Starts the spellchecker. This method is only available if the VisualSpeller tool from Visual Components has been installed. VisualSpeller is not part of VisualWriter package.

Syntax

[form.]*VisualWriter.***VTSpellCheck**

Remarks

VSpell32.dll or VSpell16.dll and the VisualSpeller dictionary must be in the same directory as the VW32.OCX or VW16.OCX.

Data Type

Integer

See also

VTSpellDictionary Property

VTSpellDictionary Property

Description

Determines the dictionary which is used by VisualSpeller.VisualWriter uses this property only if the VisualSpeller tool from Visual Components has been installed. VisualSpeller is not part of VisualWriter package.

Syntax

[form.]*VisualWriter.***VTSpellDictionary** [= *filename*]

Data Type

String

See also

VTSpellCheck Method

Width Property

Description

This is a standard Visual Basic property.

Determines the dimensions of an object.

Syntax

[form.]*VisualWriter.***Width** [= *numericexpression*]

Remarks

Measurements are calculated as follows:

- Form   the external height and width of the form, including the borders and title bar.
- Control measured from the center of the controls border so that controls with different border widths align correctly. These properties use the scale units of a controls container.
- Printer object   the physical dimensions of the paper set up for the printing device; not available at design time and read-only at run time.
- Screen object   the height and width of the screen; not available at design time and read-only at run time.

For a form, Printer object and Screen object, these properties are always measured in twips. For a form or control, the values for these properties change as the object is sized by the user or by code. Maximum limits of these properties for all objects are system-dependent.

Data Type

Single

See also

Height property

Left property

Top Property

Move method

ZOrder Method

Description

This is a standard Visual Basic method.

Places a specified control at the front or back of the z-order within its graphical level.

Syntax

VisualWriter.ZOrder [*position*]

Part	Type	Description
position	Integer	Indicates the position of the control relative to other controls. If postion is 0 or omitted, the control is postioned at the fornt of the z-order. If position is 1, the control appears at the back of the z-order.

ZoomFactor Property

Description

Specifies the zoom factor for VisualWriter. The value is specified as a percentage in the range of 10-400%.

Syntax

[form.]*VisualWriter.***ZoomFactor** [= *ZoomFactor*]

Data Type

Integer

See also

PrintZoom Property

ButtonBar Properties

The following section lists the properties specific to the Button Bar tool.

Language Property

Language Property

Description

Determines the language in which VisualWriter displays dialog boxes and error messages.

Usage

[form.]VWButtonBar.**Language** [= *Country code]*

Remarks

The default language is determined by the 'iCountry=' setting in win.ini.

Setting	Description
34	Spanish
49, 41, 43	German
else	English

Data Type

Integer

StatusBar Properties

The following section lists the properties specific to the StatusBar tool.

Language Property

TextColumn Property

TextLine Property

TextPage Property

PageModeProperty

Language Property

Description

Determines the language in which VisualWriter displays dialog boxes and error messages.

Usage

[form.]VWStatusBar.**Language** [= *Country code]*

Remarks

The default language is determined by the 'iCountry=' setting in win.ini.

Setting	Description
34	Spanish
49, 41, 43	German
else	English

Data Type

Integer

TextColumn Property

Description
Specifies the text which appears in the Column field of the Status Bar. Default is Col.

Syntax

[form.]VWStatusBar.**TextColumn** [= *text*]

Data Type
String

TextLine Property

Description

Specifies the text which appears in the Line field of the Status Bar. Default is Line.

Syntax

[form.]VWStatusBar.**TextLine** [= *text*]

Data Type

String

TextPage Property

Description

Specifies the text which appears in the Page field of the Status Bar. Default is Page.

Syntax

[form.]VWStatusBar.**TextPage** [= *text*]

Data Type

String

PageMode Property

Description
Determines if the StatusBar's 'Page' field is visible.

Syntax
[form.]VWStatusBar.**PageMode** [= *mode*]

Remarks
The PageMode settings are:

Setting	Description
0	Do not show Page field.
1	Always show Page field.
2	Show Page field only if VisualWriter is in page mode or link mode.

Data Type
String

Ruler Properties

The following section lists the properties specific to the Ruler Bar tool.

Language Property

ScaleUnits Property

Language Property

Description

Determines the language in which VisualWriter displays dialog boxes and error messages.

Usage

[form.]*VWRuler*.**Language** [= *Country code]*

Remarks

The default language is determined by the 'iCountry=' setting in win.ini.

Setting	Description
34	Spanish
49, 41, 43	German
else	English

Data Type

Integer

ScaleUnits Property

Description

Specifies the scale units for the ruler.

Syntax

[form.]*VWRuler.***ScaleUnits** [= *units*]

Remarks

The settings are:

Setting	Description
0	mm
1	cm
2	inch

Data Type

Integer.

VisualWriter Error Codes

Two kinds of errors can occur in a VisualWriter based application:

- **Trappable Errors**. Errors which are directly caused by using one of VisualWriter's Properties. These errors can be trapped with the On Error statement. For example, setting the PageWidth to a value smaller than the right and left page margin will cause an ERR_SMALLWIDTH error.
- **ErrorCode Event**. Errors which result from insufficient memory, corrupted files, or other causes which occur within VisualWriter itself. For these errors, the program receives an ErrorCode event with the error number as a parameter.

Trappable Errors

The following list describes the errors which can be trapped with the On Error statement:

Error Name	Number	Description
ERR_SMALLWIDTH	20000	Page width too small.
ERR_LARGEWIDTH	20001	Page width too large.
ERR_SMALLHEIGHT	20002	Page height too small.
ERR_LARGEHEIGHT	20003	Page height too large.
ERR_LEFTMARGIN	20004	Left margin too large.
ERR_RIGHTMARGIN	20005	Right margin too large.
ERR_TOPMARGIN	20006	Top margin too large.
ERR_BOTTOMMARGIN	20007	Bottom margin too large.
ERR_EVENT	20008	VisualWriter sends an error event to specify what error has occured.
ERR_WINTOOSMALL	20009	The window is too small to load the requested data.
ERR_PRINT	20010	Failure of page print.
ERR_OPENFILE	20011	OpenFile() failed.
ERR_IMG_MEM	20012	Image-Control: Insufficient memory.
ERR_IMG_BADFILE	20013	Image-Control: Non-existent file.
ERR_IMG_UNKNOWN	20014	Image-Control: Unknown file type.
ERR_IMG_UNSUPPORTED	20015	Image-Control: Unsupported compression type.
ERR_IMG_BADFILTER	20016	Image-Control: Filter not found.
ERR_IMG_UNSUPPFILTER	20017	Image-Control: Unsupported filter type.
ERR_NOLOCALMEM	20018	Out of string space.
ERR_ALIGNINVALID	20019	Invalid alignment value (> 3).
ERR_FIELDNUM	20020	Invalid macro field number.
ERR_BOUND_NOTXDATA	20021	Database record does not contain TX data.
ERR_NOGLOBALMEM	20022	Insufficient global memory.
ERR_TABNUM	20023	Invalid tab number in TabCurrent property.
ERR_TABTYPE	20024	Invalid tab type.
ERR_TYPEOFNULLTAB	20025	Attempt to set type of null tab.
ERR_ZOOMVAL	20026	Invalid zoom value.
ERR_SPELL_DICT	20027	Spellchecker dictionary not found.
ERR_INDENT	20028	Invalid indent setting.

ERR_RTF	20029	RTF filter error.
ERR_ICINI	20030	IC.INI not found or invalid.
ERR_PROP_IS_READONLY	20031	Property is read-only.
ERR_IMGNUM	20032	Invalid image number in ImageCurrent property.
ERR_TXDATA_INVALID	20033	Attempt to load invalid VisualWriter data.
ERR_BOUND_NOASCIIDATA	20034	Database record does not contain ASCII data.

Errors Reported by the ErrorCode Event

The following list describes the error codes which are sent by VisualWriter as a parameter of the ErrorCode Event.

Error Name	No.	Description
EVERR_GLOBALMEM	1	Insufficient global memory.
EVERR_LOCALMEM	2	Insufficient local memory.
EVERR_INTERNAL	3	Internal TX error.
EVERR_FILE	4	File read/write error.
EVERR_64K_TEXT	5	Item larger than 64 KB.
EVERR_CLIPBOARD	6	Clipboard read/write error.
EVERR_MODULE	7	Module not found.
EVERR_FORMAT	8	Unknown format.
EVERR_TXT_FORMAT	9	Text filter: Unknown format.
EVERR_TXT_TOKEN	10	Text filter: Illegal token.
EVERR_TXT_READ	11	Text filter: File read error.
EVERR_TXT_WRITE	12	Text filter: File write error.
EVERR_TXT_OPEN	13	Text filter: File cannot be opened.
EVERR_TXT_SIZE	14	Text filter: File contents too large.
EVERR_TXT_UNSUPPORTED	15	Text filter: Unsupported format.
EVERR_IMG_INTERFACE	6	Image filter: Unknown interface.
EVERR_IMG_OPEN	17	Image filter: File cannot be opened.
EVERR_IMG_SIZE	18	Image filter: File contents too large.
EVERR_IMG_FORMAT	19	Image filter: Unknown format.
EVERR_IMG_UNSUPPORTED	20	Image filter: Unsupported format.
EVERR_IMG_ABORT	21	Image filter: Import aborted.

Mouse and Keyboard Assignments

VisualWriter reacts to a variety of assigned mouse actions and keyboard activity. The following sections outline each assignment and the reaction of VisualWriter.

Mouse Assignment

Mouse Action	Reaction of VisualWriter
Click	Moves cursor to point of click or selects an image.
Shift+Click	Extends the selection to the point of click.
Double-click	Selects the word that is clicked on or opens a modal dialog box to select an image alignment.
Drag	Selects text from point of button down to point where button is released.
Double-click and drag	Extends the selection from word to word.
Triple-click and drag	Extends the selection from row to row.
PgUp/PgDown	Scrolls the text up or down one client area height minus the height of one line of text. Active only if a vertical scrollbar exists.

Moving the caret while SHIFT is pressed extends the current selection to the new caret position.

Keyboard Assignment

Key type	Reaction of VisualWriter
HOME	Moves the caret to the beginning of the line.
END	Moves the caret to the end of the line.
(Left Arrow)	Moves the caret one character to the left.
(Right Arrow)	Moves the caret one character to the right.
(Up Arrow)	Moves the caret one line up.
(Down Arrow)	Moves the caret one line down.
CTRL+(Left Arrow)	Moves the caret to the beginning of the current word.
CTRL+(Right Arrow)	Moves the caret to the beginning of the next word.
CTRL+HOME	Moves the caret to start of text.
CTRL+END	Moves the caret to end of text.
CTRL+ENTER	Inserts a new page.
SHIFT+ENTER	Creates a line feed.
CTRL+(-)	Inserts an end-of-line hyphen.
DEL	Deletes selected text.
SHIFT+DEL	Copies selected text to the Clipboard and deletes the selection.
CTRL+INS	Copies selected text to the clipboard.
SHIFT+INS	Inserts text from the clipboard.
CTRL+SHIFT+(Spacebar)	Inserts a non-breaking space.
CTRL+(Backspace)	Deletes the previous word.

Moving the caret while SHIFT is pressed extends the current selection to the new caret position.

Using VisualWriter

This section shows you how to create a small word processor from scratch with just a few lines of code. It will be able to load and save files, use the clipboard, and will have dialog boxes for character and paragraph formatting, a ruler, a status bar, and full keyboard and mouse interface.

The source code for this example is contained in the Simple sample source directory.

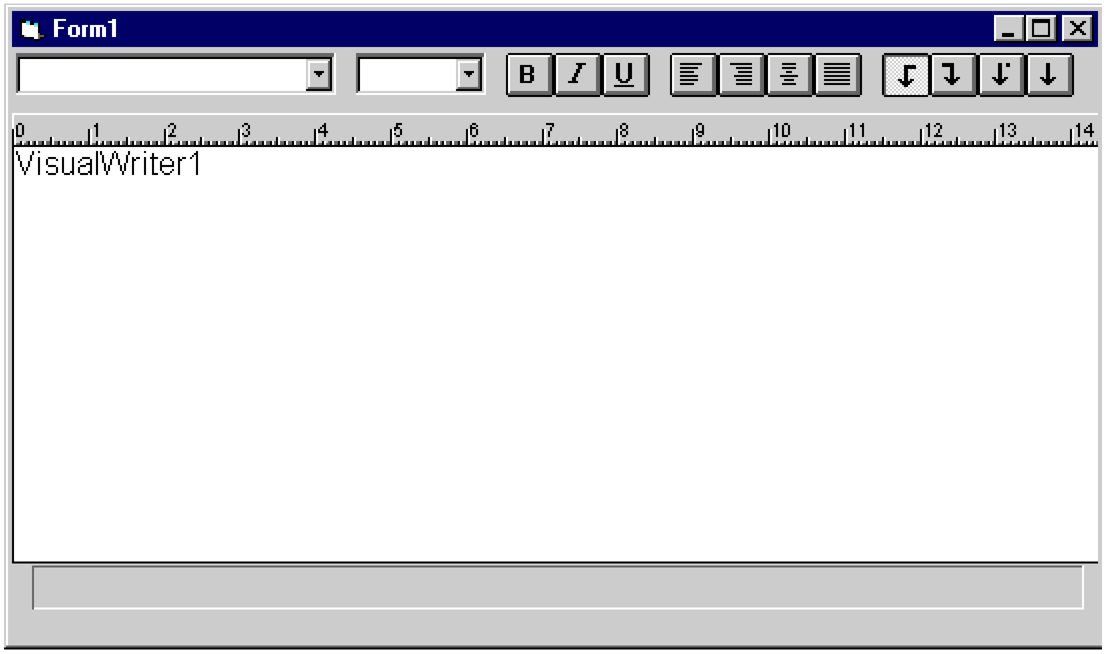
Creating the Project

Assuming that you have already run the VisualWriter installation program and started Visual Basic, the next step is to create a project for the text processor. To do this begin by selecting the New Project command from the file menu. Then use the Tools / Custom Controls... command to include the file 'VW32.OCX', or 'VW16.OCX' for 16 bit applications, into the new project. You will see four additional icons appear at the bottom of the toolbox, representing the VisualWriter text control, Status Bar, Button Bar, and Ruler:

The VisualWriter Icon
The Status Bar Icon
The Button Bar Icon
The Ruler Icon

Creating the Controls

The next step is to put these four controls on a form and connect them. Click on the VisualWriter icon and draw it on the form. In the same way, create a Ruler and a Button Bar on top of the VisualWriter text control, and a Status Bar below it. Your form should now look like this:



Connecting the Controls in Visual Basic

Add the following code to the form's Load Event procedure:

```
Private Sub Form_Load()
    VisualWriter1.ButtonBarHandle = VWButtonBar1.hWnd
    VisualWriter1.RulerHandle = VWRuler1.hWnd
    VisualWriter1.StatusBarHandle = VWStatusBar1.hWnd
End Sub
```

Running the Program

The text processor is not yet finished, but we can make a first attempt at running it and seeing what it can do. Click the "Start" button. You can type in some text, select it with the mouse, copy it to the clipboard (use the <CTRL>+<INS> keys as long as there is no menu), select a different font, set tabs, and do lots of other things. All of these features have been built into VisualWriter and can be used with almost no programming effort.

You will have noticed, however, that some features are still missing. For instance, if you resize the main window, the Controls keep their old sizes. There is no menu, and there are no scrollbars either. We will fix this in the coming sections.

Adding Scrollbars

To add Scroll Bars, click on the VisualWriter window to have its property list displayed. Click on the Scrollbars property and select 3 - Both. Select the PageWidth property and enter 12000, which is about the width of a letter in twips, the currently selected measurement. Set PageHeight to 15000 for now.

Resizing the Controls

Two steps are involved in making the controls resize properly when the main window is resized.

- Set the "Align" Property to "1 - Align Top" for the Button Bar, the Ruler, and the VisualWriter text control. Set it to "2 - Align Bottom" for the Status Bar. This will adjust everything except the height of VisualWriter.
- Open the code window for the form which contains VisualWriter. In the combo boxes on top of the code window, select "Form" in the "Object:" box and "Resize" in the "Proc:" box. The code window should show an empty procedure for the "Resize" event:

```
Private Sub Form_Resize ()
End Sub
```

Extend it as follows:

```
Private Sub Form_Resize ()
    VisualWriter1.Height = ScaleHeight - VWRuler1.Height _
    - VWStatusBar1.Height - VWButtonBar1.Height
End Sub
```

This line of code will cause the VisualWriter's height to be adjusted every time the size of the form is altered. (The ' _' character is used to extend one logical line of code to two or more physical lines).

Adding a Menu

In this section, you will add a menu to the text processor to enable you to call VisualWriters built-in dialog boxes.

Use the Visual Basic Menu Design Window to create a "Format" menu with the items "Character..." and "Paragraph...". Name the items "mnuFomat_Character" and "mnuFormat_Paragraph". (Please refer to the Visual Basic documentation if you need help with creating menus).

Add the following code to the "Click" procedures of the menu items:

```
Private Sub mnuFormat_Character_Click()
    VisualWriter1.FontDialog
End Sub
Private Sub mnuFormat_Paragraph_Click()
    VisualWriter1.ParagraphDialog
End Sub
```

Start the program again. You should be able to use the menu items to call the Font and Paragraph dialog boxes.

Now for the "Edit" menu. Again use the Menu Design Window and create an "Edit" menu containing items for "Cut", "Copy", and "Paste". The code for these menu items is:

```
Private Sub mnuEdit_Cut_Click()
    VisualWriter1.Clip 1
End Sub
Private Sub mnuEdit_Copy_Click()
    VisualWriter1.Clip 2
End Sub
Private Sub mnuEdit_Paste_Click()
    VisualWriter1.Clip 3
End Sub
```

Having added these menu items, you can exchange formatted text with other word processors via the clipboard.

The last menu for now shall be a simple file menu. Create a "File" menu including the items "Load..." and "Save As...". Place a common dialog box icon on the form and enter the following code, which will call the common dialog box to get a file name from the user, and will then load and respectively save the selected file:

```
Private Sub mnuFile_Load_Click()
    On Error Resume Next

    ' Create an "Open File" dialog box
    CommonDialog1.Filter = "TX Demo (*.tx)|*.tx"
    CommonDialog1.DialogTitle = "Open"
    CommonDialog1.Flags = cdlOFNFileMustExist Or _
        cdlOFNHideReadOnly
    CommonDialog1.CancelError = True
    CommonDialog1.ShowOpen
    If Err Then Exit Sub

    ' Pass the filename to the text control
    VisualWriter1.Load CommonDialog1.filename, 0
End Sub

Private Sub mnuFile_SaveAs_Click()
    On Error Resume Next
    ' Create a "Save File" dialog box
    CommonDialog1.Filter = "TX Demo (*.tx)|*.tx"
    CommonDialog1.DialogTitle = "Save As"
    CommonDialog1.Flags = cdlOFNOverwritePrompt Or _
        cdlOFNHideReadOnly
    CommonDialog1.CancelError = True
    CommonDialog1.ShowSave
    If Err Then Exit Sub

    ' Open the selected file
    VisualWriter1.Save CommonDialog1.filename, 0
End Sub
```

What Comes Next

VisualWriter has of course many more features than those included in our little demo program. It is up to you now to include zoom, paragraph frames, and whatever else makes up a full-blown word processor. If you need some hints about how to integrate special features, have a look at the source code of the other sample programs.

Advanced Functions

Once you have learned how to create a simple word processor, you may want to explore the advanced functions offered by VisualWriter. The following topics provide information beyond basic word processing:

Working with Files

VisualWriter uses 3 different file formats:

- Its own, native format, which you would normally use to store data in document files.
- The Rich Text Format (RTF), which can be used to exchange formatted text with other applications.
- Unformatted ASCII text.

An example of how to use the native file format has already been presented in the section titled Using VisualWriter. Using RTF or ASCII is simple: just assign a file name to the RTFImport or RTFExport, TextImport or TextExport Method to load or save a file.

With the RTF and ASCII methods, you can only read or write the contents of a single text control from or to a file. Using the Load and Save methods, however, you can write a file header prior to saving the VisualWriter data, or even write the contents of several text controls to one file. This is the reason why the Load and Save methods take a parameter which determines the position within the file where the data is written to or read from.

The Forms1 sample program shows you how to write the contents of multiple text controls to a single file. The MDIDemo sample shows you how to write a file header prior to the text controls data.

Printing

Visual Basic provides two techniques for sending information to the printer. The first one is to use the PrintForm method, the second is to use the printer object. Both methods have their drawbacks: PrintForm works with screen resolution only, which would result in very poor print quality.   The printer object, on the other hand, provides the best print quality, but requires a lot of coding. VisualWriter uses the second method to achieve the best result, but without a lot of coding.

The following example sends the contents of VisualWriter, which can be several pages long, to the default printer:

```
Sub mnuFile_Print_Click ()
    Dim wPages As Integer, No As Integer
    Printer.Print
    wPages = VisualWriter1.CurrentPages
    For No = 1 To wPages
        VisualWriter1.PrintDevice = Printer.hDC
        VisualWriter1.PrintPage No
        Printer.NewPage
    Next No
    Printer.EndDoc
End Sub
```

After initializing the printer object with the 'Printer.Print' statement, the number of pages is stored in a local variable called 'wPages'. The following 'For .. Next' loop runs from 1 to 'wPages' to print all of the pages. Inside the loop there are three lines of code which print a single page:

1. The device context handle of the printer object is assigned to VisualWriter's PrintDevice property. Without this step, a device context which is compatible to the screen device would be used, resulting again in poor print quality.

2. The number of the page to be printed is passed as a parameter to the PrintPage method. This will also start the printing process.

3. The printer object's NewPage method is invoked to advance to the next page.

Everything else, like calculating the line and page breaks, is done internally by VisualWriter. The formatting is based on the values of two groups of properties:

- PageHeight and PageWidth determine the dimensions of the printed page.
- PageMarginB, PageMarginL, PageMarginR and PageMarginT determine the print margins.

These properties are normally set in a page setup dialog box.

Using Multiple Controls

This section shows how to use VisualWriter in programs which have several text fields placed on a single page. Think of a program to print labels, to fill out forms, or to mask data entry. The Forms1 sample program, which can be found on the disk, provides the basic functionality for applications of this kind.

Running the Sample Program

Initially, when the program is started, the main window contains one framed text control where text can be entered. The rest of the window is empty.

What you can do with the program is:

- Move the text control by pressing the ALT key and dragging the window with the mouse.
- Resize the text control by pressing the ALT key and dragging the window borders with the mouse.
- Create additional controls by clicking on an empty part of the main window.
- Save, load or print.

To keep things simple, there are no scrollbars in the main window and no menu items except the ones listed above. For more information, see the Adding Scrollbars, and Zooming topics.

How it Works

The Forms1 sample uses a control array for the text fields. The first text control, the one which you see when you start the program, is placed on the form at design time. More controls are created when you click on an empty area of the form. These controls are created dynamically with the Visual Basic Load function when a MouseDown Event occurs on the form:

```
Private Sub Form_MouseDown(Button As Integer, _
        Shift As Integer, X As Single, Y As Single)
    MaxID = MaxID + 1
    Load VisualWriter1(MaxID)
    VisualWriter1(MaxID).Move X, Y
    VisualWriter1(MaxID).Visible = True
    VisualWriter1(MaxID).ZOrder
End Sub
```

Clicking on an existing text field brings it to the front. This is done by changing the Z order when a Click Event has occured:

```
Private Sub VisualWriter1_Click(Index As Integer)
    VisualWriter1(Index).ZOrder
End Sub
```

The global variable MaxID counts the total number of controls; it is initialized to a value of 1 when the form is loaded.

Moving and resizing the controls is done by VisualWriter itself. To enable these functions, the SizeMode property must be set to "3 - Move and Sizeable".

Saving the Controls

Saving a document which has been created with this program necessitates storing not only the data contained in the text controls, but also the number and the positions of the controls. In addition, a format identifier should be stored to enable the load routine of the program to determine if it can process a file which it is about to load. The following code shows you how to save the document.

```
Private Sub mnuFile_SaveAs_Click()
    On Error Resume Next

    Dim i As Integer, FileID As Long
    Dim xPos As Single, yPos As Single
    Dim xSize As Single, ySize As Single

    ' Create a "Save File" dialog box
    CommonDialog1.Filter = "TX Form Demo (*.txf)|*.txf"
    CommonDialog1.DialogTitle = "Save As"
    CommonDialog1.Flags = cdlOFNOverwritePrompt Or _
        OFNHideReadOnly
    CommonDialog1.CancelError = True
    CommonDialog1.ShowSave
    If Err Then Exit Sub

    ' Open the file
    Open CommonDialog1.filename For Binary As #1
    If Err Then
        MsgBox "Can't open file: " + CommonDialog1.filename
        Exit Sub
    End If

    ' Write file header consisting of file format ID
    ' and number of controls
    FileID = FILE_ID
    Put #1, , FileID
    Put #1, , MaxID

    ' Save the position of all text controls
    For i = 1 To MaxID
        xPos = VisualWriter1(i).Left
        yPos = VisualWriter1(i).Top
        xSize = VisualWriter1(i).Width
        ySize = VisualWriter1(i).Height
        Put #1, , xPos
        Put #1, , yPos
        Put #1, , xSize
        Put #1, , ySize
    Next i
    Close #1
    ' Save the contents of all text controls
    For i = 1 To MaxID
        VisualWriter1(i).Save CommonDialog1.filename
    Next i
End Sub
```

The Load routine first reads the format ID and the number of controls. Then it creates the required number of controls, loads their contents and finally moves them to their correct position:

```
Private Sub mnuFile_Load_Click()
```

```
    On Error Resume Next
    Dim i As Integer, lFilePos As Long
    Dim FileID As Long, xPos As Single, yPos As Single
    Dim xSize As Single, ySize As Single
    ' Create an Open File dialog box
    CommonDialog1.Filter = "TX Form Demo (*.txf)|*.txf"
    CommonDialog1.DialogTitle = "Open"
    CommonDialog1.Flags = cdlOFNFileMustExist Or _
        cdlOFNHideReadOnly
    CommonDialog1.CancelError = True
    CommonDialog1.ShowOpen
    If Err Then Exit Sub
    ' Open the selected file
    Open CMDialog1.filename For Binary As #1
    If Err Then
        MsgBox "Can't open file: " + CommonDialog1.filename
        Exit Sub
    End If
    ' Read file header
    Get #1, , FileID
    If FileID <> FILE_ID Then
        MsgBox "Wrong file type: " + CommonDialog1.filename
        Close #1
        Exit Sub
    End If
    ' Destroy existing controls
    For i = 2 To MaxID
        Unload VisualWriter1(i)
    Next i
    ' Create text controls and load their contents
    Get #1, , MaxID
    For i = 1 To MaxID
        Get #1, , xPos
        Get #1, , yPos
        Get #1, , xSize
        Get #1, , ySize
        If i <> 1 Then Load VisualWriter1(i)
        VisualWriter1(i).Move xPos, yPos, xSize, ySize
        VisualWriter1(i).Text = ""
    Next i
    lFilePos = Loc(1)
    Close #1
    For i = 1 To MaxID
        lFilePos = VisualWriter1(i).Load _
            (CommonDialog1.filename, lFilePos)
    Next i
End Sub
```

Printing Multiple Controls

Printing a document is quite straightforward. The PageWidth and PageHeight properties are set to a value of 0 at design time, so the controls are printed like they are formatted on the screen. The print margin properties are used to specify the positions of the controls on the page.

```
Private Sub mnuFile_Print_Click()
    Dim i As Integer
    Printer.Print
    For i = 1 To MaxID
        VisualWriter1(i).PrintDevice = Printer.hDC
        VisualWriter1(i).PageMarginL = VisualWriter1(i).Left
        VisualWriter1(i).PageMarginT = VisualWriter1(i).Top
        VisualWriter1(i).PrintPage 1
    Next i
    Printer.NewPage
    Printer.EndDoc
End Sub
```

The complete source code of the Forms1 sample program is contained in the Forms1 sample source directory.

A Forms Filler

With the Forms1 sample program, you can place text fields at arbitrary positions on a page. When you print the page, the text fields appear on the paper at exactly the same positions where they were previously placed on the screen. These features will be used in the following sample to create a program for filling out pre-printed forms.
The scanned image of the form is shown in the background of the screen, enabling the user to easily determine the positions of the filled-out fields. He has only to click (with the CTRL key pressed) on the area of the form where he wants to put text and then start typing. The fields can be moved and resized afterwards by holding down the ALT key and dragging them with the mouse.

The source code for this example is contained in the Forms2 sample source directory.

Adding ButtonBar, Ruler, and StatusBar

The Button Bar, Ruler, and Status Bar are used in a special way in this sample program. If you run the program and click on various fields you will notice that the tools automatically switch to the text field which has been clicked on. This switching is done internally by VisualWriter, so no programming is required for it. The tools are simply connected to the first member of the VisualWriter array at design time.

Displaying the Background Image

The background image is displayed by an Image-Control. You could also use the Visual Basic PictureBox for this, but the PictureBox can not handle the large image files which result from scanning a full document page, and it does not support the TIFF file format, which is used by most scan programs.

The Image-Control is not a separate custom control, but a child window of VisualWriter. To display the background image, you create a text control which has the size of the whole page, and then load an image using VisualWriter's ObjectInsertAsChar method.

The text control which displays the background image has an additional function, which again saves you a lot of programming work. It acts as a container for the text controls which are used as fill-out fields. (A container control enables you to draw other controls within it at design time. Examples of container controls are frames and picture boxes.) The big advantage of a container is that it handles all of the clipping for the controls which have been created on top of it. Otherwise, scrolling the background image would cause the text fields to overwrite anything that lies within the form's boundaries, like ButtonBar, Ruler, and even the scrollbars. It would require many calculations of field positions and sizes and some direct calls to the Windows DLLs on every scroll and resize Event to do the clipping without a container control. Using the background text control as a container, you need only create the first text field inside of it, and everything else is done automatically.

Working with Transparent Text Controls

Run the program, load a background image and create a few text fields by clicking on this background image. You will notice that the text fields are transparent, so you can see the background image shining through. Using this feature in a program requires some fine-tuning of the clipping areas with the ClipChildren and ClipSiblings properties.

These two properties determine which areas of an image are repainted when a new part of a control becomes visible or when its contents have been changed. For example, if one control is covered by another, it only has to be repainted if the one which lies on top of it is transparent. You will always want to repaint as little as possible to make the application run fast and to avoid unnecessary flickering on the screen, and you will not want your computer to spend time drawing things which are not visible.

For maximum flexibility in setting the clipping areas and mixing transparent and opaque controls, two properties have been implemented which share this task:

The ClipChildren property is used only for text controls which act as a container for other text controls. When ClipChildren is set to True, the areas occupied by the child controls are excluded from the update area. So, if as in the forms filler program, transparent controls are used as children of the container control, this property must be set to False.

The ClipSiblings property determines the behavior between each of the child controls. It must be set to False if the program allows transparent text controls to overlap others.

Zooming

Zooming is simply done by setting the Zoom property of each of the text controls:

```
Private Sub mnuView_ZoomItem_Click(Index As Integer)
    Dim nZoom As Integer, i As Integer
    nZoom = Val(Mid$(mnuView_ZoomItem(Index).Caption, 2))
    VisualWriter2.ZoomFactor = nZoom
    For i = 1 To MaxID
        VisualWriter1(i).ZoomFactor = nZoom
    Next i
    For i = 1 To 5
        mnuView_ZoomItem(i).Checked = (i = Index)
    Next
End Sub
```

Using Marked Text Fields

Marked Text Fields are markers which are inserted in the text. They can be used to implement a wide range of special functions in a text processor. To name just a few:

- Mail Merge functions
- Spreadsheet-like calculation fields
- Bookmarks
- Automatic table of contents and index generation
- Hypertext viewers which include any kind of buttons, images, pop-up windows, or even OLE objects in the text

Any group of characters within the text can be a Marked Text Field. The maximum number of fields is 65,535. VisualWriter maintains the positions and numbers of the fields. It also takes care of loading, saving and clipboard operations. Any additional information connected with the fields has to be managed by the application.

A Simple Example

This first sample program will show you how fields are created and what happens when they are clicked on. The code shown here is contained in the FIELD1 sample source directory.

The program consists of a form with just one menu item, 'Insert Field!', with an exclamation mark to say that clicking on this item will cause an immediate action instead of dropping a menu. There are two text-controls on the form, one of which is used as a normal text window (VisualWriter1), the other one as a pop-up window (VisualWriter2).

The following code is executed when the menu item is clicked on:

```
Private Sub mnuInsertField_Click ()
    VisualWriter1.FieldInsert "--------"
End Sub
```

This inserts a field at the current caret position. If you move the cursor over the field, VisualWriter changes the mouse pointer to an upward pointing arrow (ñ) to indicate that there is something to click on.

If you click on the field, the application receives a FieldClicked Event, to which it responds by popping up a window which displays the field number.

Only four lines of code are required for this:

```
Private Sub VisualWriter1_FieldClicked(ByVal FieldIndex _
As Integer)
    VisualWriter1.FieldCurrent = FieldIndex
    VisualWriter2.Text = "This is field no. " & FieldIndex _
        & ". Its text is: " & VisualWriter1.FieldText
    VisualWriter2.Move VisualWriter1.FieldPosX, _
        VisualWriter1.FieldPosY
    VisualWriter2.ZOrder
End Sub
```

The first line selects the Marked Text Field which has been clicked on. Line 2 builds the string that is to be displayed in the pop-up window. Line 3 moves the pop-up window, which is initially hidden behind the text window, to the position of the Marked Text Field. Line 4 puts the pop-up window in front of the text window to make it visible. When the mouse button is released, the text window is moved to the front again:

```
Private Sub VisualWriter1_MouseUp(Button As Integer, _
Shift As Integer, X As Single, Y As Single)
    VisualWriter1.ZOrder
End Sub
```

Bookmarks

This example shows you how to use VisualWriter's Marked Text Fields to create bookmarks. The first version will reference the bookmarks simply by their field numbers. The source code for this example is contained in the FIELD2 sample source directory.

The sample application has a 'Bookmark' menu with two items which are named 'Insert' and 'Go to...'. Clicking 'Insert' creates a Marked Text Field at the current caret position. If a text selection exists, the selected text is converted into a field. If not, the character next to the caret is selected.

```
Private Sub mnuBookmark_Insert_Click()
    If VisualWriter1.Text = "" Then
        MsgBox "Cannot insert bookmark if control is empty."
    Else
        If VisualWriter1.SelLength = 0 Then _
            VisualWriter1.SelLength = 1
        VisualWriter1.FieldInsert ""
    End If
End Sub
```

After typing in some text and inserting a few bookmarks, select the 'Go To...' menu item. This will launch a dialog box which allows you to enter the number of the bookmark to jump to. There is no error processing in this example, so if you enter the number of a non-existent field, nothing will happen.

Clicking the 'OK' button executes the following procedure:

```
Private Sub cmdOk_Click()
    Form1.VisualWriter1.FieldCurrent = Text1.Text
    Form1.VisualWriter1.SelStart = _
        Form1.VisualWriter1.FieldStart - 1
    Form1.VisualWriter1.SelLength = _
        Form1.VisualWriter1.FieldEnd - _
        Form1.VisualWriter1.FieldStart + 1
    Unload Me
End Sub
```

The number which has been entered in the dialog box is taken as a value for the FieldCurrent property.

Adding Strings to Marked Text Fields

In commercial word processors, bookmarks are normally referenced by names, not just by numbers. The names are typed in by the user when he creates a bookmark. The 'Goto Bookmark' dialog box then presents a listbox or combobox in which one of the strings can be selected.

To be able to add this feature to the 'Bookmarks' sample program presented in the previous chapter you need a mechanism for storing and retrieving an arbitrary number of text strings. The functions for this are contained in the module LIST.BAS, which is part of the FIELD3 sample program.

The list module maintains a dynamic array of information structures with one element for each bookmark. The array is defined as being local to the module, so other modules can only access it via function calls. In this way, changes in the internal list structure do not require a revision of the whole program. The list is defined as follows:

```
Type FIELD_INFO
    Index As Integer
    Text As String
End Type
Dim FieldInfo() As FIELD_INFO
Dim NoOfItems As Integer
```

In the piece of code printed above, FieldInfo is the name of the array which stores the information, while NoOfItems is the number of items which are currently stored in this array.

The module contains functions to add, search and delete items and instructions to return the number of items which are currently part of the list. Please refer to the source code for a complete list and description of the functions and their parameters.

The Insert Bookmark menu item in this version of the program creates a dialog box where the user can enter a label for the bookmark. When the OK button is clicked, the following code is executed:

```
Private Sub btnOK_Click()
    If Text1 <> "" Then
        If Form1.VisualWriter1.SelLength = 0 Then
            Form1.VisualWriter1.SelLength = 1
        End If
        Form1.VisualWriter1.FieldInsert ""
        AddString (Text1.Text), _
            (Form1!VisualWriter1.FieldCurrent)
        SendKeys "{LEFT}"
    End If
    Unload Me
End Sub
```

First, a Marked Text Field is created at the current caret position. Second, the text which has been typed in by the user is stored together with the field number. The call to SendKeys is required to return the caret to where it was before the field insertion.

The Goto Bookmark dialog box contains a combo box which lists all of the bookmarks which have been created so far. The combo box is filled with the bookmark titles when its form is loaded:

```
Private Sub Form_Load()
    Dim Count As Integer
    For Count = 0 To FieldInfoEntries() - 1
        cboBookmark.AddItem GetFieldInfoText(Count)
    Next Count
End Sub
```

When the OK button is clicked, the bookmark list is searched for the string which has been selected in the combo box, and the corresponding Marked Text Field is selected.

```
Private Sub cmdOk_Click()
```

```
    Dim Field As Integer
    Field = GetFieldInfoIndex((cboBookmark.Text))
    If Field <> -1 Then
        Form1.VisualWriter1.FieldCurrent = Field
        Form1.VisualWriter1.SelStart = _
            Form1.VisualWriter1.FieldStart - 1
        Form1.VisualWriter1.SelLength = _
            Form1.VisualWriter1.FieldEnd - _
            Form1.VisualWriter1.FieldStart + 1
    End If
    Unload Me
End Sub
```

Finally, you need a way to remove a bookmark. This is required, for instance, when part of a text which contains a bookmark is completely deleted. The Marked Text Field is then automatically deleted by VisualWriter. The text string which contains the bookmark title must, however, be deleted separately. This can be done in the Event procedure of the FieldDeleted Event by calling the DeleteString function of the list management module:

```
Private Sub VisualWriter1_FieldDeleted(ByVal FieldIndex _
As Integer)
    DeleteString (FieldIndex)
End Sub
```

You can also extend the sample program with a dialog box, similar to the Go To Bookmark... dialog, in which a bookmark can be deleted without deleting the text. Besides calling the DeleteString function, this would require converting the Marked Text Field to normal text. Use the FieldDelete Method to achieve this.

A Word Processor

The program is based upon the MDI sample from the Visual Basic Programmer's Guide, with the TextBox controls replaced by VisualWriters. If you are not familiar with MDI, control arrays or creating a toolbar you may want to read that section first.

The source code for this example is contained in the MDIDEMO and COMMON sample source directories.

Adding a PageSetup Dialog Box

The Page Setup dialog box is used to determine the page size and print margins. The maximum page size is resticted by the capabilities of the default printer. For implementation details, look at the source code of the PAGEDLG form.

A Print Dialog Box

When the 'Print...' menu item is clicked, first a Common Dialog box is shown to let the user enter the range of pages, number of copies and printer specific information. The rest of the procedure, which is part of the MDIChild form, is just a loop which for every page to be printed sets the appropriate VisualWriter properties.

Search and Replace

Searching and replacing is entirely done in VisualWriter. You just have to assign a value of 1 for Search or 2 for Search And Replace to the FindReplace Method. VisualWriter then opens the Windows Common Dialog box.

Dialog Boxes for Text and Background Color

This is also done with Common Dialogs. The color value returned from the dialog box is assigned to the ForeColor or BackColor properties.

Using Paragraph Frames

With VisualWriter, you can add lines and frames to a paragraph or a range of paragraphs. For instance, you can put a line on top of a caption like in the property reference of this manual. Or you can create tables by using the 'tab lines' feature which draws a vertical line at every tab stop.

The dialog box for paragraph frames is not included in the text control, but the source code is included in the MDI sample.

The Propeties which are responsible for paragraph frames are FrameDistance, FrameLineWidth, and FrameStyle.

Using VisualWriter as a Bound Control

If you are not familiar with the Data Control or with Bound Controls in general please refer to the Visual Basic documentation.

The source code for this example is contained in the DATA sample source directory.

Connecting VisualWriter to a Data Control enables you to store the contents of VisualWriter as a record in a database. Not only is the plain text stored, but also all formatting information, e.g. font and paragraph attibutes, colors, and image file names. The data is stored in a binary format which is the same as that used by the Load and Save methods.

The Data sample program is connected to a small database which contains descriptions of some of VisualWriter's properties. The data base was created with the Visual Basic Data Manager and then filled by inserting text from the clipboard. You can browse through the records of the data base by clicking the Data Control buttons on the lower left side of the window. If you change something in the current record, the changes will automatically be written to the database as soon as you click on one of the buttons.

Storing VisualWriters contents with all formatting information like in this example requires the DataFormat property to be set to 1 - Binary. In the default mode, which is 0 - Text, only the text is stored. The 0 - Text mode can be used to access databases which have been created by other programs that do not use the VisualWriter data format.

Headers and Footers

This example shows you how to print a header on top of each page.

The source code for this example is contained in the HEADERS sample source directory.

The sample program consists of a form with two text controls on it. The one which is named VisualWriter1 contains the normal text. VisualWriter2 is only used for printing the header and is not visible to the user.

To make the second text control invisible at run time, it is simply moved out of the visible area when the form is loaded. The position where the header text is to be printed is determined at run time by setting the PageMarginL and PageMarginT properties.

```
Sub Form_Load ()
    ' Move the header control out of the visible area so
    ' it does not pop up when its contents are changed.
    VisualWriter2.Top = -10000
    VisualWriter1.PageWidth = A4WidthInTwips
    VisualWriter1.PageHeight = A4LengthInTwips
    VisualWriter1.ScrollBars = 3
End Sub
```

The following procedure does the printing:

```
Sub mnuPrint_Click ()
    Dim Copy, CurPage, StartPage, EndPage As Integer

    StartPage = 1
    EndPage = VisualWriter1.CurrentPages

    On Error Resume Next

    ' Initialize and call the common print dialog.
    CMDialog1.Copies = 1
    CMDialog1.FromPage = 1
    CMDialog1.ToPage = EndPage
    CMDialog1.Min = 1
    CMDialog1.Max = EndPage
    CMDialog1.Flags = PD_HIDEPRINTTOFILE Or PD_NOSELECTION
    CMDialog1.CancelError = True
    CMDialog1.ShowPrinter
    If Err Then Exit Sub

    ' The user sets the first and last page.
    If CMDialog1.Flags And PD_PAGENUMS Then
        StartPage = CMDialog1.FromPage
        EndPage = CMDialog1.ToPage
    End If

    ' Print the pages. Put header 1 inch from the left upper
    ' corner of the page. Put standard text just below the
    ' header.
    VisualWriter2.PageMarginT = 1440
    VisualWriter2.PageMarginL = 1440
    VisualWriter1.PageMarginT = 1440 + VisualWriter2.Height
    VisualWriter1.PageMarginL = 1440
    VisualWriter2.Width = VisualWriter1.PageWidth - _
        VisualWriter1.PageMarginL - _
        VisualWriter1.PageMarginR
    Printer.Print
    For Copy = 1 To CMDialog1.Copies
        For CurPage = StartPage To EndPage
            ' Print header
            VisualWriter2.Text = _
                "This is the header of page " & Str$(CurPage)
            VisualWriter2.FrameStyle = BF_BOTTOMLINE
            VisualWriter2.ZOrder 1
            VisualWriter2.PrintDevice = Printer.hDC
            VisualWriter2.PrintPage 1
            ' Print page
            VisualWriter1.PrintDevice = Printer.hDC
            VisualWriter1.PrintPage CurPage
            Printer.NewPage
        Next CurPage
    Next Copy
    Printer.EndDoc

End Sub
```

Adding a Spell Checking Tool

VisualWriter has no built-in spell checker, but can be used with the VisualSpeller tool from Visual Components, Inc. Having installed VisualSpeller, all you have to do is to start it with just one line of Basic code which calls VisualWriter's VTSpellCheck Method:

```
VisualWriter1.VTSpellCheck
```

It is not necessary to put a VisualSpeller icon on the form or to add it to your project.

You can start the spellchecker, for instance, from a toolbar button or from a menu item. The spellchecking process is handled entirely by VisualSpeller's built-in dialog boxes.

The VTSpellDictionary property enables you to specify a different dictionary for the spellchecker. Dictionaries can be created and edited with a tool which is part of the VisualSpeller package.

VisualWriter Objects

The VisualWriter objects interface allows you to embed objects in the text. Objects can be Image-Controls or any other control which has a window handle, i.e., a *hWnd* property.You can insert an object as a character so that it moves with the text as the text changes, or you can insert it at a fixed position and let the text flow around it.

For the built-in Image-Control, VisualWriter also takes care of loading, saving, printing, and adjusting the zoom factor. For other objects, this must be done within the application.

For an example of how to use the Object properties and methods, refer to the MDI sample program.

Mail Merge

Using VisualWriter as a Bound Control showed you how to store VisualWriters entire contents in a database field. For implementing functions like mail merge, however, the requirements are different: the contents of database fields have to be inserted at specified positions in a previously prepared document. The following sample program provides you with the basis of how to this.

The code shown here is contained in the \STDLET sample source directory.

The Sample Program

The program consists of two forms, Form1 for creating a text and Form2 for connecting it to the database.

Start the program and use the 'File / Open...' command to load the sample file ACCOUNT.TX The file contains three fields which are to be replaced by database entries. Select Insert / Data to access Form2. When you click the Insert button in Form2, the contents of the three database fields are copied to the text fields in Form1. You can select a different record by clicking one of the data control buttons in Form2, and then clicking Insert again to replace the fields.

How it Works

Each of the 3 edit controls in Form2 are connected to a field in the database. The edit controls are used as bound controls, so when you browse through the database by clicking on the data control buttons, the contents of the selected database record are automatically copied to the edit controls. The only thing left to do is to copy the data from the edit controls to the text fields in the document. This is done when you click on the Insert button:

```
Private Sub cmdInsert_Click()
    Form1.VisualWriter1.FieldCurrent = 1
    Form1.VisualWriter1.FieldText = Form2.Text1
    Form1.VisualWriter1.FieldCurrent = 2
    Form1.VisualWriter1.FieldText = Form2.Text2
    Form1.VisualWriter1.FieldCurrent = 3
    Form1.VisualWriter1.FieldText = Form2.Text3
    ' Uncomment this to send the result to the printer.
    ' Printer.Print
    ' Form1.VisualWriter1.PrintDevice = Printer.hDC
    ' Form1.VisualWriter1.PrintPage 1
    ' Printer.EndDoc
End Sub
```

To implement a real mail merge function you will have to add a dialog box in which the user can select the database to be used. You may also want to provide a variable number of database fields which are dependent on the contents of the selected database.

Using VisualWriter in Visual C++

VisualWriter can be used as an OCX with several Windows-based development environments. This chapter highlights procedures required to use VisualWriter as an OCX with the Microsoft Visual C++ environment.

Creating Applications in Visual C++

Before using VisualWriter with Visual C++, you should read the Microsoft Visual C++ 4.0 documentation and on-line help.

Creating a Dialog, CFormView, or CView Based OCX Application

1. Start Visual C++.
2. From the File menu, choose New. The New dialog box appears
3. In the New box, select Project Workspace and click OK.
4. The New Project Workspace dialog appears.
5. Browse to the desired directory path.
6. In the Name text box, type a name for your project.   This will create a sub-directory of that name in the current path.
7. From the Type list, select MFC AppWizard(exe) to create a project based on the MFC library.
8. Click the Create button.

The MFC AppWizard - Step 1 Dialog appears.

- If you wish to create a Dialog based application, click the Dialog radio button, click NEXT and procede to the section, Dialog Based Applications.
- If you wish to create a CFormView based application, click the "Single Document" or "Multiple Documents" radio button, click NEXT and procede to the section, CFormView Based Application.
- If you wish to create a CView based application, click the "Single Document" or "Multiple Documents" radio button, click NEXT and procede to the section, CView Based Applications.

Dialog Based Applications

1. In the Step 2 dialog, click on the OLE Controls check box to add built-in support for OCX products.

2. Click on NEXT button.

The Step 3 dialog will appear

3. In the Step 3 dialog, you can accept the default options by clicking the NEXT button.

4. In Step 4, you can accept the default options by clicking the FINISH button. VC++ will build your project.

The New Project Information dialog will appear.

5. Click OK

CFormView Based Applications

1. In the Step 2 dialog you can accept the default options by clicking the NEXT button.

2. In the Step 3 dialog, click on the OLE Controls check box to add built-in support for OCX products.

3. Click on Next button.

4. In the Step 4 and 5 dialogs you can accept the default options by clicking the NEXT button.

5. In the Step 6 dialog, select the class view name from the class list at the top of the dialog.

CView will appear in the Base Class listbox.

6. In the Base Class listbox, change CView to CFormView.

7. Then click on the FINISH button to have VC++ build your project.

CView Based Applications

1. In the Step 2 dialog you can accept the default options by clicking the NEXT button.
2. In the Step 3 dialog, click on the OLE Controls check box to add built-in support for OCX products.
3. Click on Next button.
4. In the Step 4 and 5 dialogs you can accept the default options by clicking the NEXT button.
5. In the Step 6 dialog, click on the FINISH button to have VC++ build your project.

Adding the VisualWriter Component to your Project

To insert a VisualWriter component into your project:

1. From the Insert menu, choose Components.

The Component Gallery dialog box appears.

2. Select the OLE Controls tab.

3. If the VisualWriter Text Control icon is not visible in the Gallery, click Customize to add the control.

4. Select the control from the Component list on the right and click OK.

This returns you to the Component Gallery.

5. Select the Text Control icon in the Gallery and click Insert.

The Confirm Classes dialog will display.

6. Click OK to confirm and exit the dialog.

7. Repeat steps 5 and 6 for the Status Bar, Ruler Bar, and Button Bar controls.

8. Click Close to exit the Component Gallery.

The Text Control and its tools should now appear in the Control palette.

When VC++ adds components to your project, it creates CPP and H source files defining the class, properties, and methods for the control. It is a good idea to take a look at these files to understand what they contain. Methods and properties are not accessed the same in C++ as they are in many other languages like Visual Basic. When these files are generated, VC++ creates both a Get and Set function for most methods and properties. VisualWriter, for example, has a Text property. VC++ will create both a GetText and SetText member functions.

Adding the Component to your Dialog or CFormView:

1. In the Resource Editor, bring up the dialog that you want to place VisualWriter into.
2. Click on the VisualWriter component in the Editor's Control palette.
3. Draw the component on the dialog box.
4. Now this can be placed and sized as desired using the handles around the control.
5. Click on the right mouse button to bring up a floating menu.   The design-time properties for the control can be viewed and modified through this menu.

Assigning Member Variables

Once you have added the text control to the dialog, it will be necessary to assign a member variable to each control to gain access to the methods and properties at runtime.

Assigning member variables:

1. From the View menu, choose ClassWizard.
2. Select the Member Variables tab.
3. Select the control in the Control ID window for which you wish to add a variable and click the Add Variable button.

   The Add Member Variable dialog will display.
4. Type in the member variable name e.g. something like m_vwctrl.   Accept the default variable category and type, by clicking OK.
5. The MFC ClassWizard dialog will display the variable you added in the Control ID window.
6. Repeat steps 3 and 4 for each of the VisualWriter controls, specifying a new name for each.
7. Once you have added all the variables, click on OK in the MFC ClassWizard dialog to return to your project.

Adding the VisualWriter Component to your CView:

1. In the file list, bring up the header file for the view (<projname>view.h).
2. At the top of the file, include each of the VisualWriter control header files:

```
#include "tx4ole.h"

#include "txbbar.h"

#include "txruler.h"

#include "txsbar.h"
```

3. In the Attributes section, as a public member, add the following to create member variables for each of the controls in your view:

```
CTX4OLE m_vwctrl;

CTXBBAR m_vwbbar;

CTXRULER m_vwruler;

CTXSBAR m_vwsbar;
```

4. Now through the file list, bring up the C++ source file for the view (<projname>view.cpp).
5. Start the ClassWizard.   Make sure the view class is selected as the Class Name.
6. Select the View object in the Object Id listbox.
7. Select the "Create" message in the Messages listbox.

The Create handler will initially come up with the following code:

```
return CWnd::Create(lpszClassName, lpszWindowName, dwStyle, rect,
pParentWnd, nID, pContext);
```

Change this to the following:

```
if (CWnd::Create(lpszClassName, lpszWindowName, dwStyle, rect, pParentWnd,
nID, pContext) == 0)

   return 0;


if (m_vwctrl.Create("VisualWriter", dwStyle, rect, this, 1000) == 0)

   return 0;

if (m_vwbbar.Create("VisualWriter ButtonBar", dwStyle, rect, this, 1001) ==
0)

          return 0;

if (m_vwruler.Create("VisualWriter Ruler", dwStyle, rect, this, 1002) == 0)

   return 0;

if (m_vwsbar.Create("VisualWriter StatusBar", dwStyle, rect, this, 1003) ==
0)
   return 0;


return TRUE;
```

8. Start the ClassWizard. Select view class as the Class Name.
9. Select the View object in the Object Id listbox.
10 . Select the "WM_SIZE" message in the Messages listbox.
11. Click on the Add Function button to create the OnSize handler function for this message.
12. Add the following code to the handler:

```
// TODO: Add your message handler code here
if (m_vwctrl && m_vwbbar && m_vwruler && m_vwsbar) {
   m_vwctrl.MoveWindow(0, 60, cx, cy-(25+60));
   m_vwbbar.MoveWindow(0, 0, cx, 30);
   m_vwruler.MoveWindow(0, 30, cx, 30);
   m_vwsbar.MoveWindow(0, cy-25, cx, 25);
   }
```

Connecting the VisualWriter Controls

Connecting the Controls:

1. In the Create handler, add the following code:

```
m_vwctrl.SetButtonBarHandle(m_vwbbar.GetHWnd();

m_vwctrl.SetRulerHandle(m_vwruler.GetHWnd();

m_vwctrl.SetStatusBarHandle(m_vwsbar.GetHWnd();
```

Handling Events in your Dialog or CFormView:

Assigning Message Handlers:

1. Start ClassWizard

2. In the Class Name listbox, select the Dialog or CFormView class that was created.

3. In the Messages listbox, select the desired message to handle and click on Add Function button to add a handler for this. For our example, select the "Click" event and click on the Add Function button to add the handler for this.

4. Click on the Edit Code button to edit the new function.

5. Add the following code in the function:

```
    MessageBox ("Click Event","You clicked on the document");
```

6. Run the program and when the document is clicked on, the message "You click on the document".

Setting Properties in Visual C++

You can easily set specific properties for each of the controls you include in your project.

To set properties for a control:

1. Double-click on the control in your project that you wish to set properties for.

   The Control Properties dialog will display.

2. Select the appropriate tab for the property settings you wish to modify.

   Properties are grouped together in categories, such as paragraphs, fonts, and pages.

3. Modify the property settings as needed. For more information on each property, see VisualWriter Properties, Events, and Methods.

4. Once you have set the properties for the active control, close the Control Properties dialog to return to your project.

5. Repeat steps 1 through 4 for each control.

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX

Welcome to VisualWriter OCX