ALLTEXT™ 4
(Standard and HT/Pro)

Bennet-Tec Information Systems, Inc

ALLTEXT™ 4

Created by:
Jeffrey W. Bennett,
Victor A. Chernov,
Vladislav V. Golovkov,
Alexander E. Nikiforov,
Pavel V. Illich-Switych.

Published and Distributed by:
Bennet-Tec Information Systems, Inc.
10 Steuben Drive, Jericho, NY 11753 -USA
Controls@Bennet-Tec.Com
Phone (516) 433-6283 Fax (516) 822-2679

Services and Products available from Bennet-Tec

Bennet-Tec Information Systems, Inc. is a leading developer of software components and a provider of custom software development services.

Custom Controls by Bennet-Tec:

	**ALLText ™**	Multi-Font Text Control, Also **ALLText HT/Pro** - Hypertext Enhanced Edition
	**TList™**	Enhanced Outline control (multifont, multipicture, drag-drop, restructuring, and much more.)
	MetaDraw™	Object Oriented Graphic Image and HyperGraphic Control.
	**FileIcon™**	Icon Extraction, Display & Launch control for Visual Basic.
	**PicScroll™**	Scrollable, Zoomable, DropFile Aware Picture Box control for Visual Basic.
	**VBX Artist™**	Image Editing Control For Visual Basic
	**ScatterPlot3d ™**	Three-D Scatter Plot control for Visual Basic
	**Global ™**	InterApplication variable sharing for Windows

Custom software development services:

As a Microsoft Solution Provider, Bennet-Tec is pleased to be able to offer full custom software development services for the Microsoft Windows environment.

Services include:

- OCX/VBX Customization
- DLL/Component Creation
- Complete Windows application Development

Past projects include:

- Electronic Mail, Web Browsers, Electronic Exams
- Financial Trading and Order Entry Systems
- CAD, Games

Technology Licensing:

Bennet-Tec has unique technology available for licensing in areas of

Internet / HTML

Embedded Word Processing - for any Windows application

Forms processing

Document and image annotation

Our aim is to make you look sharp!

ALLText

Licence agreement: Use and Distribution.

This is a legal agreement between you (the purchaser of the ALLTEXT) and Bennet-Tec.

0. **Ownership**: The ALLText control is owned by Jeffrey W. Bennett and Bennet-Tec Information Systems, Inc. and is protected by US copyright laws and international treaty provisions. Neither the software nor the documentation may be copied for distribution or resale without express permission from Bennet-Tec, except as stipulated below (see Distribution of RunTime Software).
1. **Grant of License**. This license agreement permits you, a single individual, to use the enclosed custom control "ATX40S.VBX" or ATX40H.VBX in creating applications or running non-compiled applications within a design time environment such as Visual Basic. With respect to use in a networked environment, this means that only one copy of the software may be running at one time for each license you have purchased. A separate copy of the software must be purchased for each developer using the software in a design time environment. **The License file may not be shared or distributed.**
2. **Distribution of RunTime software**: In order to distribute applications created with the ALLText control, you will need to provide your users with a run time VBX. The file ATX40S.VBX or ATX40H.VBX may be distributed with your applications for this purpose. You may distribute this file without any additional fees or royalties. You may not distribute the help file ATX4.HLP, or the design-time support file ATX4S.LIC or ATX4H.LIC under any circumstances - these files are for use solely by the purchaser of the licence from Bennet-Tec. In distributing your code, you should copy the file ATX40S.VBX or ATX40H.VBX onto your distribution disks and upon installation of your application, copy the file into the client's windows/system directory. If referred to in your documentation (on-line files or hard copy), you should note the proper copyright information for the ALLTEXT control.
3. **Limited Warranty:** Bennet-Tec warrants that the software (ALLTEXT) will perform as advertised and as provided for in the documentation for a period of ninety (90) days from the date of receipt. Should the software (ALLTEXT) fail to perform as advertised within such a period, Bennet-Tec will make every reasonable effort to satisfy any such claims, alternatively the customer may return the software (original disks and documentation) within this period with proof of purchase for a full refund of the purchase price (not to include shipping and handling charges).
4. **Other Warranties:** To the maximum extent permitted by law, Bennet-Tec disclaims all other warranties, expressed or implied.
5. **No Liability for Consequential Damages:** To the maximum extent permitted by law, Bennet-Tec will refuse to accept any liability for damages whatsoever arising out of the use or inability to use this product (ALLTEXT), even if Bennet-Tec has been advised of the possibility of such damages.
6. **Contact Information:** Should you need to discuss this agreement, or have any questions concerning the product (ALLTEXT) itself, please contact Bennet-Tec at phone (516) 997-5596, Fax (516) 822-2679, or by Electronic Mail at Controls@Bennet-Tec.Com. Our address is 50 Jericho Tpk, Jericho, NY 11753

Getting Started

This section is intended to get you started quickly using ALLText. More detailed information is available on specific programming techniques and properties in the relevant sections of this manual. Note too that you may feel free to start off with our demonstration application (included on your disk) and modify any of the code to meet your needs.

We hope that this introduction to ALLText proves helpful to you. If you have any suggestions as to how to make this section (or any other section) of the manual more useful, please let us know.

Adding the ALLText Control to a Form

To begin using the ALLText control, make sure you have the appropriate files on your hard disk and all in the same directory:

ALLText Standard Files: ATX40S.VBX, ATX4S.LIC and ATX4.HLP.

ALLText HT/Pro Files: ATX40H.VBX, ATX4H.LIC and ATX4.HLP

The preferable location for these files in design mode is your \Windows directory. Alternatively you may locate them in any other directory. The windows\system directory is NOT a preferred choice as this directory is generally used by end-user applications which may overwrite your design version of ALLText with an older version of the VBX.

Now open a new VB Project and add an ALLText control to your form using the File Add menu choice to select ATX40S.VBX (standard edition) or ATX40H.VBX (HT/Pro edition). The ALLText icon should then appear on your toolbar. You may select the control and place on your form as you would any other control.

After adding the control to your form, you may set the various default properties. Note that the text and text characteristic properties may not be accessed at design time, this may change in some future version of the control, but for now it is required that text manipulation be handled at runtime.

Constants

All constants and file declarations used by ALLText and referred to in this manual are shipped with the control in a constant declaration file. You should add the file ATX4SAPI.BAS (standard edition) or ATX4HAPI.BAS (HT/Pro edition) to you Visual Basic application.

Definitions

Paragraph - Much of ALLTexts internal functioning is based on manipulation of paragraph sized units. A paragraph is generally defined as a unit of text terminated by a hard line break - either a carriage return/ linefeed {chr$(13) & Chr$(10)} in terms of Raw Text, or the embedded code \par in terms of formatted text. Paragraphs are characterized by Alignment, Margins, and Text content.

Font Table - An internal table of fonts in use is maintained by ALLText to allow quick manipulations of character formatting.

Point - A unit of measure equal to 1/72 of an inch, applicable to font sizes.

Twips - The standard, device independent, unit of measure used by Windows applications. A twip is equivalent to 1/1440 inches.

Pixel - A hardware dependent unit of measure equal in size to the smallest addressable element for a given physical device.

How To Move Code from the Visual Basic Textbox to ALLText

With no additional coding, simply including an ALLText window on your VB form will provide you with a text box much like the standard text box. There are some basic concepts however which may be supported differently by ALLText, and a few features which are not supported at all by ALLText. These are described below.

1. **Text Formatting and Default Fonts**.

The most basic difference between ALLText and the VB textbox is with regard to formatting and related properties. ALLText was designed to support mixed fonts within a single control. Unlike the standard text box, most formatting properties associated with ALLText produce their effect on the current cursor location or select region, not the entire text box.

Text is first selected, or the caret positioned at an insertion site and then the FontName property is set or read. With no selection active, only the current cursor and text immediately entered at that location is affected.

In order to modify the default font, use the ALLText FontTableGet and FontTablePut functions.

```
Sub Form_Load ()
    'Sets up default font - font table entry 0
    'First read old font table 0 parameters
    st% = FontTableGet(ALLText, 0, fam%, CharSet%, FntName$, FntSize%, FntWidth%,
                       Bold%, Italic%, Underline%, StrikeOut%, shadow%, SubSup%)
    FntNam$ = "Times New Roman"
    'Now reset font name and size parameters
    st% = FontTablePut(ALLText, 0, 0, 0, FntNam$, 30, 0, 0, 0, 0, 0, 0, 0)
End Sub
```

2. **MultiLine**: ALLText always operates in a multi-line mode and does not support the *MultiLine* property. It is possible however to simulate this in some instances by setting the *DocWidth* to a large value. So long as there are no carriage return/line feed sequences (ie: the end-user doesn't hit enter), line wrapping may appear to be turned off.

3. **DDE**: ALLText does not support DDE nor any of the DDE properties: *LinkItem, LinkMode, LinkTopic, or TimeOut.* We do support use of the clipboard for data exchange however. And ALLText HT/Pro supports the embedding of OLE objects.

4. **HideSelection**: ALLText does not support the *HideSelection* property. We do however provide a mechanism through the *WriteProtect* property which may be called upon to hide the visible highlighting of a select region.

5 .**MaxLength**: ALLText does not support the *MaxLength* property. We do provide a *TextLength* property however which could be used within the KeyPress event to prevent entering text when the document is over a certain length.

6. **Scrollbars**: ALLText does not support the standard *ScrollBars* property. ALLText does however provide it's own more flexible support for selecting the display mode of scrollbars through the *ScrollBarV* and *ScrollBarH* properties

7. **Password**: ALLText does not support the *Password* property

8. **Parent**: ALLText does not support the *Parent* property

How to Add Text or Load a Document

The first thing you will probably want to do with ALLText is to add and display some text with the ALLText control. There are several methods of going about this.

Using Text and FText Properties

Using SelText and SelFText Properties

Pasting Text from the Clipboard

Loading Text from a File

Using Text and FText to Set the Content

Support for setting or reading the entire ALLText content is provided through the **Text** and **FText** properties. The Text property contains the complete text content of the control, without any formatting codes, up to the first 32k. You can set or read this property. Setting the Text property will replace the entire content of the textbox (even that beyond the first 32 k) with the string specified. After setting the Text property, the content of the control will be unformatted.

eg: ALLText.Text = This is some text

or ALLText.FText = One \cf1 word \cf0 here will be blue

The FText Property is similar to the Text property but includes embedded formatting codes like \cf1 for blue, or \par for begin a new paragraph. These embedded codes will be interpreted by ALLText resulting in the display of formatted text. The interpretation of formatted text by FText is controlled by the SelFType property.

Note that in cases where the majority of the text is not changing, it is more useful to add text using either *SelText* or *SelFText*.

Using SelText and SelFText to Insert Text

Text can be inserted at any location within an ALLText document using the **SelText** property as you would with a standard VB text box, or our own **SelFText** property. SelText reads and writes only the unformatted text content (no embedded codes), while SelFText includes and interprets embedded formatting codes. Note that when you set SelText or SelFText, the new text replaces any currently selected text, or is inserted at the current cursor position if no text was previously selected. If you are going to be adding text to the textbox in bits, using SelText or SelText will be much faster and produce better results than constantly resetting the Text or FText properties. Note that you can position the cursor with the standard SelStart property. More information on positioning the cursor and selecting text is presented below **(see How To Control the Cursor Position and Select Text)**

```
eg: ALLText.SelStart = ALLText.TextLength          ' set insert position to end of document
    ALLText.SelText = Here we paste some text
    ALLText.SelStart= ALLText.SelSTart + ALLText.SelLength      ' move insert position
    ALLText.SelFText = Here we paste some \cf1 blue text
```

The interpretation of formatted text by SelFText is controlled by the SelFType property.

Pasting Text from the Clipboard

ALLText provides full Clipboard support. You can paste from the clipboard programmatically or using the keyboard at runtime. ALLText supports both the <*Shift*> <*Insert*> and <*Cntrl*> V keyboard style combinations for pasting.

To copy RTF from the clipboard programmatically, use the .*ClipboardAction* property. Setting to 1 cuts selected text from ALLText to the clipboard, to 2 copies selected text to the clipboard and 3 pastes selected text from the clipboard. Do NOT use VBs ClipBoard object as the text held by the Clipboard object is NOT formatted.

```
ALLText.ClipboardAction = ATX_Paste   '= 3 see constants file
```

All font, color and paragraph formatting will be preserved when copying from applications which support RTF Clipboard operations (Word, WordPerfect, AmiPro).

NOTE: In order to properly support cut and paste of embedded Pictures (Supported under HT/Pro edition and Pen editions only) within an RTF document, you must have the ATXPIC.DLL present and REGISTERED To paste a BMP or File from the clipboard see our PastePic demo application in the Samples directory.

```
DECLARATION:      Declare Function ATX_REGNEWEXTERN Lib "atx40h"
                                  (ByVal noMsg%, ByVal lpstrModule$) As Integer
In Form_Load:     Ret_code = ATX_REGNEWEXTERN(1, "ATXPic.dll")
```

Loading from a File

ALLText makes loading a document simple. You can load a Text, RTF or ATX formatted file using the ALLText I/O stream property, *FileLoad*. Additionally *FileSave* may be used to save data to a file.

First set the *DataType* property to specify the file format: 0 for Raw Text, 1 for ATX_S (does not include font table and is useful only when hardcoding the font table), 2 for ATX_F (includes font table), or 3 for RTF level 1 (HT/Pro edition only) formatted file. Set the FileName property, including the full path to the desired file name. Now, Initiate file loading by setting the FileLoad property to 1 This indicates that ALLText should load the complete file without need of further programmatic intervention. For Example:

```
ALLText1.FileName = "AutoExec.BAT"                ' Set the file name
ALLText1.DataType = ATX_FORMAT_TEXT ' see constant def file for constant values
ALLText1.FileLoad     = 1                                 ' Automatically load the file
```

or

```
ALLText1.FileName = "SomeDoc.RTF"                 ' Set the file name
ALLText1.DataType = ATX_FORMAT_RTF     ' see constant def file for constant values
ALLText1.FileLoad     = 1                                 ' Automatically load the file
```

Other settings of the FileLoad property invoke the ATXGet event to allow greater programmatic control. These are somewhat more involved and are discussed in the section dedicated to ALLText Low Level I/O.

Note that you may want to prevent end-user interaction during the load procedure by setting the *WriteProtect* property to True prior to the *FileLoad* statement, and returning to a value of False upon completion.

PLEASE NOTE - File I/O has side effects, ALLText internally prevents reading or writing of certain properties during I/O. Closing the ALLText window or an application during running of I/O may lead to errors.

How to Save a Document

The easiest way to save text from an ALLText control is to select the text you wish to save and then use the standard ALLText I/O stream property property: *FileSave*.

First set the *DataType* property to specify the desired file format: 0 for Raw Text, 1 for ATX_S (does not include font table and is useful only when hardcoding the font table), 2 for ATX_F (includes font table), or 3 for RTF level 1 (HT/Pro edition only) formatted file. Set the FileName property, including the full directory path to the desired file name. Select the desired text and then, initiate file saving by setting the FileSave property to 1. This indicates that ALLText should save the complete file without need of further programmatic intervention. (Other settings of the FileSave property invoke the ATXPut event to allow greater programmatic control . These are somewhat more involved and are discussed in the section dedicated to ALLText Low Level I/O.)

For Example:

```
FileName = "\FRED\ALBERT\ATX"                ' Set the file name
ALLText1.FileName = "\FRED\ALBERT\ATX"               ' Set the file name
ALLText1.DataType = ATX_FORMAT_RTF      ' see constant def file for constant values
....
ALLText1.SelStart =0 : ALLText.SelLength = ALLText.TextLength     ' select all the text
ALLText1.FileSave     = 1                                          ' Automatically save the file
```

Note that you may want to prevent end-user interaction during the save procedure by setting the *WriteProtect* property to True prior to setting FileSave, and returning to a value of False upon completion.

PLEASE NOTE - File I/O has side effects, ALLText internally prevents reading or writing of certain properties during I/O. Closing the ALLText window or an application during running of I/O may lead to errors.

How to Set the Default Font

Upon loading ALLText, the default font in use is the SYSTEM font of 10 pt size.

In order to modify the default font, use the FontTableGet and FontTablePut functions to read the current default font (entry 0) and modify it to the desired parameters. Any text currently formatted according to the default font will be changed to reflect the new default font definition.

```
Sub Form_Load ()
    'Sets up default font - font table entry 0 as Times New Roman, 30 pt.
    'First read old font table 0 parameters
    st% = FontTableGet(ALLText, 0, fam%, CharSet%, FntName$, FntSize%, FntWidth%,
                       Bold%, Italic%, Underline%, StrikeOut%, shadow%, SubSup%)
    FntNam$ = "Times New Roman"
    FntSize% = 30
    'Now reset font name and size parameters
    st% = FontTablePut(ALLText, 0, 0, 0, FntNam$, FntSize% , 0, 0, 0, 0, 0, 0, 0)
End Sub
```

How to Format Text - Fonts, Colors , Paragraph Formatting

The principle objective of ALLText is to support the display and manipulation of fully formatted, WYSIWYG, text. Towards this end, ALLText provides complete programmatic control over the content and formatting of documents. Full support is provided for both character and paragraph formatting - along these lines if Word can do it, ALLText can too (unfortunately we couldnt figure out how to use up 3.5 Megabytes in the process the way they do).

Programatically, just about everything is available via direct settings of font and paragraph format properties (such as *FontBold*, *LeftIndent*, *TabStep*, etc) with only a few necessary function calls required to compose the proper formatting for our more flexible format options (eg: borders and custom font shadowing). Alternatively, formatting may be managed through the manipulation of text with embedded strings. Either method may be used equally well, however the use of the property settings is generally prefered where practical; the resulting code will be easier to read, debug, and maintain. Finally, A Font Selection Dialog box is also provided for end-user access through the F2 Key (if enabled with the *F2On* property).

Using Formatting Properties

Keyboard Formatting

Formatting Text using Embedded Codes

Using Format Properties to Format Text

Property settings (*FontName*, *Alignment*, etc) in ALLText are associated with text within a select region or at the current cursor location (see Cursor Positioning and Text Selection). In general, reading the value of the various formatting properties will return the appropriate characteristics for the active select region or cursor location. A value of ATX_Undefined (-4096) or ATX_PAR_Undefined (-32000) will be returned when reading a property whose value varies within a currently selected region of text.

Setting a font property with no text currently selected prepares ALLText to treat the following typed text (or text entered at that location with the SelText property) as being of that characteristic (bold, italic, etc). If a select region is active, ALLText applies the font attributes to that selected text.

In addition to supporting Character formatting ALLText supports full control over paragraph formatting. Each paragraph may independently be set with respect to alignment, margins, tabs, and paragraph borders. Setting a paragraph formatting property reformats the current paragraph according to the new setting. If a select region is active ALLText applies the font attributes to any paragraphs which the select region overlaps.

Font Properties manipulating the font characteristics in this manner include:

FontBold	FontColor	FontFamily
FontHidden	FontItalic	FontName
FontShadow	FontSize	FontSubSup
FontUnder	FontWidth	FontVisible

Properties supporting the direct formatting of paragraphs include:

Alignment	Border (HT/Pro only)	BottomIndent
FirstLineIndent	LeftIndent	LineSpacing
RightIndent	TabStep	TopIndent

Note that as new text is entered at some particular location, it takes on the formatting characteristics of that location. This is generally the same as that of the immediately preceding character.

As an example, you may wish to underline a string or change its color to Blue. First select the string as you would with the standard textbox, set the FontUnder property to True and set the color to Blue. Then turn the selection off.

```
ALLText.Text = The ball is blue, isnt it?       ' Control now contains some unformatted text
ALLText.SelStart = 12: ALLText.SelLength =4     ' Select the word Blue
ALLText.FontColor = QBColor (1)                 ' Set the color to Blue.
ALLText.SelLength =1                            ' Reduce select region to include only the B
ALLText.FontUnder = TRUE                        ' Underline the letter B
ALLText.Select = FALSE                          ' Turn off Select region
```

FontColor, *Italics*, *StrikeThrough*, *FontName*, *FontSize*, *SupSubscripts*, etc are similarly set via properties. Simple underlining may be accomplished by setting the *FontUnder* property to True, while more complicated underlining (double, colored, dashed, etc) may be set in using other values as computed by the *Make_Underline* function. The same font properties may be read at any time to determine the formatting at the current cursor position or as contained within a select region.

> ***** IMPORTANT:** Upon loading ALLText, the default paragraph is Left Aligned with no indentation and the default font name is "System", with no font characteristics set (ie the text is not bold, italic, etc). Note that the System font looks exactly the same regardless of whether the Bold property is set True or False. If you have tried to create some Bold text and do not observe a change, check the FontName property for that text region, try changing to Arial or some other font and you should see the effect. The default font may be changed using the *FontTableGet* and

FontTablePut functions to change FontTable Entry 0. (See description of Font Table below).

Paragraph formatting is just as easy. Each paragraph may be independently formatted. Simply set the relevant property - *LeftIndent*, *RightIndent*, *FirstLineIndent*, *LineSpacing*, *TopIndent* and *BottomIndent* may all be set to some value in Twips, while *Alignment* may be set to an integer value representing:Left (0), Right (1), Centered(2), or Left/Right Justify (3) alignments.

```
ALLText.CurPar = 5              'position start of select region in paragraph 5
ALLText.SelToPar =8             'position end of select region on paragraph 8
ALLText.LeftIndent = 1440       ' 1 inch =1440 twips
ALLText.Alignment = ATX_CENTERED     ' see constants and declaration file
```

Keyboard Formatting

A Font Selection Dialog box is provided for end-user (run-time) access through the **F2** Key (if enabled with the *F2On* property set to TRUE).

ALLText does not assume, however, any keystroke interpretation for setting fonts to bold, italics etc, or for setting paragraph formatting. If you wish, you may trap control key sequences in a *Key...Event* routine. Alternatively you may design a menu or button bar. For example

```
Sub MnuBold_Click()
' switches FontBold property for selected text or current cursor location
    ALLText1.FontBold = Not ALLText1.FontBold
    End Sub
```

or

```
Dim KeyPress_Exit    ' include this statement in your module level declarations
Sub ALLText_KeyDown (KeyCode As Integer, Shift As Integer)
Const CTRL_MASK = 2
KeyPress_Exit = FALSE
If Shift And CTRL_MASK Then
    Select Case KeyCode
      Case Asc("B"), Asc("b")
          ALLText.FontBold = Not ALLText.FontBold
              KeyCode = 0
          KeyPress_Exit = TRUE
      End Select
    End If
End Sub
  Sub ALLText_KeyPress (Key as Integer)
      'KeyPress +_Exit may be set by the keydown event
      If KeyPress_Exit  then Key =0
Exit sub
```

Formatting Text using Embedded Codes

(This section is for somewhat more agressive users and may be ignored by the less daring).

In addition to formatting text through use of the above described format properties, pre-formatted text strings with embedded format codes may be entered into the document directly using SelFText or FText. The characteristics of existing selected text may also parsed by reading SelFText or FText and manipulated by changing the string in SelFText after selecting some text region. In general we suggest using the format properties whenever possible - it's simpler to understand, simpler to debug if things go wrong, and less likely to result in problems (for example, errors due to mistyping an embedded formatting code which needs to be entered just right to have the desired effect). Still we document the embedded codes in an appendix to this manual, and offer full support for your programming enjoyment. *If you really want to have fun, try to create a file with embedded codes using an ASCII text editor (not the best way to go about it - Use our JED editor instead.)*

Embedded codes are available for just about everything in ALLText. Typically format codes start with a backslash and end with a terminating space. The terminating space is important - it lets the control know where the embedded code stops and the body text starts. Most of the codes have been borrowed from RTF. Others are of our own design and usually beggin with a \ATX string.

Most formatting codes are pretty straight forward. There are codes for paragraph formatting and codes for character formatting. A current listing of all formatting codes understood by ALLText is included in the file ATXCODES.WRI.

Paragraph formatting codes are generally inserted at the beginning of the formatted text string, but actually they can be included anywhere in the string and will be properly interpretted. When reading the formatted string back out however, they will be placed by ALLText at the beginning.

```
ALLText.SelFText = \par \li1440 \fi-360 This is \ql a line of text
```

In this example, we are inserting some text at the current cursor location (or replacing any previously selected text). The \par code says start a new paragraph (SelFText ignores carriage return/line feeds and relies instead upon the embedded \par code just as does RTF). The codes, \li1440 and \fi-360 , set up the paragraph as having a 1 inch (1440 twips) left indentation, and a first line indentation 1/4 inch (360 twips) in from the rest of the paragraph. In other words it is a **hanging indent.** The code \ql specifies that this paragraph should be left justified. This could equivalently be programmed as follows:

```
ALLText.SelText = Chr$(13) & Chr$(10) & This is a line of text
ALLText.LeftIndent = 1440
ALLText.FirstLineIndent = -360
ALLText.Alignment = 0     'left
```

As you can see, embedded codes save programming lines, but can be much more difficult to read and maintain.

Most embedded font formatting codes are equally straight forward. ALLText supports 16 font colors - the same as are specified by Visual Basics QBColor function. Color formatting codes begin with a backslash and the letters cf, and then are followed by a number indicating the desired color. As with paragraph formatting, the code is terminated by a space, chr$(32). The format is as follows \cfN where N is an integer from 1 to 16. An example of formatting with embedded colors follows below:

```
SelFText = \cf9 Fred \cf14 John \cf0 thats all
```

The codes \cf9 and \cf14 corresponding to blue and light yellow. The \cf" string is followed by a value referring to one of the 16 colors supported for ALLText fonts - the same color encoding is used as in QBColor -

This is equivalent to the following:

```
SelText =   Fred   : FontColor = QBColor(9)            ' light blue
SelStart = SelStart + SelLength
SeText = John   : FontColor = QBColor(14)              ' light yellow
```

```
SelStart = SelStart + SelLength
SelText = Thats all : FontColor = QBColor(0)        ' black
```

Font characteristic formatting, including everything but the font name, can be set by simple codes as well. \b-1 means the following will be bold, while \b0 means not bold (note that the difference is not observable for the default System font.). A simple underline may be created using \u-1 (Think of -1 as True and 0 as False). Note that for underlining, other values as created by the MakeUnderline function may be used to implement double underlines, colored underlines, dotted or dashed underlines.

FontNames are unfortunately not accessible by this mechanism. Instead one needs to refer to the entries in the **Font Table**. ALLText employs a FontTable mechanism internally to keep track of all fonts and font characteristics (name, size, bold, italics, etc). Think of this as a table of **font styles**. The font table architecture greatly speeds up the control's screen manipulations, and makes for a smoother interface when changing numerous font characteristics at one time. The FontTable also offers an efficient mechanism for file storage when saving in ATX format (compare to RTF).

Each entry in the Font Table defines a font style, a combination of font characteristics (Font Name, Size, Bold, Italic, etc - everything associated with the visible formatting of the character except for color, which is handled separately).

An embedded code may then be used used to format text simply by pointing to the appropriate style entry in the font table. For example "\f2 This will be formatted as per the definition of font 2. Here, the text is formatted according to font table entry 2. It's a whole lot faster and more efficient in memory than embedding a separate code for each independent font characteristic. In fact its alot easier than a string of format codes as well, compare \f2 with \b-1 \i0 \u2048

Upon loading the control, there is one font in the font table. This first entry, corresponding to the embedded code \f0 , is by default the System font without any special formatting characteristics (this can be changed with the *FontTablePut* function). Other font table entry definitions are created in sequential order automatically by ALLText whenever the manipulation or entry of new text requires a font which is not already defined in the font table (for example, if no text is already formatted according to the resulting combination of fonts when a property such as FontBold is set to True). Font Table entries may also be created programmatically through use of either the *FontIndex* property or the *FontTablePut* function.

The *FontTableSize* property may be read to determine the number of font entries in the table. *** NOTE:** Setting *FontTableSize* to a value of zero will remove all font definitions not currently in use from the table. Other settings have no effect.

FontTable Creation and Manipulation of Embedded Fonts

(This section is for even more agressive users and may certainly be ignored by the less daring).

ALLText uses a FONT TABLE to establish **font styles** - multiple font formatting characteristics (name, size, bold, italics, etc) applied together to a region of text. The embedded code used to format text is then simply a pointer to the appropriate entry in the font table.

In general ALLText handles the font table for you automatically. There are some instances however where you may wish to manipulate the font table manually.

If you are creating a set of read only documents or even editable documents where you know in advance all the fonts which may be used, you may wish to hard code those font table definitions into your code. This will save space when saving the documents. It will also make it very easy to format text with embedding fonts or to identify a font style (just read the *FontIndex* property) in response to a click event .

Like everything else with ALLText, there are several ways to control the FontTable. You can use the *FontIndex* property or the *FontTableGet* and *FontTablePut* Functions.

The FontIndex property can be read at any time to determine the font table entry of the current font. If Font properties have been set but do not yet apply to any text, reading the FontIndex property will add that font definition to the FontTable. (If the font properties have been applied to selected text, the font definitions have already been automatically added to the font table).

The FontTableGet function can also be used at any time and will return all character formatting properties associated with a given font table entry. The FontTablePut function can be used to define or modify character formatting properties for any font table entry.

Two examples are shown below:

EXAMPLE 1 - USING FONTINDEX TO CREATE A FONT TABLE

```
Dim FontNumberAsString(50)   ' a global array referencing font style entries
Sub FontTableInit ()
  ' -This subroutine will define a number of fonts for you, call upon form load
  ' -The indicies for these font table entries are storred in an array for easy use
   Static FontNumberAsString(5) As String
      For Index = 1 To 5
       'define font i characteristics
       ALLText.FontName = "Arial": ALLText.FontSize = 4 * Index
       ALLText.FontBold = -.5 + .5 * (-1) ^ Index
       fontIndexValue = ALLText.FontIndex          ' Reading FontIndex creates the font table entry
       'Now save embedded font code in array for easy reference and use
        FontNumberAsString(Index) = Mid$(Str$(fontIndexValue), 2)+" "
       Next Index
  End Sub
Sub Test_it_Out()
  '   Demonstrates use of embedded font codes
     For Index = 1 To 5
              Color$=Mid$(Str$(Index),2)+" "
              ALLText.SelFText = "\f" + FontNumberAsString(Index) + "\cf" + Color$
                       + " This text will now be size:" & Str$(4 * Index)   & "\par ": ALLText.Select = False
              Next Index
```

Note that the color of text is not contained in the font table definition. In the above example, the color is specified by the embedded string "\cf" + Color$ + " ". The string "\par " indicates termination of a paragraph. Other embedded strings may be used to control paragraph formatting, etc.

EXAMPLE 2 - USING FONTTabLEGET & ...PuT

to Modify the Default font & CREATE NEW FONT TABLE ENTRY

```
Sub Form_Load()
  ' -This subroutine redefines FontTable Entry 0 as Arial 12 pt,
  And creates another font table entry for Times New Roman 12 pt Bold.
  '-get details for 0th entry (the default font)
      Dummy = FontTableGet (ALLText, 0, family%, CharSet%, FntName$, FntSize%, FntWidth%, Bold%,
          Italic%, Underline%, StrikeOut%, Shadow%, SubSup%)
  '-change some parameters
 FntName$=Arial        :        FntSize%=12
 '-Replace entry 0 in font table
      Dummy = FontTablePut (ALLText, 0, family%, CharSet%, FntName$, FntSize%, FntWidth%, Bold%,
          Italic%, Underline%, StrikeOut%, Shadow%, SubSup%)
  '-Now create Entry 1
 FntName$ = Times New Roman    :    Bold%= TRUE
    Dummy = FontTablePut (ALLText, 1, family%, CharSet%, FntName$, FntSize%, FntWidth%, Bold%,
          Italic%, Underline%, StrikeOut%, Shadow%, SubSup%)
  End Sub
```

How to Add Paragraph Marks/ Line Breaks

Users should note that there are two types of paragraph delineations in use by ALLText.

When using the *Text* and *SelText* properties, to work with raw unformatted text , an end of paragraph marker consists of a two character Carriage Return/Line Feed (CrLf) string; ie, CHR$(13) followed by CHR$(10). Each of the two characters is counted when using *SelStart* and *SelLength*. The cursor however can not be positioned in between them however so there are certain SelStart values which just wont be accepted by ALLText. In this event SelStart will be reset to point at the first character after the paragraph break.

The formatted text properties, *FText* and *SelFText,* follow the RTF standard for paragraph delineation. FText and SelFText terminate paragraphs by concluding with the five character string "\par ". CrLf strings are totally ignored by ALLText when used to set FText or SelFText string. When reading these properties there may be extraneous CrLf strings within the text being read - these do not have any meaning - they are simply added to break up the string. This may seem strange, but this is based on the correspondence between ATX format and RTF formatting.

Note that due to the counting of CrLf as two characters by SelStart, one may have some confusion in decrementing through the content of ALLText. The following subroutine shows how this can be done properly using SelStart.

```
While ALLText.SelStart > 0
    If ALLText.CurChar = 0 Then
                ALLText.SelStart = ALLText.SelStart - 2
        ALLText.SelLength = 2
             Else
        ALLText.SelStart = ALLText.SelStart - 1
                ALLText.SelLength = 1
      End If
 Wend
```

Another alternative would be:

```
Sub Command1_Click ()
  While ALLText.SelStart > 0
        If ALLText.CurChar = 0 Then
          ALLText.curpar = ALLText.curpar-1
          ALLText.curchar = 32766 ' will reset to last char in paragraph
        Else
        ALLText.curchar = ALLText.curchar-1
      End If
     Wend
```

How to Control WordWrapping

ALLText automatically wraps text on screen to a width set by the *DocWidth* property (value in twips - 1440 twips/inch). You may want to set the DocWidth property to something like 85% of your control width.

```
Sub Form_Resize ()
    ALLText.Width=Form.ScaleWidth
    ALLText.DocWidth=.85 *ALLText.Width
    ...
```

Be aware that when reading back the DocWidth property, you may find it slightly different than the value it was set to. This is because when set, ALLText converts DocWidth to a value corresponding to a whole number of pixels on your particular system.

Note also that the DocWidth property may be change when loading in a document using FileLoad, or when the control content is changed as a result of repositioning a data control to which the control is bound. In this case ALLText will read in the width of the document as specified in the stored file. You should reset the DocWidth after reading or in the Data.Reposition event if you want to insure a specific Word Wrap width.

How to Prevent Editing

ALLText allows you to set a *WriteProtect* property to prevent end user updates to the screen in a text retrieval mode. You can even choose whether or not to allow end-users to copy information to the clipboard.

Setting the WriteProtect property to ATX_Protect_Changes (-1) allows the user to select and copy portions of text, but still prevents him from modifying the text while ATX_Protect_Keyboard (1) prevents the user from interacting with ALLText in any way other than through your menus or other controls. This may be useful if you have proprietary information you want viewed on the screen but not easily distributed to non-users. Setting WriteProtect to FALSE allows full end-user editing. You should include the constants definition file in your VB project to capture the appropriate constant definitions.

Note that there is one more setting of the WriteProtect property used to prevent screen updates. See the section below How to Prevent Screen Updates.

To prevent editing only specific portions of the document, tag that portion using HTags, NTags or PTags. You can then read the appropriate property during the KeyDown and KeyPress events and determine whether to throw away the keystrokes.

How to Prevent Screen Updates

When formatting text in a visible ALLText control, youll probably want to avoid having the end-user see all your manipulations - changing select regions, changing fonts, etc. Thats fine. ALLText allows you to enter into a special mode preventing screen updates until you are ready. Simply set the WriteProtect property to a value of *ATX_Protect_Screen* (make sure youve included the constant definition file in your project). Then make your changes to the text, and finally reset the WriteProtect property to ATX_PROTECT_NONE, ATX_PROTECT_KEYBOARD, or ATX_PROTECT_CHANGES in order to update the screen.

```
Sub MakeChanges( )
    wp%=ATX.WriteProtect
    ATX.WriteProtect=ATX_Protect_Screen
    '    < Make changes to content or formatting here >
    ALLText.Select=FALSE
    ATX.WriteProtect=wp%
End Sub
```

Note that the Protect_Screen setting prevents any refreshing of the ALLText control window. If another window is moved in front and then away, ALLText will not redraw itself until WriteProtect is Reset.

How To Print

ALLText offers a variety of flexible printing options, allowing end-user triggered printing from the keyboard <F3> Key, or programmatic support using the internal API functions.

The following ALLText API functions are designed for basic printing requirements. Only a single function call is required to print:

Print_ATextPages - prints specified pages

Print_AText - prints a specified paragraph range

The following ALLText API functions and events offer a great deal of flexibility but require more complex programming. See the section Using advanced printing functions.

ATX_Print_Start Function - initializes ALLTexts link to printing device & displays dialog box.

ATX_Print_Finish_ Function - terminates ALLTexts link to an output printing device

ATX_Print_Title Function - sets the text to be used in the printer progress dialog box.

ATX_Print Function - Prints a paragraph range, generating new pages as required.

ATX_Print_Region Function - Prints within a specified region on a single page.

PrintThisPage Event- This event is triggered for each page after calling the ATX_Print Function and before the ATX_PrintStartPage event

PrintStartPage Event -This event is triggered just before printing each page.

PrintEndPage Event - This event is triggered after printing each page.

The following properties are also useful in printing:

F3On	- Enables or Disables the F3 key for printing
ATX.DocWidth	- This is the word wrap width shown on screen. The same width is used when initiating printing with the F3 key.
ATX.PrinterDC	- Set to True in order to use the Printer Device Context for formatting the document on Screen.
ATX.PageHeight	- Specifies the height of a printed page - used when initiating -printing with the F3 key.

The following status codes may be returned by the printing functions:

CODE NAME		MEANING
ATX_PRINT_OK	0	Everything is OK.
ATX_PRINT_ERR_NODEFPRINTER	101	No default printer set.
ATX_PRINT_ERR_NOSUPPORTPRINTER	102	Printer not supported.
ATX_PRINT_ERR_INVALIDARGS	103	Invalid arguments set.
ATX_PRINT_ERR_NOROOM	104	Not enough memory.
ATX_PRINT_ERR_PRINTERROR	105	Printer error occured
ATX_PRINT_ERR_NOPRINTSTART	106	ATX_PRINT_START was not called.
ATX_PRINT_ERR_INVALIDSCALE	107	Values of the log_n% and rel_n% parameters are not equal to 1 in the ATX_PRINT_START function.

ATX_PRINT_ERR_INVALPAGAGRAPHRANGE	108	Invalid paragraph range was specified.
ATX_PRINT_ERR_REGIONTOOSMAL	109	Returned by ATX_Print_Region function when text is too small to fit in specified area.
ATX_PRINT_ERR_FATAL	113	Fatal error.

Using the F3 Key to Print the control content.

With the F3On property set to True, ALLText will interpret the F3 key as an instruction to send the complete content of the control to the default printer as defined by the Windows control panel settings

Hitting F3 may also be simulated using the SendKeys statement

ALLText.SetFocus

SendKeys {F3}, True

Note that the document will be printed without margins. The current DocWidth property value will be used to specify the location of line breaks (the same word wrap width as shown on screen). No events will be generated. The Back Picture will not be included in the output.

Using a single ALLText printing function to print to the default printer:

The following two functions are built into the ALLText control. They must be declared in your Visual Basic Program (see: ATX4hAPI.BAS or ATX4SAPI.BAS).

Print_ATextPages - prints specified pages to the default printer

Print_AText - prints a range of paragraphs to the default printer

While easy to use, these functions do not provide any control over the printer dialog box, and always force a page eject after printing each page.

Note that the DocWidth property setting does not affect the text wrapping width when printing using the built in printer functions. This is independently set by the parameters of the print functions.

Also note that unless the PrinterDC property is TRUE, even with the width settings exactly the same, due to differences in printer resolution the output may line break a word prior to, or after the break shown in the control window.

These functions may be called as follows:

```
i% = Print_ATextPages(CntrlName, pTop&, pBottom&, pLeft&, pWidth&, pPages$,
    pFlag%)
```

```
i% = Print_AText(CntrlName, pTop&, pBottom&, pLeft&, pWidth&, PrintDoc_from_par
    %, PrintDoc_to_par%, PrintDoc_Flag%)
```

The Parameters are:

CntrlName - the Name of the control as in the Name property.
pTop&, pBottom& - Top and bottom margins in Twips
pLeft&, pWidth& - Left Margin and Line Length in Twips.
PrintDoc_from_par%, PrintDoc_to_par% - Specified range of paragraphs
pPages$ - A string indicating which pages are to be printed. ALLText scans this string and prints pages corresponding to non-blank elements in the string. ie: if the string consists of two blanks followed by a character, another blank and two characters, ALLText will print pages 3,5 and 6.
pFlag% - set to 1 to include the background bitmap, 0 otherwise.

Using advanced printing functions and events

The following functions and events are built into the ALLText control. They allow complete control over printing including specification of the output device (even a picturebox), control over the printer progress dialog box, suppression of page breaks, printing of additional information such as headers and footers. These functions must be declared in your Visual Basic Program (see: ATX4hAPI.BAS or ATX4SAPI.BAS).

ATX_Print_Start Function - initializes ALLTexts link to an output printing device. Depending on settings of the parameters, end-user access to the control during printing may be disabled, and the printer progress dialog box may be turned off.

ATX_Print_Finish Function - terminates ALLTexts link to an output printing device

ATX_Print_Title Function - sets the text to be used in the printer progress dialog box. To prevent display of the PrinterDialog box the FormWnd parameter of ATX_Print_Start function should be set to 0.

ATX_Print Function - Prints a paragraph range, generating new pages as required.

ATX_Print_Region Function - Prints within a specified region on a single page.

ATX_PrintThisPage Event - This event is triggered for each page after calling the ATX_Print Function and before the ATX_PrintStartPage event

ATX_PrintStartPage Event -This event is triggered just before printing each page.

ATX_PrintEndPage Event - This event is triggered after printing each page.

The following describes the steps in using the Advanced Printing functions to control printing (The Third and fourth steps can be repeated numerous times):

Note that even with the width settings exactly the same, unless the PrinterDC property is set to TRUE, the output may line break a word prior to, or after the break shown in the control window. This is largely due to differences in printer resolution

1. If you are using the PRINTER OBJECT you should initialize it.

```
Printer.EndDoc
Printer.print
```

2. Call the **ATX_PRINT_START** function specifying the current form window handle and printer or BITMAP device context (DC).

```
X% = Atx_Print_Start(Form.HWND, Printer.hDC,1,1)
```

This function initializes ALLText link to the proper output device and presents ALLTexts standard print dialog box. Setting the first parameter to 0, will prevent display of the print dialog.

3. (Optional) Create a Title Page or initial information on a given page using standard coding techniques. This does not involve ALLText.

4. Call the ATX_Print **or** ATX_Print_Region function

4a.Call the ATX_PRINT function passing it the ALLText control handle for the source document. After calling this function, the specified ALLText window starts to generate printing events. For each printing page (except the last one) three following events are generated:

PrintThisPage (PageNumber As Integer, DialogString As String, SkipPageFlag As Integer)

This event can be used to skip printing of some pages. For example you can use this event if you want to print only odd or only even pages. If you set SkipPageFlag parameter to TRUE during processing this event current page will not be printed and next two event will not come.

PrintStartPage (PageNumber As Integer, TopIndent As Long, LeftIndent As Long)

This event can be used to change location of the printed page on printing media. It is called just before printing a given page.

PrintEndPage (PageNumber As Integer, SkipEndPageFlag As Integer)

This event is called just after completing a page, before ALLText issues an end-page command. It may be used to print some additional information on the current page. This can be done with Visual Basic functions, WINAPI functions or ATX_PRINT_REGION function. Using these techniques it is possible to print Headers and Footers. Furthermore SkipEndPageFlag determines whether to print next page on the same sheet or not. If you set this flag to TRUE printing will be made on the same sheet.

4b. Call the The ATX_PRINT_REGION passing it the ALLText control handle for the source document. This function restricts output within a specified region on the page. If the text does not fit within a given region it will be truncated and an error code will be generated. No events occur while this function is operating.

5. (*Optional*) Add additional information to the printed page. Additional information on the page can be added with Visual Basic functions, WINAPI functions or the ATX_PRINT_REGION function. To do this you should use StartTop& parameter that tells where user can print additional text without spoiling text that has been already printed.

6. Call the ATX_PRINT_FINISH function. This function shuts printing dialog down and stops printing process for the ALLText. If you printing were sent to the PRINTER OBJECT you should call printer.EndDoc after calling this function.

Example:

```
Sub Command1_Click ()
'Call printer.Print to initialize the printer object
' This is important for next ATX_PRINT_START call
    printer.Print " "
'Initiate link from ALLText to the Printer
    i% = ATX_PRINT_START(Form1.hWnd, printer.hDC, 1, 1)
'Specify margins etc and start to print
      printfrm_StartTop& = 1440              ' 1 inch top margin
      printfrm_top& = 1440                   ' top margin after 1st page
      printfrm_left& = 1440                  ' left margin
      printfrm_TextOnPageY& = 2 * 1440       ' height of print area.
      printfrm_width& = 6 * 1440             ' width of print area
      i% = ATX_PRINT(atx4h1, printfrm_StartTop&, printfrm_top&, printfrm_TextOnPageY&,
           printfrm_left&, printfrm_width&, 0, -1, 0)
'Break connection to Printer
      Call ATX_PRINT_FINISH


     On Error Resume Next
     printer.EndDoc
End Sub
```

How to Control the Cursor Position & Select Text

Cursor Control

ALLText provides full control over reading and setting cursor position and/or select region.

ALLTexts *SelStart*, *SelLength* properties work in a manner similar to the standard textbox. SelStart points at the character offset from the start of the document. Setting SelStart=0 places the caret at the start of the document. Setting SelStart to the TextLength property places the caret at the end of the document. SelLength specifies describes the length of the select region.

Cursor positioning and selections may also be specified through paragraph and paragraph offset properties: *CurPar* and *CurChar* define the current cursor location as well as the start of a select region in terms of the current paragraph number and current character offset within that paragraph. The end of a select region is specified in the same fashion using the *SelToPar* and *SelToChar* properties. To select a text fragment, set the caret at one edge (start or finish) of this fragment by assigning appropriate values to CurPar and CurChar, and then designate the second edge of the desired range through assignment of values to SelToPar and SelToChar. The text selected in this manner has a starting point specified by CurPar, CurChar and final edge specified by SelToPar, SelToChar. Note that prior to setting a select region, SelToPar=CurPar, SelToChar=CurChar and Select=FALSE.

For example,

```
' Set caret 5 characters from the start of the 10th paragraph
      AllText1.CurPar = 9   'CurPar starts counting at 0
      AllText1.CurChar = 4 'Same for Curchar
'Select to the end of the paragraph
'Note that setting CurChar or SelToChar at the maximum possible value (ATX_MAXCHARS) will
'result in automatic setting to the last character in the text.
     ALLText1.SelToChar= ATX_MAXCHARS
```

Lastly cursor positioning may be specified by page number using the *PageNumber*, *LineNumber* and *PageLineNumber* Properties. Setting the PageNumber property moves the caret to the start of a given page (0,1,2,3..). Setting LineNumber moves the caret to the start of a given line counted from the beginning of the document. Setting PageLineNumber moves the caret to the start of a given line within a page.

One more property *Select*, may be used as a true/false flag to determine whether a select region is active. Setting Select to False is equivalent to setting the SelLength property to 0.

IMPORTANT: The SelStart property may not be set to a value pointing beyond the end of the current selected region. Doing so will result in an error. To avoid this error it is generally advised to set the Select property to FALSE prior to moving SelStart.

Content Manipulation

AllText supports two distinct properties to manipulate selected text: *SelText*, which sets or returns the selected text with no embedded formatting codes, and *SelFText,* which functions in a manner similar to SelText but with embedded formatting codes included within the string.

Example:

```
' Replace 1st paragraph by a formatted string "ZeroOne"
' Set the formatted string with embedded codes
      Str$ = "\f0 \cf0 Zero \f1 \cf1 One"          ' Zero will be black, One will be Blue
' Set the select region to include the 1st paragraph
      AllText1.CurPar = 0:       AllText1.CurChar = 0
      AllText1.SelToPar = 1:    AllText1.SelToChar = 0
' Replace the contents of the first paragraph
      AllText1.SelFText = Str$
```

Note also that by properly setting the *WriteProtect* property, an end user need not see your changes in

setting select regions.

```
' Read the first 15 characters of a 4th paragraph (ALLText starts counting with 0)
    wp% = ALLText1. WriteProtect                    ' save write protect setting
    ALLText1. WriteProtect = ATX_PROTECT_KEYBOARD   ' prevent display of selection bar
    ALLText1.CurPar = 3: ALLText1.CurChar=0:ALLText1.SelToChar=14
    s$=ALLText1.SelFText      ' may be longer than 15 characters as it will include formatting codes
    ALLText1.SelText = FALSE ' turn off select region
    ALLText1. WriteProtect = wp%                    ' return to former edit mode
```

To Summarize: Current position and selection properties are:

CurPar	Current Paragraph or Paragraph specifying start of selected text.
CurChar	Number of Current Character within Current Paragraph text buffer or Character specifying start of selected text
SelToPar	Paragraph specifying end of selected text
SelToChar	Character specifying end of selected text
SelText	String containing selected text without font and color control information.
SelFText	String containing selected text with font and color control information.
Select	Indicates that some fragment of Text is selected.
SelLength	Sets or returns the Length of the Select region.

PageNumber - specifies the current page.

LineNumber - specifies the current line number from the top of the document

PageLineNumber - specifies the current line number within the current page

How to Programmatically Control Scrolling

ALLText offers complete programmatic control over scrolling. ScrollHorz and ScrollVert Properties determine the scrolled distance between the top left corner of the document and the top left corner of the ALLText window. Using these properties it is possible to scroll the window in increments as fine as allowed by the physical device.

```
For I = 0 to 50
        ALLText.ScrollVert = I * Screen.TwipsPerPixelY
        Next I
```

ALLText *ATXVScrollClick and ATXScrollHClick* events offer additional control. These events are triggered after the control is scrolled with the scroll bar click. Normally ALLText responds to vertical scroll clicks by scrolling the control contents vertically by a fixed number of twips. Since each text line can be of some size determined by the fonts used it is not always going to be the case that this scroll distance corresponds to a integer number of lines. The result is that the top line of the textbox may be cut off in the middle as a result of scrolling. If desired, code can be written to set the *ScrollVert* property after a user click such that the top line is always lined up with the top of the control window (not cut off). Sample code is included below.

```
    Sub ALLText1_ATXVscrollClick (ScrollVert As Long)
'Identify top line in textbox
    X& = 0
    Y& = ALLText1.ScrollVert
    Ret_Code% = ATX_XYToCur(ALLText1, X&, Y&, p&, c%)
'Identify top of top line in textbox
    Ret_Code% = ATX_CurToXY(ALLText1, p&, c%, x1&, y1&)
'Set ScrollVert to top of top line in textbox
    ALLText1.ScrollVert = y1&
End Sub
```

Note that this will not prevent the bottom line from cutting off in the middle. If this is a big concern, youll want to choose your fonts and line spacing such that an integral number of fonts fit in the textbox, or you can write similar code to identify the top of the line immediately below the client area of the control, and then dynamically adjust the textbox height. For best results in this sort of application, the controls border should be turned off so that the end-user does not see the height changes.

Of course it is easy to turn the Scrollbars on and off, either at design time or run-time. ScrollBarH and ScrollBarV properties provide this support.

How to Add a Table

ALLText 4 does not currently support embedded Tables. Were working on this and expect Full Table support in Fall 1995.

ALLText HT/Pro can read in an RTF document with embedded Tables and will translate to tab delimited text, attempting to preserve the general look of the table.

Alternatively developers may embed OLE objects such as a section of an excel spreadsheet to achieve the look of a table.

How to Add Header and Footer Support

ALLText 4 supports the definition of Headers and Footers to be stored with the document. Headers and footers may be defined for the document as a whole, for even pages, for odd pages, or for the first page.

To create a Header or footer set the appropriate HeaderFText or FooterFText property, Note that these are list properties. Thus :

HeaderFText (0) specifies a header for the document as a whole
HeaderFText (1) specifies a header for even pages
HeaderFText (2) specifies a header for odd pages
HeaderFText (3) specifies a header for the document as a whole
These properties actually hold the Formatted text string corresponding to the header or footer. If you read in an RTF file with headers or footers, the headers and footers will NOT be shown in ALLText, but will be interpreted and stored by the HeaderFText and FooterFText properties. You can if you wish then add this text to secondary ALLText boxes. For Example:

```
Sub Command5_Click ()
    ALLText1.DataType =3
    ALLText1.FileName = Sample.RTF
    ALLText1.FileLoad = ATX_IO_FAST
    For i = 0 To ALLText1.FontTableSize - 1    'copy font table to secondary controls
        ret_code = FontTableGet (ALLText1, I,.....)
        ret_code = FontTablePut (ALLText2, I,.....)
        ret_code = FontTablePut (ALLText3, I,.....)
        Next I
    If Len(ALLText1.HeaderFText(ATX_HEADER_SIMPLE)) > 0 Then
        ALLText2.FText = ALLText1.HeaderFText(ATX_HEADER_SIMPLE)
        End If
    If Len(ALLText1.FooterFText(ATX_FOOTER_SIMPLE)) > 0 Then
        ALLText3.FText = ALLText1.FooterFText(ATX_FOOTER_SIMPLE)
        End If
End Sub
```

The ATXPrintRegion function can then be used to print the header and footers.

For further information refer to the sample project: **PrintHF**.

How to Cut, Copy & Paste

ALLText supports formatted cut and paste between different sections of the document and different Windows applications such as MS Word™ or AMI Pro™. All font, color and paragraph formatting will be preserved when copying from applications which support RTF Clipboard operations (Word, WordPerfect, AmiPro). When copying from the clipboard, ALLText will not recognize formatting (such as tables) which it does not support. It will however do it's best to preserve the formatting. Likewise, certain ALLText formatting styles (such as custom shadowing and Hypertext tagging (HT/Pro only) are not supported by other applications and may be incorrrectly interpreted by them.

NOTE: In order to properly support cut and paste of embedded Pictures, you must have the ATXPIC.DLL present and REGISTERED.

KeyBoard support for Cut, Copy and Paste

ALLText automatically recognizes certain key combinations as a command to interact with the windows clipboard. Note that these keyboard operations may be ignored by ALLText depending on the *WriteProtect* property setting.

Cut	<Cntrl> X	or	<Cntrl> <Delete>
Copy	<Cntrl> C	or	<Cntrl> <Insert>
Paste	<Cntrl> V	or	<Shift> <Insert>

Programmatic Support for Cut, Copy and Paste

The ALLText .**ClipboardAction** property provides for full clipboard support. The following property settings apply:

```
ALLText .ClipboardAction   =1      ' cut selected text and place in clipboard

ALLText .ClipboardAction   =2      '   copy selected text to clipboard

ALLText .ClipboardAction   =3      ' paste text from clipboard.
```

Note that when pasting from the clipboard, text will be inserted at the current cursor location and will replace any text already selected when this property is set. The cursor will then be set immediately AFTER the newly inserted text.

DO NOT use Visual Basics ClipBoard object for cut and paste support - the text held by the Clipboard object is NOT formatted.

Using SelText, SelFText, Text and FText to copy between controls

Copying Text from one location to another within a given ALLText document may also be accomplished by reading and setting the *SelFText* property. When copying between two distinct ALLText windows however this may very well result in unexpected results due to differences in the font table definitions held by distinct instances of the ALLText control (see font tables). Instead use the clipboard.

Eg: ' copy selected text from start of document to end

```
ALLText.SelStart = 0: ALLText.SelLength =5   'select first 5 characters.

X$ = ALLText.SelFText    ' get selected text with formatting codes

ALLText.Select = False    ' turn off selection

ALLText.SelText = ALLText.TextLength   ' move cursor to end of document

ALLText.SelFText = X$   'insert formatted text at end of document
```

After setting SelText or SelFText, the newly inserted text will remain formatted.

How to Create a Transparent Background

ALLText supports a BackStyle property which allows you to specify either a Transparent or Opaque background. It is important to note however that setting this property is really capturing a Static image of what lies behind ALLText. Visual Basic doesnt send messages saying that items behind the control have changed. For instance, ALLText has no way of knowing when a forms picture property is changed. Updating what lies behind ALLText will therefore have no immediate affect on the display. In many cases it may be better to place a bitmap image in the BackPicture property instead of working with the transparent background.

Ideally the BackStyle property should be set at Design time only via the properties window. There are however several situations when it is necessary to change the BackStyle in runtime mode. In this event, it is important to set the controls Visible property to false when changing the background style at runtime. For example:

```
Sub SwitchToOpaque (atx As Control)
    atx.Visible = False
    atx.BackStyle = ATX_Opaque
    atx.Visible = True
End Sub
Sub SwitchToTransparent (atx As Control)
    ALLText.Visible = False
    ALLText.BackStyle = ATX_Transparent
    ALLText.Visible = True
End Sub
```

When items behind the control are changed, a transparent ALLText control should be 'manually' updated by resetting the backstyle to Transparent again, this could be done in the Forms repaint event, or after specific actions such as changing the back picture.

```
Sub NewFormPicture (Pict As Control)
    Picture = Pict.Picture     'changes the forms picture property
    If ALLText.BackStyle = ATX_Transparent Then
        ALLText.BackStyle = ATX_Transparent
        End If
End Sub
```

How to Add a Background BitMap / WaterMark

ALLText uniquely supports the display of a background image positioned between the standard control background and the text itself. The effect may be used in a variety of manners - simulating transparency, setting up a spot light, framing text within a set of curtains, combining text and pictures as in a cartoon, or simulating the sliding of a cardboard over an overhead transparency as in a presentation. For example: Set up ALLText with black text and a black background. Nothing shows up. Now set in a round white ball as an image using BackPicture. Move it around with BackPictureX and BackPictureY. The effect is that of a spotlight with the black text showing up wherever the circle is displayed.

The background image is set using the BackPicture Property. The picture itself is set by assignment to the BackPicture property either through use of the standard Visual Basic LoadPicture function, or by assignment based on the picture property of another control. Positioning of the background picture is supported through the BackPictureX and BackPictureY properties. The actual display location of the picture does not change with resetting of the x & y coordinates until refreshed by setting the BackPictureRefresh property.

Example

```
AllText1.BackPicture = Picture1.Picture
for i% = 1 to 10
    AllText1.BackPictureX = i%*100
    AllText1.BackPictureY = i%*50
    ALLText.BackPictureRefresh = 1
  Next
```

How to Size the Control to Display ALL the Text

Many applications call for a textbox sized such that all the text fits into the control without scrolling. The ALLText DocWidth property can be set to control wordwrapping within the control. For most applications the DocWidth should be set to a value of .85 to .95 times the control width. The ALLText DocHeight property can be read at run-time to determine the total height of the formatted text content in twips. The controls Height property can then be set to a greater value.

How to Link to a Database

ALLText HT/Pro can be bound to either a Memo or a Binary field. Note that the Standard version of ALLText is not a data aware control and can not be bound to a database.

The following standard properties are associated with the use of ALLText as a bound control:

DataSource,

DataField,

DataChanged.

These properties work in a maner analogous to the standard Visual Basic properties.

In addition, the format in which the document is to be accessed in the database is controlled by DataType property. Valid DataType settings include:

- Text format (0) can be used for Raw unformatted text, making it easy to read using any editor.
- RTF format (3) may be useful if you have some other application which can read the database and can understand RTF.
- ATX_F format (2) is generally preferred as storing the document and its font table in the most compact form.
- ATX_S format (1) may be used in preference to ATX_F if your font table is set up by your application code using FontTableGet and FontTablePut Functions. ATX_S differs from ATX_F in that it does not include the font table. This may save some small amount of space in your database.

IMPORTANT: Note that if you wish to handle the database manipulations yourself (the only option if using the standard version of ALLText), you can grab the formatted text using the FText property, this will hold up to 32K including formatting codes.

If your document is larger than will fit in the FText property, you can use Low Level I/O to read and write the document to strings rather than disk files. Sample code is provided below:

```
Sub GetText()
    'loads the the control with the data returned in ATX GET
    Dummy% = ALLText.ClearALL           ' clear control
    ALLText.FontTableSize=0             ' remove all unneeded fonts
    ALLText.DataType = ATX_Format_F     ' = 2
    ALLText.FileLoad =ATX_IO_LowLevel   ' = 3   ,  triggers ATX_GET event
    End Sub
Sub atxInfo_ATXGet (Flag As Integer, UserSTR As String)
    ' get data from database .  Event is triggered by FileLoad setting
     Static position&
    Flag = 1
    UserSTR = kbase!HyperText.GetChunk(position&, 25000)
    position& = position& + 25000
    If Len(UserSTR) = 0 Then
                Flag = 0
                position& = 0
            End If
        End Sub
Sub SaveText()
    'Sends control content out by triggering ATX_Put event
    ALLText.FontTableSize=0             ' remove all unneeded fonts
    ALLText.DataType = ATX_Format_F     ' = 2
```

```
    ALLText.SelStart = 0:   ALLText.SelLength = ALLText.TextLength ' select all text
    ALLText.FileSave =ATX_IO_LowLevel     ' = 3     ' triggers ATX_Put event
    End Sub
Sub atxInfo_ATXPut (Flag As Integer, UserSTR As String)
    'saves data in database.   Triggered by FileSave setting
        Static sFlag%
        If sFlag% = 0 Then
            kbase!HyperText = ""
        End If
        kbase!HyperText.AppendChunk UserSTR
        sFlag% = Flag
    End Sub
```

IMPORTANT: Note that LowLevel I/O is restricted to passing not more than 32K of information in each call of the ATXPut or ATXGet events. Also ALLText can not process portions of images or OLE objects - these must be passed in a single piece. Thus the limit on the size of such objects is 32K when using Low Level I/O.

Note that none of this manual manipulation is necessary if you are using the HT/Pro edition and binding the control to the database field. It will all be handled for you automatically.

How to Set and Modify Tab positions (HT/Pro only)

ALLText HT/Pro supports two forms of tab support: relative (served by the TabStep property discussed under paragraph formatting) and absolute (served by TabLocations, TabCount, TabAdd, and TabDel properties). Each paragraph can have its own tab configuration.

Tab Support is provided via the following properties

TabEnabled- enables the use of the TabKey as a trueTab rather than change of focus.

TabStep- Sets relative tab stepsize (distance between relative tabs) in twips.

TabLocations - An Array property holding all the absolute tab locations for current or selected paragraphs.

TabCount - Returns the max number of tabs in selected paragraphs of text. Also used in clearing tabs.

TabAdd - Adds a new tab into the current or selected paragraphs.

TabDel - Removes a tab from the current or selected paragraphs.

TabAlignment - An array property holding the tab alignments for selected paragraphs (TabAlignment), indexed from 0 to *TabCount*-1. Also sets or reads the default alignment for TabAdd.

Relative Tabs are measured in twips from each other. For example, in the absense of absolute tab settings, Setting .TabStep =500, results in tabs positions of 500, 1000, 1500, 2000, ... twips from left margin of text.

Absolute tabs positions may be set as well. Use the TabAdd property to add a new tab setting.

```
ALLText1.TabAdd = 150       'adds a tab location at 150 twips from the left margin.
```

Note that Absolute Tabs are specified by their positions measured in twips from left margin of the document (Not the paragraph margin). Thus a tab set at 150 twips is not affected by LeftIntent settings of less than 150 twips, while such a tab is ignored as unachievable for LeftIndent settings greater than 150 twips.

Each absolute tab location may have its own alignment. To set or change alignments the TabAlignment property may be used. This is a list oriented property whose index refers to a given tab location for a paragraph TabAlignment(0) for the first absolute tab location in a paragraph, TabAlignment(ALLText.TabCount) for the last tab position in a paragraph.

```
ALLText1.TabAdd = 200:
ALLText1.TabAdd = 150:
ALLText1.TabAdd = 368

'Note at this point there are three absolute tab positions,
'          The first at 150 twips, the second at 200 twips and the third at 368 twips.

ALLText1.TabAlignment(0) = ATX_TAB_CENTRED   ' Index 0 points at left-most tab
ALLText1.TabAlignment(1) = ATX_TAB_RIGHT      ' 200
ALLText1.TabAlignment(2) = ATX_TAB_LEFT       ' 368
```

Absolute tabs are of greater priority than relative tabs in the sense, that relative tab acts only to the positions after last absolute tab position. For example, setting

```
ALLText1.TabStep = 150:          ALLText1.TabAdd = 140:          ALLText1.TabAdd = 280
```

forces the following tab positions: 150, 280, 300, 450, 600, ... up to paragraphs right margin.

Tabs are measured with 16 pixels accuracy. If the position of a new tab specified by the TabAdd property is less than 16 pixels away from nearest existing tab, a new tab is not inserted, but the position of the old tab is modified. The constant 16 pixels is declared in ATX4sAPI.BAS or ATX4hAPI.BAS file as ATX_TAB_MIN_PIX. In addition, the tab positions will be rounded to the nearest multiple of the

PixelsPerTwip setting for the current screen resolution.

NOTE: in previous versions of ALLText, Right and Centered tabs applied only to the word immediately following the tab character. ALLText 4 handles Right and Centered tabs in a manner more akin to stand alone applications - the alignment is applied to all further text on the same line up to the next tab location.

How to Change Case

ALLText includes two functions to facilitate changing from Upper to Lower Case or Lower to Upper case. These functions are declared in the ATX4HAPI.BAS and ATX4SAPI.BAS file :

ATX_ToLower and ATX_ToUpper.

Simply select the text whose case must be changed and call the appropriate function:

Call ATX_ToLower (Cntrl_Name)

How to Search & HighLight, or Search & Replace

ALLText has two built in search functions: Find_Phrase and Find_PhraseEX. Find_Phrase searches for the next occurance of a given phrase in the document, without regard for the formatting. Find_PhraseEX can be used to to search for text with specific formatting. In either case, the search begins at the current cursor position.

Input parameters of this function include specification of the control to search, the text to search for, the search direction, case sensitivity, and (for Find_PhraseEX) the formatting parameters for the text being sought. For a complete list of the parameters and sample code, refer to the technical descriptions section. Upon returning, the function sets the StartPos& and StopPos& functions to the location of the found phrase. The return value is True if the phrase is found, false otherwise.

To Search and Replace, use the Find_Phrase or Find_PhraseEX functions, select the text which is found and set SelText or SelFText to overwrite the selection.

A sample search and replace project is included in the samples directory FIND.

How to Select on DoubleClick

ALLText does NOT automatically select words on a double click. This is an intentional ommision. The code required to support such a feature is minimal and adding it to ALLText would limit its flexibility to users who do not want such a feature.

To support Auto Select of a word on DoubleClick add the following code to the DoubleClick Event:

```
SendKeys ^{Right}, TRUE        ' Simulate keyboard entry of move to end of word
SendKeys +^{Left}, TRUE        ' Simulate keyboard entry of Select to start of word.
```

Embedding OLE Objects, Images and VB Controls (HT/Pro Only)

ALLText HT/Pro, ALLText/Forms, ALLText/HTML, and ALLText/Pen support the embedding of external objects such as Images, OLE Objects, Pen Inkings, and even other custom controls. Such embedding is handled with the aid of an External Object DLL (ExtObjDLL) library such as ATXPIC.DLL or ATXOLE.DLL.

Each external object appears in the document as a bitmap and takes the place of a single character when displayed within the text stream.

Embedded objects can be printed with the document, saved in standard RTF format and even cut, copied or pasted.

OLE Object embedding is supported through the ALLText External Object DLL, ATXOLE.DLL. Embedded OLE objects may be used for the linking of applications or for the simple display of non-textual data such as spreadsheets or charts within the text application. Note that no OLE server is required to veiw a document with embedded OLE objects. An OLE server is only required for the actual embedding process and to respond to an OLE action event such as a double click.

Picture embedding is supported through the ALLText External Object DLL, ATXPIC.DLL, provides full support for the direct embedding of images. No external OLE server is required. Supported picture formats include: BMP, DIB, WMF, JPG, and ICO. ATXPIC is responsible for the initial embedding of pictures, as well as for the manipulation of the objects: setting picture properties, displaying, printing, saving, restoring etc.

Custom Controls and Ink Objects may be embedded within an ALLText document only through the use of the ALLText External Object DLL, ATXCTL.DLL. Such support requires either ALLText/Forms, ALLText/HTML, or ALLText/Pen. Separate documentation and licensing is required.

TECHNIQUE

In general, working with embedded objects involves four activities.

1. Registering the External Object DLL to support the type of object desired.

2. Embedding the object

3. Setting any desired attributes for the object (does not apply to OLE objects)

4. Responding to any specific events associated with the object

Registering the External Object DLL is done using a call to the *ATX_REGNEWEXTERN* function. This function establishes a link between the ALLText control and the DLL. Once the link is established it remains in effect until ALLText is removed from memory. Thus the function need only be called once for each External Object DLL while ALLText is in memory. Usually this will be in the initialization routine of your application (such as in a Form_Load event).

```
Sub Form_Load ()
        Ret_code = ATX_REGNEWEXTERN (0, "atxpic.dll")
        Ret_code = ATX_REGNEWEXTERN (0, "atxole.dll")
        ....
End Sub
```

Embedding the object is handled by setting the *OLEObject* property. Note that the property name is a holdover from version 3 of ALLText when OLE objects were the only type of object which could be supported. Embedding of other objects such as pictures using the ATXPic External Object DLL, does not make use of OLE technology or require the overhead associated with OLE. Only objects embedded using the ATXOLE.DLL uses OLE technology.

To embed an object position the cursor to the insertion point (ex: ALLText.SelStart = 5) , or select a portion of text to be overwritten (ex: ALLText.SelLength =20), and then set the OLEObject property as follows:

Ole Object - ALLText.OLEObject = OleServerName & "*" & FileName.

'Example: inserting a Paintbrush document
ALLText1.OLEObject="PBrush*d:\windows\cars.bmp"

'Example: inserting a bitmap using default OLE server defined by Windows
ALLText1.OLEObject="*d:\windows\cars.bmp"

'Example: inserting a Word Document
ALLText1.OLEObject="WordDocument*C:\Sample.Doc"

'Example: Creating and inserting a NEW OLE object using Excel
ALLText.oleobject = "Excelworksheet"

Ole Object - ALLText.OLEObject = External ObjectName & : & FileName

'Example: Embeds CARS.BMP and then BLUESEDAN.WMF
ALLText.OleObject = "atxpic:\cars.bmp"

ALLText.OleObject = "atxpic:blusedan.wmf"

'Example: Embeds empty picture object (see section below on setting attributes)
ALLText.OleObject = "atxpic:"

Reading or setting attributes of External Object

The attributes of OLE objects are set by the OLE server and can not be reset from within ALLText.

Picture objects have the following attributes accessible from AllText through the ExtObjParam and ExtObjValue properties:

Width - Specifies the width of the picture in twips.

Height - Specifies the height of the picture in twips.

Base Line Descent - Specifies an offset relative to the text base line in twips.
(Positive values lower the object relative to the text).

Frame Type - Specifies the type of the frame that will be drawn around the picture.
FrameType% = 0 No frame around the picture
FrameType% = 1 Single line around the picture
FrameType% = 2 Shadow frame
FrameType% = 3 Up 3D frame
FrameType% = 4 Down 3D frame

PictureHandler - Specifies a handle to a picture. The picture can be changed by setting ExtObjPicture to a valid picture.

When first loading a picture object into an ALLText document, ALLText initializes the size and other attributes using settings read from the source picture file itself. Picture attributes then may be read or modified through the ExtObjValue property (supports Width, Height, FrameType, and Base Line Descent attributes) and the ExtObjPicture property (allows you to change the image).

1. First select the picture object or position the caret immediately before the picture object you wish to modify.

2. Next specify the attribute you wish to read or modify by setting ExtObjParam to a string containing the module name atxpic, a colon (:), and the desired attribute name.

3. Now read or set the desired attribute using the ExtObjValue property, or the ExtObjPicture property to manipulate the picture:

```
Syntax:  ATX.ExtObjParam=atxpic:<propname>
         ATX.ExtObjValue = Value

Ex:      ATX.OleObject = "atxpic:help.ico"
         ATX.ExtObjParam=atxpic:Width
         ATX.ExtObjValue = Width%

Ex:      ATX.OleObject = "atxpic:"                    'Create empty picture
         ATX.SelStart = ATX.SelStart - 1      'position cursor
         ATX.ExtObjParam="atxpic:PictureHandler"
         ATX.ExtObjPicture = LoadPicture("\windows\arcade.bmp")
```

```
or   ATX.ExtObjPicture = clipboard.GetData()
```

As you can see from the second example it is possible to create an empty picture object and then assign the picture at a later time.

Responding to events of external objects

Double clicking on embedded images and OLE objects triggers the following events.

An End-user double click upon a picture object will trigger a standard double click event, followed by an ATX_ExternOleAction event. Parameters of the ATX_ExternOle event which will be set include: OleAction = 6 and OleServer = ATXPic, other parameters may be ignored.

An End-user double click upon an OLE object will launch the OLE Server unless the *OLEVerb* property has been set to ATX_VERB_UNDEFINED.

Ex: ALLText1.OLEVerb=ATX_VERB_UNDEFINED 'Prevent user launch of OLE server

How to Use the TabKey

As a Word Processing control it is important that ALLText be able to correctly interpet the TAB key as a method of entering tab characters in the document. In various applications, however it may be desirable to process the tab key as a change focus request. To accomodate either mode of operation ALLText supports a *TabEnabled* property.

With TabEnabled set to a value of True, ALLText accepts the Tab character for insertion into the text. With TabEnabled set to a value of FALSE, ALLText treats the TabKey as a focus change mechanism similar to other controls.

How to Implement Drag & Drop

ALLText 4 supports standard VB DragDrop methods as well as support for the dropping of files from file manager style programs into ALLText. Dragging of selections from within an ALLText document is not implemented.

The standard VB DragDrop properties and events are supported for standard drag and drop. (See standard VB help for documentation information.)

In order to support dragging of files from FileManager, ALLText 4 includes a DropFileMode property as well as ATX_DropFileStart and ATX_DropFile events. The DropFileMode property determines whether ALLText will (=1) or will not (=0) respond to files dropped from File Manager. With DropFileMode ON, ALLText will first trigger the event ATX_DropFileStart providing X and Y coordinates for the drop location and a third parameter indicating the number of files being dropped. Next the ATX_DropFile event will be triggered once for each file dropped and will provide the name of the file being dropped.

How to Implement UNDO

ALLText keeps track of changes to the control's content as they take place. Reading the UnDoAction property will return the number of possible UnDo steps currently stored in the buffer.

Setting the UnDoAction property to 1 will undo the last action. Setting to 0 will empty the buffer.

For example:

```
ALLText.SelText = "fred"
ALLText.atx.UnDoAction = 1      'Perform one step undelete operation
```

The <Alt> <Backspace> keyboard combination is also recognized for end-user support.

How To Trap Special Keys

ALLText supports the standard KeyDown, KeyPress and KeyUp events. These may be used to trap special keys for processing.

For example

```
Sub ALLText_KeyPress (Key as Integer)
    If Key = 13 then Text1.Text = ALLText.FText
Exit sub
```

How to Setup BookMarks

ALLTexts HTag and NTag properties may be used in conjunction with Find_HTag and Find_NTag functions to create bookmarks. The main difference between HTags and NTags is that by setting MouseHPointer property, a distinct mouse pointer may be shown over the HTagd phrases.

In response to an end-user action such as a menu choice Create BookMark, check to see if any text is selected. If not, select a single character by setting SelLength = 1. Next set either the NTag or HTag properties to a long integer value. The text is now tagged.

In response to an end user action such as a menu choice GoToBookMark, turn off selected text (ALLText.Select = False), move the caret to the start of the text (ALLText.SelStart =0) and then call either the Find_NTag or Find_HTag functions to search for a specific tag, or simply the next tag.

```
                x% = Find_HTag(ALLText, 0, 1)       ' find the next tagged text
or              x% = Find_HTag(ALLText, tag&, 1)    ' find a specific tag
```

How to set up Mail Merge fields (HT/Pro edition only)

ALLText does not have merge fields per se, but this is easy to set up using ALLText HT/Pros *NTag* property.

To build such an application, assign an integer tag value to selected phrases in your document. Use embedded code \ATXntNNN to identify start of region tagged with value NNN, and \ATXnt0 to identify the end of the region. For example, if you have a phrase such as, "The amount of ####s money is ****", You could tag the "****" with a value such as 30, and the #### with a value such as 40.

```
ALLText.FText = "The amount of \ATXnt40 ##\ATXnt0 's money is \ATXnt30 **\ATXnt0 "
```

After loading the tagged document into ALLText, use the Find_NTag function in a loop to jump to each tagged phrase. Read off the NTag value and replace the tagged text with information from a database (or other information source) based on that value. Phrases tagged with number 30 would be replaced by data from one field and tag number 40 would be replaced by data from another field.

```
Select Case ALLText.NTag
  Case Is = 30
      ALLText.SelText = Text1.Text
      ALLText.NTag = 0
  Case Is = 40
     ALLText.SelText = Text2.Text
      ALLText.NTag = 0
  End Select
```

Sample code may be found in the MailMerg directory installed with ALLText.

Note that to insert large blocks of text you may initiating a FileLoad without first clearing the control. This add text at the current location, or replace any currently selected text.

How To Build a Hypertext Application (HT/PRO edition only)

ALLText HT/Pro has been specifically designed to include features for the support of Hypertext systems. Specifically we have added the following Hypertext supporting enhancements to our standard ALLText product.

1. The ability to assign a hypertext tag phrase to any string. The general idea is to create a select region, and assign a Tag number using the *HTag* or *NewHTag* properties. Any string with an assigned tag is now a hot phrase and will be supported by the *MouseHPointer*, the *Find_HTag* function and the *ATXChange* event.

2. The ability to assign a different mouse pointer which will automatically be displayed when passing over any hypertext tagged region. The *MouseHPointer* property determines which mouse pointer will be associated with hypertext regions.

3. The ability to use ALLText as a bound control. ALLText HT/Pro is now data aware. This means that you can use the data control to move between records, automatically updating the content of the ALLText control. One way to make use of this is to use the HTag value of any tagged phrase as a keyed index in a database. When a user double clicks on a tagged phrase, moving to the record pointed to by this value and display the new information.

4. The ability to take action when a user moves his cursor into a tagged phrase. The *ATXChange* event is triggered whenever the cursor moves into or out of a tagged phrase. The event parameters include the HTag value of both the previous and current region.

5. The ability to find a given hypertext phrase via the tag number. The *Find_HTag* function allows you to jump forward to any hypertext phrase.

ALLText HT/Pro was designed largely to address the needs of a large section of our users for Hypertext design functionality. To this end ALLText HT/Pro is a bound control, and supports hypertext tagging of phrases.

Selecting a phrase and then setting ALLText HT/Pros *HTag* property (*ALLText1.HTag=22)* creates a tagged region associated with a tag value. You can easily read off that tag value at any time, such as during a double click event when you may wish to use the value to pop up a definition window, move to a new document, or even initiate some other program.

Tagged phrases are automatically sensed when moving the cursor or the mouse. When moving the mouse over a HyperText tagged region, the mouse pointer will automatically change from that specified by the *MousePointer* property to that specified by ALLTexts *MouseHPointer* property. In addition the movement of the cursor into or out of a tagged area generates an *ATXChange* event. Mouse movements will also trigger this event once MousePointer and MouseHPointer properties have been set to distinct values.

The *Find_HTag* function (make sure you have included the constants definition file in your project to capture this function declaration) may be used to locate and move to a given occurrence of a tagged region, or to select the entire current hypertext phrase (if the cursor is positioned within such a phrase).

```
ALLText1_Click()
   If HTag =0  then  exit sub
'     select the current tagged phrase.
      x% = Find_HTag(CntrlName, TagNumber, 0)
      End Sub
```

Build a Word Processor

A basic Notepad Replacement

Adding A Ruler

Adding A Button Bar

Adding A Status Bar

ALLText Properties

There are many properties affecting the overall functioning and appearance of the ALLText control window. These are listed described below. (Properties requiring the HT/Pro edition are so noted)

Alignment	Specifies Paragraph Alignment
Backcolor	Specifies ALLText background color
BackPicture	Specifies a picture behind the text - a Watermark
BackPictureRefresh	Upon setting, moves the background picture to the X,Y coordinates
BackPicture	Paint bitmap picture within the Text.
BackPicture	X-coordinate of bitmap picture
BackPictureX	Specifies horizontal offset of background picture
BackPictureY	Specifies vertical offset of background picture
BackStyle	Specifies whether ALLTexts background is Transparent or Opaque
Border	**(HT/Pro)** Specifies the border style for a paragraph
Borderstyle	Specifies the border for the ALLText control
BottomIndent	Specifies line spacing after a paragraph
CaretWidth	Sets or returns the width of the ALLText current position marker.
ChangeEventMask	Specifies events which will trigger the Change Property
ClearAll	Empties the text box
ClipboardAction	Supports Cut, Copy and Paste operations
CurChar	Specifies cursor position relative to start of current paragraph
CurPar	Specifies current paragraph in which cursor is located
Datachanged	**(HT/Pro)** Indicates whether control content has changed
Datafield	**(HT/Pro)** Database field to which ALLText is bound
DataSource	**(HT/Pro)** Data Control to which ALLText is bound
DataType	Specifies format to use for file or database I/O
DirectScreenOut	Specifies method in which ALLText prepares text for screen
DocHeight	Specifies the height of document within the control
DocWidth	Specifies the word wrap width of document
DropFileMode	Specifies whether to allow file dropping
Enabled	Specifies whether the control is enabled
ExtDataType	**(HT/Pro)** Controls which formatting codes are used when conducting File or Database I/O with DataType set to 4.
ExtObjParam	**(HT/Pro)** Specifies the an external object attribute to be accessed with the ExtObjValue property
ExtObjPicture	**(HT/Pro)** Specifies a picture value of an embedded picture object
ExtObjValue	**(HT/Pro)** Specifies an attribute of an embedded external object
F2On	Determines whether the control responds to the F2 Key by displaying the Windows Font Selection Dialog box
F3ON	Determines whether the control responds to the F3 Key by sending the control contents to the default printer.
FileLoad	Initiates File I/O
FileName	Specifies file name for File I/O
FileSave	Initiates File I/O

FirstLineIndent	Specifies hanging indentation for a paragraph
FontBold	Specifies whether selected character is Bold
FontColor	Specifies color of selected characters
FontFamily	Specifies Font Family of selected characters
FontHidden	**(HT/Pro)** Specifies whether selected character is Hidden
FontIndex	Specifies font table entry for formatting of selected characters
FontItalic	Specifies whether selected character is Italic
FontName	Specifies font family of selected characters
FontShadow	Specifies shadowing style for selected characters
FontSize	Specifies Font Family of selected characters
FontStrike	Specifies whether selected character is RedLined
FontTableSize	Specifies number of entries in font table
FontUnder	Specifies underline style for selected characters
FontWidth	Specifies font width of selected characters
FooterFText	**(HT/Pro)** Contains formatted footer text for the document
FormatPaste	Determines whether copy and paste functions include interpretted font formatting codes.
FText	Contains first 32k of formatted content
HeaderFText	**(HT/Pro)** Contains formatted header text for the document
Height	Specifies height of the control
HelpContextID	Specifies context ID for link to an external help file
HTag	**(HT/Pro)** Specifies a Hypertext HotSpot Tag for selected text
HTagL	**(HT/Pro)** Specifies a Hypertext HotSpot Tag for text to left of cursor
HWND	Returns the handle to a control
Index	Specifies a unique instance of the control within a control array
Left	Specifies offset from left edge of ALLTexts container
LeftIndent	Specifies a left margin for selected paragraph.
LineNumber	Specifies line number offset from start of document
LineSpacing	Specifies the line spacing within selected paragraph
MouseHPointer	**(HT/Pro)** Specifies the MousePointer to use over hotspots
Mousepointer	Specifies the Mouse Pointer for use over the ALLText control
Name	Contains the Name of the ALLText control
NTagL	**(HT/Pro)** Specifies a bookmark Tag for text to left of cursor
NTag	**(HT/Pro)** Specifies a bookmark Tag for selected text
NumParagraphs	Returns number of paragraph within the document
OLECode	**(HT/Pro)** Specifies a character code to be returned within SelText string as a placeholder for embedded objects
OleObject	**(HT/Pro)** Controls embedding of external Objects
OleVerb	**(HT/Pro)** Controls action of a double click upon embedded OLE objects
Overtype	Specifies whether typing inserts new characters or replaces existing characters
PageHeight	Specifies height of a page of text
PageLineNumber	Specifies line number offset from start of current page
PageNumber	Specifies current page number
Parent	Returns parent

PrinterDC	Controls whether ALLText uses Printer DC for on-screen formatting
PTag	**(HT/Pro)** Specifies a Long Integer tag value for selected paragraph.
RightIndent	Specifies a right margin for selected paragraph.
ScrollBarH	Sets the presentation of the horizontal scroll bar either On, Off or Auto.
ScrollBarV	Sets the presentation of the Vertical scroll bar either On, Off or Auto.
ScrollHorz	X-coordinate of edit window in relation to start of Text.
ScrollVert	Y-coordinate of edit window in relation to start of Text.
Select	Determines whether select region is on or off.
SelFText	Specifies the selected text including embedded format codes
SelFType	Specifies how formatting codes are interpretted for selected text
SelLength	Specifies the length of a select region
SelStart	Specifies starting location for a select region
SelText	Specifies the selected text without embedded codes
SelToChar	Specifies character offset within paragraph to which selection region extends
SelToPar	Specifies paragraph to which selection region extends
ShowHidden	**(HT/Pro)** Determines whether or not to show hidden text.
TabAdd	**(HT/Pro)** Adds a defined tab stop for specified paragraph
TabAlignment	**(HT/Pro)** An array holding the alingment types of tabs defined for selected paragraph
TabCount	**(HT/Pro)** Returns the number of tab positions defined for selected paragraph
TabDel	**(HT/Pro)** Removes a tab stop location from the selected paragraph
TabEnabled	**(HT/Pro)** Specifies the function of the Tab key
Tabindex	Specifies the position within the focus change order of controls on the form
TabLocations	An array holding locations of aligned tabs for selected paragraph
TabStep	Specifies default tab step size for selected paragraph.
TabStop	Indicates whether the control can receive focus through use of the Tab key
Tag	Specifies a user defined string associated with the control
TextFormatted	Indicates how much of a document has been fully formatted after I/O
TextLength	Returns the number of characters in the document
Text	Contains first 32k of content - unformatted, without embedded codes
TopIndent	Specifies spacing before a selected paragraph
Top	Specifies offset from top edge of ALLTexts container
UnDoAction	Triggers an Undo Action
Visible	Specifies whether ALLText is Visible
Width	Specifies the width of the ALLText control
WriteProtect	Specifies the WriteProtect mode

NewHTag	Returns lowest possible HotSpot tag value not yet in use by control.
NewNTag	Returns lowest possible Bookmark tag value not yet in use by control.
FontCharSet	
InvisibleHidden	

Alignment Property

Description

Sets or returns the Alignment of the current paragraph.

Usage

ALLText.Alignment= iParNum%
iParNum% = ALLText.Alignment

Access

	Read	Write
Design	-	-
Execution	+	+

DataType

Integer

Values

ATX_LEFT (0)	for Left Aligned Paragraph
ATX_RIGHT (1)	for Right Aligned Paragraph
ATX_CENTERED (2)	for Centered Paragraph
ATX_JUSTIFIED (3)	for Justified Paragraph

Default

ATX_LEFT

Example

AllText1.Alignment = ATX_CENTERED

BackColor property

Description

ALLText supports a pallete of up to 256 pure (undithered) colors for the Back Color. When set, ALLText chooses the nearest pure color and fills the background with this color.

Default

Default value is equal to system window background color.

Remarks

If you need a dithered color background you may use a Transparent AllText placed on dithered form or over a dithered picturebox.

(It may be possible to use more than 256 colors depending on display driver and palette, but this has not yet been fully tested.)

BackPicture Property

Description

ALLText uniquely supports the display of a single background image which may be positioned between the standard control background and the text itself. This picture is set or read via the BackPicture Property.

The picture itself is set by assignment to the BackPicture property, either through use of the standard Visual Basic LoadPicture function, or by assignment based on the picture property of another control. Positioning of the background picture is supported through the BackPictureX and BackPictureY properties. The actual display location of the picture does not change with resetting of the x & y coordinates until refreshed by setting the BackPictureRefresh property.

Usage

ALLText.BackPicture = LoadPicture(FName$)

Picture1.Picture=ALLText1.BackPicture

Access

	Read	Write
Design time	+	+
Run time	+	+

DataType

Picture

Values

Any picture available via the LoadPicture property or contained in the Picture property of another control.

Example

```
for i% = 1 to 10
    AllText1.BackPictureX = i%*100
    AllText1.BackPictureY = i%*50
    AllText1.BackPicture = Picture1.Picture
    Next
```

Remarks

The BackPicture property may be used for a variety of special effects. The easiest way to understand how the background picture is displayed is to imagine Black text on a Black background. Nothing shows up. Now set in a round white ball as an image using BackPicture. Move it around with BackPictureX and BackPictureY. The effect is that of a spotlight with the black text showing up wherever the circle is displayed. Other effects possible using the Background Picture include; framing text within a set of curtains, combining text and pictures as in a cartoon, simulating the sliding of a cardboard over an overhead transparency as in a presentation. We leave the rest to your imagination. Please let us know if you come up with any novel ideas.

BackPictureX, BackPictureY Properties

Description

Set the top left corner coordinates of an included picture to paint on the Text background (coordinates are measured in relation to top left corner of Text).

Usage

ALLText.BackPictureX= x& ALLText.BackPictureY= y&
x& = ALLText.BackPictureX y& = ALLText.BackPictureY

Access

	Read	Write
Design time	+	+
Run time	+	+

DataType

Long (twips)

Values

-*MaxTextWidth* ≤ XTwipsToPixels(x&) ≤ *MaxTextWidth*
-*MaxTextWidth* ≤ YTwipsToPixels(y&) ≤ *MaxTextHeight*

Default

0

Remarks

When you set these properties nothing happens immediately. The result becomes visible only after setting of either the BackPicture property or the BackPictureRefresh property.

There are many situations when resulting paint will be without of scope. To be sure that result is always visible you must keep within the following constraints:

0 <= (x& - ALLText.ScrollHorz) <= ALLText.Width
0 <= (y& - ALLText.ScrollVert) <= ALLText.Height

BackPictureRefresh Property

Description

Setting the BackPictureRefresh property to a value of 1 resets the background picture to the location defined by BackPictureX and BackPictureY properties.

Usage

ALLText.BackPictureRefresh = 1

Access

	Read	Write
Design time	-	-
Run time	-	+

DataType

Integer

BackStyle Property

Description

Specifies whether the background is transparent or opaque.

Usage

ALLText.BackStyle = ATX_TransParent

Notes

Setting this property is really capturing a Static image of what lies behind ALLText. Updating what lies behind ALLText will therefore have no immediate affect on the display. In many cases it may be better to place a bitmap image in the BackPicture property instead of working with the transparent background.

Ideally the BackStyle property should be set at Design time only via the properties window. The Visible property should be set to False whenever changing the BackStyle at RunTime.For example:

```
Sub SwitchToOpaque (atx As Control)
    atx.Visible = False
    atx.BackStyle = ATX_Opaque
    atx.Visible = True
End Sub
Sub SwitchToTransparent (atx As Control)
    atx.Visible = False
    atx.BackStyle = ATX_Transparent
    atx.Visible = True
End Sub
```

When items behind the control are changed, a transparent ALLText control should be 'manually' updated by resetting the backstyle to Transparent again, this could be done in the Form's repaint event, or after specific actions such as changing the back picture.

```
Sub NewFormPicture (Pict As Control)
    Picture = Pict.Picture     'changes the form's picture property
    If atx.BackStyle = ATX_Transparent Then
            atx.BackStyle = ATX_Transparent
        End If
End Sub
```

Border Property (HT/Pro Version)

Description

The Border property of ALLText HT/Pro sets or resets a border around current or selected paragraphs.

Paragraph borders are defined by a variety of characteristics including which sides are bordered and the nature of the border (eg: shadowed, thick, double).

The state of underlining is determined by the value of the *Border* property. A value of False indicates no borders . Other values may be built up or interpreted through the use of two ALLText functions: *MAke_Border* and *Get_Border*.

Usage

alltext.Border = b&: b& = alltext.Border

Access

	R	W
Design	-	-
Runtime	+	+

Data Type

long

Values

True may be used for single line not shadowed box border

False may be used to reset borders to the default value - unbordered.

In general, values depend on border configuration.

For setting or getting real values, ALLText's *get_border* and *make_border* functions are strongly recommended.

Default

0 (not bordered)

If Out of Range

Nothing happens.

Example

```
' Make the border configuration for last paragraph the same as for first one.
AllText1.CurPar = 0
call get_border( AllText1.Border, side%, shape%, shadow%)
AllText1.CurPar = ATX_MAXPARAG
AllText1.Border = make_border( side%, shape%, shadow%)
' Set the second paragraph's bordering to double line, shadowed, right
' and bottom lines only.
AllText1.CurPar = 1
AllText1.Border = make_border( ATX_BORDER_RIGHT+ATX_BORDER_BOTTOM,
ATX_BORDER_LINE_DOUBLE, ATX_BORDER_STYLE_SHADOWED)
```

Remarks

If two paragraphs of text contain the same border configuration then their borders are joined. To split the borders of two paragraphs, insert an unbordered empty paragraph between them.

BottomIndent Property

Description

Sets or returns indent distance between the Bottom of the paragraph and the Top of the next Paragraph. (gets added to Topndent of nextparagraph)

Usage

ALLText.BottomIndent= n&
n& = ALLText.BottomIndent

Access

	Read	Write
Design	-	-
Execution	+	+

DataType

Long (twips)

Values

0 ≤ YTwipsToPixels(n&) ≤ *MaxTextHeight*

Default

1

Example

```
AllText1.BottomIndent = .5 * 1440       'set to ½ inch
```

Notes

The resolution of this property is dependent on the physical device. ALLText holds such information internally in Pixels such that

```
ATX.BottomIndent = Screen.TwipsPerPixelX + 1
```

is equivalent to

```
ATX.BottomIndent = Screen.TwipsPerPixelX
```

CaretWidth Property

Description

Determines the ALLText caret width in pixels.

Usage

n% = AllText.CaretWidth

AllText.CaretWidth = n%

Access

	R	W
Design	+	+
Runtime	+	+

Data Type

Integer (pixels)

Values

0 <= n% <= 100

Default

1

If Out of Range

A trappable error is generated, property value remains unchanged.

ChangeEventMask

Description

The ChangeEventMask property determines which AllText state changes cause AllText to trigger the ATXChange event.

The AllText state refers to the state (formatting, tags, etc) of selected text or the state of text at the current caret position if no text is currently selected (SelLength=0).

Usage

ATX.ChangeEventMask = l&

l& = ATX.ChangeEventMask

Data Type

Long

Remarks

This property can be assigned with a combination (boolean OR, or arithmetic addition) of following values listed in ATX4HAPI.BAS or ATX4SAPI.BAS:

Global Constant	Value	Trigger Event in response to Change in:
ATXTAG_CHANGED_MASK	&H1&	Htag
ATXMOUSE_CHANGED_MASK	&H2&	Mouse cursor
ATXFONT_CHANGED_MASK	&H4&	Font, including font characteristics such as underline
ATXCOLOR_CHANGED_MASK	&H8&	Font color
ATXCURSORLOCATION_CHANGED_MASK	&H10&	Caret location
ATXSELECTION_CHANGED_MASK	&H20&	Selected area
ATXLOCATIONX_CHANGED_MASK	&H40&	X location of the AllText window
ATXLOCATIONY_CHANGED_MASK	&H80&	Y location of the AllText window
ATXSELPARAGS_CHANGED_MASK	&H100&	Selected paragraph area
ATXPARALIGNMENT_CHANGED_MASK	&H200&	Paragraph Alignment
ATXMARGIONX_CHANGED_MASK	&H400&	Paragraph Margins: LeftMarg, RightMarg or FirstLineIndent
ATXSCROLLH_CHANGED_MASK	&H800&	Horizontal scroll
ATXSCROLLV_CHANGED_MASK	&H1000&	Vertical scroll
ATXPAGE_CHANGED_MASK	&H2000&	Current page number
ATXPAGELINE_CHANGED_MASK	&H4000&	Current Page Line Number
ATXLINE_CHANGED_MASK	&H8000&	Line number.
ATXFORMATED_CHANGED_MASK	&H10000	The value of the Textformatted property has changed - this may indicate progress in the background formatting process.
ATXCHANGED_CHANGED_MASK	&H20000	DataChanged property.

Example

To trigger an ATXChanged event upon either the changing of caret location or upon changes to the current font the VB code would be:

```
ATX.ChangeEventMask = ATXCURSORLOCATION_CHANGED_MASK or ATXFONT_CHANGED_MASK
```

Data Type

Long

See Also

ATXChange event

Default

(ATXTAG_CHANGED_MASK or ATXMOUSE_CHANGED_MASK)

ClearAll Property

Description

Upon reading the ClearAll property the control is reset, removing all Text and Paragraph structures (paragraphs, etc.). What remains is one empty paragraph. Returns a value of true upon success.

Usage

n% = ALLText.ClearAll Clears the control, but does not reset font table
If success returns IS_OK.

Access

	Read	Write
Design	-	-
Execution	+	-

DataType

Integer

Values

TRUE	The document has been successfully cleared.
FALSE	Error

Example

i% = AllText1.ClearALL

Remarks

Clearing the control does not change the DocWidth property.

Clearing the control does not change the FontTableSize - This is a change from versions of ALLText prior to 3.0. To eliminate unused fonts from the system, set the FontTableSize property to a value of 0.

ClipboardAction property

Description

The ClipboardAction property allows easy programmatic access to the clipboard (Cut, Copy and Paste) by assigning a value to this property.

Usage

ALLText.ClipBoardAction = x%

Access

	Read	Write
Design	-	-
Execution	-	+

Values

ATX_CUT=1

ATX_COPY=2

ATX_PASTE=3

(See constants in atx4hapi.bas or atx4sapi.bas file.)

Remarks

The ClipboardAction Property provides full Formatted cut and paste support between applications. Note that Visual Basics clipboard object does not support Formatted text. Use of GetText or SetText methods will only support the Raw Text.

Note that After Pasting, the Caret will be located BEFORE the pasted text.

ClipControls Property

Description

Standard property (see Visual Basic Help), sets or returns a value that determines whether graphics methods in Paint events repaint the entire object or only newly exposed areas. Also determines whether the Microsoft Windows operating environment creates a clipping region that excludes nongraphical controls contained by the object. Read-only at run time.

Usage

atx.CLIPCONTROLS = <true/False>

b% = atx.CLIPCONTROLS

Data Type

BOOL

Default

TRUE

CurChar Property

Description

Setting moves the cursor/caret to the nth character within the current paragraph (as defined by the CurPar property), simultaneously determining the starting edge of a select region.

Returns either the current cursor/caret offset into a paragraph (if Select property is False), or the character offset of the starting edge of the select region (if Select property is set True)

Usage

ALLText.CurChar = n%
n% = ALLText.CurChar

Access

	Read	Write
Design time	-	-
Run time	+	+

DataType

Integer

Values

0 <= n% <= Number of Characters in the Paragraph specified by CurPar property

Default

0

Remarks

If out of range, the cursor is moved to the last Character of the Paragraph specified by CurPar property.

CurPar Property

Description

Sets the cursor/caret location to the start of a given paragraph, automatically reseting CurChar to 0. Generally returns the paragraph number of the current cursor/caret location, except when a select region is active (ALLText.Select=True) in which case CurPar returns the paragraph number containing the the start of the select region.

Usage

ALLText.CurPar = iParNum&
iParNum& = ALLText.CurPar

Access

	Read	Write
Design time	-	-
Run time	+	+

DataType

LONG

Values

0 <= iParNum& < ALLText.NumParagraphs

Default

0

Example

```
' Set caret into last Paragraph of the Text
   AllText1.CurPar = ALLText.NumParagraphs
' Set caret into last Character of the Paragraph
' specified above.
   AllText1.CurChar = ATX_MAXCHARS
```

Remarks

If out of range this property is set to last Paragraph of the Text.
Setting CurPar will automatically reset CurChar to 0.

DataChanged Property (HT/Pro Version)

Description

The datachanged property works as per the standard VB DataChanged property, with the exception that it is not automatically set TRUE upon chages to the control content. This discrepancy is expected to be corrected in the next release of ALLText.

Usage

n% = alltext.datachanged

alltext.datachanged = true|false

Access

	R	W
Design	-	-
Runtime	+	+

Data Type

Integer(Boolean)

Values

True - Data is changed

False - Data not changed

DataType Property

Description

Defines how strings, including embedded codes, are formed or interpreted by the ALLText control during I/O in particular, this affects file or database I/O.

Usage

n% = AllText.DataType

AllText.DataType = n%

Access

	R	W
Design	+	+
Runtime	+	+

Data Type

Integer (ENUM)

Values (Constants may be found in file ATX4SAPI.BAS or ATX4HAPI.BAS for HT/Pro version)

0 - ATX_FORMAT_TEXT - Text

1 - ATX_FORMAT_S - AllText special format

2 - ATX_FORMAT_F - AllText full format

3 - ATX_FORMAT_RTF - Rich Text format (HT/PRO edition only)

4 - ATX_FORMAT_FLX - Flexible / User Defined,
Interpret codes based on ExtDataType property (HT/PRO)

Default

0

If Out of Range

A trapable error is generated. The property value is unchanged.

Remarks

The setting ATX_FORMAT_FLX is quite powerful as it allows the code to determine which formatting codes to interpret or include in an output file based upon the setting of the ExtDataType property. See the example included with the description of ExtDataType.

Example

The following two code snippets are equivalent and will save any Selected text to a file Full.ATX

```
atx1.FileName="FULL.ATX"
atx1.DataType=ATX_FORMAT_FLX
atx1.ExtDataType=ATXF_MASK
atx1.FileSave = ATX_IO_FAST

atx1.FileName="FULL.ATX"
atx1.DataType=ATX_FORMAT_F
atx1.FileSave = ATX_IO_FAST
```

DirectScreenOut Property

Description

Determines whether ALLText display text directly (DirectScreenOut = TRUE) or should make use of an internal buffer for preparing text before presenting it on the screen (DirectScreenOut = FALSE).

Visual Basic

ATX.DirectScreenOut = <True/False>

b% = ATX.DirectScreenOut

Default

FALSE

Remarks

Use of an internal buffer prevents blinking of the text string being edited. It also results in smoother output of other objects when using AllText/Pen features. Note that the internal buffer takes additional memory and may result in slower output on some computers.

DocHeight Property

Description

Returns the total height of the text within the control. If greater than the window height, a vertical scroll bar will automatically be activated.

Usage

n&=ALLText.DocHeight

Access

	Read	Write
Design	**-**	**-**
Execution	+	**-**

Data Type

Long (twips)

Values

0 ≤ YTwipsToPixels(n%) ≤ *MaxTextHeight*

Example

```
if AllText1.DocHeight > AllText1.Height then
    print "This text does not fit into window"
    end if
```

DocWidth Property

Description

Changes or returns the width of Text displayed within the control window.
Text is automatically word-wrapped to fit within the specified DocWidth. To turn word wrapping off, set DocWidth to ATX_MAXTEXTWIDTH.

Usage

ALLText.DocWidth = n&
n& = ALLText.DocWidth

Access

	Read	Write
Design	+	+
Execution	+	+

Data Type

Long (twips)

Values

MinTextWidth ≤ XTwipsToPixels(n&) ≤ *MaxTextWidth*

resolution = Screen.TwipsPerPixelX

Default

XTwipsToPixels(400)

Example

```
If AllText1.DocWidth > AllText1.Width-1 then
' Word Wrap wrap
AllText1.DocWidth = AllText1.Width-1
end if
```

Remarks

Attempts to set DocWidth to an out of range value will be ignored by ALLText.

Note that reading the DocWidth property may return a slightly different value than it was set to. This is because DocWidth settings are limited in resolution to the number of Twips per pixel in the horizontal direction.

Upon setting DocWidth, paragraphs are reformatted, and their widths are changed to keep their margin indents the same. DocWidth may never be set to a value less than the maximum sum among all paragraphs of left plus right indents.

The DocWidth property setting does not affect the text wrapping width when printing using the built in printer functions. This is independently set by the parameters of the print functions.

The DocWidth property may be reset by ALLText when loading in a document using FileLoad, or when the control content is changed as a result of repositioning a data control to which the control is bound. In this case ALLText will read in the width of the document as specified in the stored file. You should reset the DocWidth after reading or in the Data.Reposition event if you want to insure a specific Word Wrap width.

DropFileMode Property

Description

The setting of this property determines whether AllText will accept files Dropped from File Manager, triggering DropFile and DropFileStart Events.

Usage

ATX.DropFileMode = <0,1>

e% = ATX.DropFileMode

Settings

DropFileMode = 0	Do not accept files dropped from File Manager
DropFileMode = 1	Accept files dropped from File Manager

Default

1

Data Type

Enum

See Also

DROPFILESTART event

DROPFILE event

ExtDataType Property (HT/Pro only)

Description

The ExtDataType (External Data Type) property determines what RTF data (embedded codes) will be saved with the text during File or Database I/O.

This property can only be set when DataType the DataType property is set to a value of ATX_DATATYPE_RTF (4).

Usage

ATX.ExtDataType = n%

n% = ATX.ExtDataType

Remarks

The values of property ExtDataType are ORd from the following table (Global Constants are defined in the file ATX4hAPI.BAS)

ATX_FNTTBL_MASK = 1	Save font table
ATX_CLRTBL_MASK = 2	Save color table
ATX_PRGSET_MASK = 4	Save text and paragraph formatting codes (alignment, indenting, etc.)
ATX_OLEOBJ_MASK = 8	Expand embedded ole objects with full details necessary for reading back. Without this flag, only the code identifying the location of an OLE object is saved.
ATX_RTFSTD_MASK = 16	Save as real 7 bit RTF with symbol conversion This mask causes AllText to generate an RTF output including initial ("{\rtf1\ansi\deff0\deflang1033") and final ("}") for RTF compatibility. In addition all special symbols such as endash, emdash, non breaking space, etc. will be converted into the standard RTF form. The output generated without this mask cannot be read by foreign RTF readers.
FULLRTF_MASK = 31	Based on logical OR of ATXF_MASK, ATX_CLRTBL_MASK, and ATX_RTFSTD_MASK
ATXS_MASK = 4	Same as ATX_PRGSETMASK, but name intended to show compatibility with ALLText 3 ATX_S format.
ATXF_MASK = 13	Compatible with ALLText 3s ATX_F Format. Based on logical OR of ATXS_MASK, ATX_FNTTBL_MASK, and ATX_OLEOBJ_MASK.

Data Type

Integer

Default

= 0

Example

The following two Visual Basic code snippets both save any selected text to a file, but the first one will not

save the font table in the NOFNTTBL.ATX file.

```
atx1.FileName="NOFNTTBL.ATX"
atx1.DataType=ATX_FORMAT_FLX
atx1.ExtDataType=ATX_PRGSET_MASK or ATX_OLEOBJ_MASK
atx1.FileSave = ATX_IO_FAST

atx1.FileName="FULL.ATX"
atx1.DataType=ATX_FORMAT_FLX
atx1.ExtDataType=ATXF_MASK
atx1.FileSave = ATX_IO_FAST
```

ExtObjParam Property

Description

This property determines which attribute of an embedded external object, such as a picture, is to be accessed via either the ExtObjValue or ExtObjPicture properties.

Usage

ATX.ExtObjParam = s$

s$ = ATX.ExtObjParam

Default

Empty string

Values

The valid values of this property depend upon the external object to be addressed. For Picture objects embedded with the ATXPicDLL, the following values are valid:

Width - Specifies the width of the picture in twips.

Height - Specifies the height of the picture in twips.

Base Line Descent - Specifies an offset relative to the text base line in twips.
(Positive values lower the object relative to the text).

Frame Type - Specifies the type of the frame that will be drawn around the picture.

FrameType% = 0	No frame around the picture
FrameType% = 1	Single line around the picture
FrameType% = 2	Shadow frame
FrameType% = 3	Up 3D frame
FrameType% = 4	Down 3D frame

PictureHandler - Specifies a handle to a picture. The picture can be changed by setting ExtObjPicture to a valid picture.

Example

For any of these objects we could try to set some property. For example if we want to set width for an object we discussed we should write

```
ATX.ExtObjParam = "Width"
ATX.ExtObjValue = str(10000)

ATX.ExtObjParam = "PictureHandler"
ATX.ExtObjPicture = Picture1.Picture
```

Remarks

Note the spaces in the values Frame Type and Base Line Descent

There is no error triggered by setting an invalid value. This is to allow for support of future External Object DLLs.

Data Type

String

See Also

ExtObjPicture -- property

ExtObjValue -- property

ExtObjValue Property

Description

This property sets or returns an attribute of an external object. The attribute accessed is specified by the ExtObjParam property.

If a select region is active, the object accessed is the first object within the select region for which the desired attribute (as specified by ExtObjParam) is valid. If no select region is active, the property accesses the object at the current cursor location.

Usage

ATX.ExtObjValue = s$

s$ = ATX.ExtObjValue

Example

Place a frame around a picture:

```
ALLText.ExtObjParam = "Frame Type"
ALLText.extobjvalue = "1"
```

Remarks

The value should be passed as a string.

Data Type

String

Default

empty String

See Also

ExtObjParam -- property

ExtObjPicture -- property

ExtObjPicture Property

Description

Sets or returns the value of the PICTURE attribute for the first external picture object within a select region, or the external object PICTURE after cursor if no select region is active. The appropriate property name (a string) for the External Object must first be defined in the ExtObjParam property.

Usage

ATX.ExtObjPicture = CTL.picture

CTL.picture = ATX.ExtObjPicture

Example

Embed an empty picture, move the cursor back before the picture and set the picture

```
ATX.OLEObject = atxpic:
ATX.SelStart= ATX.SelStart-1
ATX.ExtObjParam = "PictureHandler"
ATX.ExtObjPicture = Picture1.Picture
```

Data Type

Picture

See Also

ExtObjParam - property

ExtObjValue - property

F2On and F3On Properties

Description

By default, the F2 key will call up the standard windows font dialog box when in runtime mode the ALLText control has the focus. To prevent this action, set F2On to False.

Similarly, the F3 key will initiate printing of the control content when in runtime mode the ALLText control has the focus. To prevent this action, set F3On to False.

Usage

ALLText.F2On = n%
n% = ALLText.F2On

Access

	Read	Write
Design time	+	+
Run time	+	+

Data Type

Logical

Values

True/False

Default

True

FileName Property

Description

In FAST load and save modes, the FileName property determines the Full Path Name of the file read from, or written to by ALLText. The programmer need not open or close the file via VB Code - this is fully automatic.

Usage

path$ = AllText.FileName

AllText.FileName = path$

Access

	R	W
Design	+	+
Runtime	+	+

Data Type

String

Default

""

If Out of Range

Trapable error is generated. Property value is unchanged.

FileLoad Property

Description

Setting FileLoad switches ALLText into state of loading new content at the current cursor position, or replacing content of active select region. The value of FileLoad determines how loading is handled. In standard and low level mode an *ATXGet* event is triggered for each chunk of document being read in. (At the time of this writing, a chunk is defined as up to 2000 characters, this may be subject to change however). In Fast mode, the ATXGet event is not triggered and loading from a file procedes automatically as determined from the *FileName* and *DataType* properties.

Usage

AllText.FileLoad = LoadMode%

LoadStatus% = AllText.FileLoad

Access

	R	W
Design	-	-
Runtime	+	-

Data Type

Integer

Values - (see constants definitions in ATX4sAPI.BAS or ATX4hAPI.BAS for HT/Pro version)

On setting:

ATX_IO_FAST - fast loading mode is activated

ATX_IO_STANDARD - standard loading mode is activated

ATX_IO_LOWLEVEL - low level loading mode is activated

On reading:

ATX_IO_COMPLETED - loading operation is completed successfully.

< 0 - error on loading.

DEFAULT

ATX_IO_COMPLETED

Remarks

The first reading of the FileLoad property after a load operation will return either ATX_IO_COMPLETED, or a negative value error code. Following readings return 0.

Reading a file may change the current DocWidth setting to that of the document loaded.

Example: Fast Loading

```
Sub load_from_file (file_name$, file_format%)
    alltext1.WriteProtect =  -1
    alltext1.FontTableSize = 0   ' adjust font table to remove unused entries
    i% = alltext1.ClearAll           ' empty the control, otherwise will file merge.
    alltext1.DataType = file_format%   ' set the file type
    alltext1.FileName = file_name$       ' set the file name
```

```
    alltext1.FileLoad = ATX_IO_FAST       ' initiate automated loading
    ' Note, the ATXGet event is not generated.
End Sub
```

FileSave Property

Description

Switches ALLText into state of forming strings for saving from **selected** area. In standard and low level mode an *ATXPut* event is generated each time when next AllText line is ready for saving another chunk of information (at this time, the highly technical term 'chunk' may be taken as consisting of up to 2000 characters, this is however subject to change in future versions). In fast mode no ATXPut event is triggered, and saving to a file (see *FileName* property) procedes automatically according to the file format set by the *DataType* property.

Usage

AllText.FileSave = SaveMode%

SaveStatus% = AllText.FileSave

Access

	R	W
Design	-	-
Runtime	+	-

Data Type

Integer

Values

On setting:

ATX_IO_FAST - fast loading mode is activated
ATX_IO_STANDARD - standard loading mode is activated

ATX_IO_LOWLEVEL - low level loading mode is activated

On reading:

ATX_IO_COMPLETED - loading operation is completed successfully.

< 0 - error on loading.

DEFAULT

ATX_IO_COMPLETED

Remarks

If saving is completed with error then first reading of FileLoad property returns negative value error code. Following readings return 0.

Note that ALLText will only save the Selected text region.

Example: FAST SAVING

```
Sub save_to_file (file_name$, file_format%)
    ' Don't allow end user modifications during the save.
      wp% = alltext1.WriteProtect:      alltext1.WriteProtect = 2
    ' select the region of text to save, recalling old select settings
      alltext1.SelStart = 0:            alltext1.SelLength = 2000000


    ' Initiate automated Saving in desired file format
```

```
    ' Note that the ATXPut event is not generated.
        alltext1.DataType = file_format%
        alltext1.FileName = file_name$
        alltext1.FileSave = ATX_IO_FAST

    ' Return to originial configuration
         alltext1.WriteProtect = w%
End Sub
```

FirstLineIndent Property

Description

Sets or returns indent of the first text line in a paragraph from the left edge of the rest of the paragraph. Designed for support of hanging indents.

Usage

ALLText.FirstLineIndent= n&
n& = ALLText.FirstLineIndent

Access

	Read	Write
Design	-	-
Execution	+	+

Data Type

Long (twips)

Default

1

Values

-32000 <= n& <= DocWidth

Notes

The resolution of this property is dependent on the physical device. ALLText holds such information internally in Pixels such that

ATX.FirstLineIndent = Screen.TwipsPerPixelX + 1

is equivalent to

ATX.FirstLineIndent = Screen.TwipsPerPixelX

FontBold Property

Description

Sets the current font to Bold. Any currently selected text is changed to bold. Other details remain unchanged.

Returns TRUE or FALSE depending on current font is Bold or not. For a select region containing text of mixed bold and unbold characters this property returns ATX_UNDEFINED.

Usage

ALLText.FontBold = n%
n% = ALLText.FontBold

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

True	current font is Bold,
False	current font is not Bold.
ATX_UNDEFINED	current font is undefined

Default

False

Example

```
SUB MENU_BOLD_CLICK()
        ALLText1.FontBold=Not ALLText1.FontBold
        End Sub
```

FontColor Property

Description

Set or returns the foreground color for a select region or the current cursor position. Other details remain unchanged. Returns ATX_COLOR_UNDEFINED for regions with mixed color settings.

Usage

ALLText.FontColor = c&
c& = ALLText.FontColor

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Long (RGBcolor)

Values

QBcolor(0) ≤ c ≤ QBcolor(15) or ATX_UNDEFINED

Example

AllText1.FontColor = QBColor(ATX_LT_Blue)

Remarks

If you use VB 3.0 You can also manipulate colors using the constants specified in file CONSTANT.TXT. Otherwise the ATX constants are defined in the constants declaration file shipped with ALLText.

FontFamily Property

Description

Sets or returns the current font family.

The Aim of this property is to help the user to manipulate with fonts reliably across end-user environments with different installed fonts. Generally there is no need to use it directly. But if the VB Developer has no assurance that End-User's computer will have the desired fonts installed, he or she may ensure that the font chosen by Windows will be at least of the same family.

Usage

ALLText.FontFamily = i%
i% = ALLText.FontFamily

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

ATX_DONTCARE	family is chosen by system itself
ATX_ROMAN	Roman family
ATX_SWISS	Swiss family
ATX_MODERN	Modern family
ATX_SCRIPT	Script family
ATX_DECORATIVE	Decorative family

Default

ATX_DONTCARE

Example

AllText1.FontName = "Arial"
AllText1.FontFamily = "Sans Serif"

Remarks

It is recommended to use this property only for reading, to verify the proper family of font has been chose.

FontHidden Property (HT/Pro only)

Description

Specifies whether the currently selected text is assigned a Hidden attribute. Text with the Hidden attribute will be visible only when the ShowHidden property is True. Otherwise such text is not visible.

For a select region containing a mixture of italic and non-italic text this property returns ATX_UNDEFINED.

Usage

ALLText.FontHidden = n%
n% = ALLText.FontHidden

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

True	current font is Hidden,
False	current font is not Hidden.
ATX_UNDEFINED	current font is undefined

Remarks

To retrieve hidden text when ShowHidden is False, OR the SelFType property with 4 and 8, and read either the FText or SelFText property.

Default

FALSE

FontItalic Property

Description

Set current font Italic. Sets any currently selected text to *Italic*. Other details remain unchanged.

Returns True or False depending on current font is Italic or not. For a select region containing a mixture of italic and non-italic text this property returns ATX_UNDEFINED.

Usage

ALLText.FontItalic = n%
n% = ALLText.FontItalic

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

True	current font is Italic,
False	current font is not Italic.
ATX_UNDEFINED	current font is undefined

Default

FALSE

Example

```
i% = AllText1.FontItalic
' Select text
   AllText1.SelToPar = AllText1.CurPar+1
   AllText1.SelToChar = 0

' Set Selected text Italic
   AllText1.FontItalic = TRUE

' Return to previous value.
   AllText1.FontItalic = i%
```

FontIndex Property

Description

The FontIndex property applies font characteristics as specified by a given entry of the font table to the current character location or to a select region.

If a selection region is active, setting this property will change all font properties (FontName, FontSize, FontFamily, FontWidth, FontUnderline, FontItalics, FontBold, and FontStrikeThrough) of the selected text to those specified by the designated entry from the Font Table. It will not affect the FontColor property of the text which is separately controlled.

Reading this property value will cause the ALLText control to search for the current font in Font Table. If the current font is not presented in Font Table, it is automatically added and the assigned index is returned.

Usage

ALLText.FontIndex = n%
n% = ALLText.FontIndex

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

0 <= n% < ALLText.FontTableSize

Default

0

Example 1

```
' Copy the text characteristic formatting from one location
 ' and apply it to text being pasted in another location.

' 1) read current font from one location
 x=ALLText1.FontIndex

' 2) Create an embedded code to replicate the font characteristics.
 FormatCode = "\f" &Mid$( Str$(x), 2 ) & " "

' 3) Move cursor to another location
 ALLText1.CurChar=532

' 4) paste in some string with the same font characteristics as the initial string.
 ALLText1.SeLFText = FormatCode & "some string"
```

Example 2

```
' Identify the components which make up the a given entry in a font table.

' 1) set the current font to the font table entry in question
 ALLText1.FontIndex = X
```

```
' 2) Inquire of each individual font characteristic property.
 A=ATX.FontBold: B=ATX.FontItalic: C=ATX.FontName .....
```

Remarks

Assignment statements setting FontIndex to an out of range value are ignored, the current font remaining unchanged.

FontName Property

Description

Sets or returns the current font name - ie: the font applied to the current select region or cursor location . An empty string, "", is returned if an active select region contains text of mixed font name.

Usage

ALLText.FontName = s$
s$ = ALLText.FontName

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

String

Default

system

Example

Remarks

If the control is empty this property is set to "SYSTEM." Otherwise this property designates the font placed at the start of Text.

Unlike the standard Textbox, setting FontName does not change the font for the entire textbox, only for text at the current location, or for currently selected text.

FontShadow Property

Description

Sets or returns the Shadowing characteristics (as an integer value) for text at the current caret position or within a selected text region. ALLText supports a unique shadow characteristic echoing up to two copies of a character, each offset by some distance (to the upper left and lower right of the base positioning) and each with a different color.

Manipulation of the shadowing requires use of API Functions Make_Shadow and Get_Shadow. A value of FALSE may however be readily used as indicating the absense of shadowing. Likewise reading a value of ATX_UNDEFINED indicates a select region with mixed shadowing characteristics.

Usage

ALLText.FontShadow= n%
n% = ALLText.FontShadow

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

Specially organized values are used to define the shadowing. False indicates no shadowing, Other values may be composed or interpreted using the Make_Shadow and Get_Shadow funcitons.

Default

FALSE

Example

AllText1.FontShadow = make_shadow(1,1,0,15)

FontSize Property

Description

Sets or returns the current font size. For a select region with fonts of mixed sizes this property returns ATX_UNDEFINED.

Usage

ALLText.FontSize = n%
n% = ALLText.FontSize

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer (points)

Values

0..n%

Default

ATX_DEFAULT

Remarks

Actually designates font height.

At the start of program if Text is empty this property is set to ATX_DEFAULT else this property designates font is placed at the start of Text.

FontStrike Property

Description

Sets current font to Strikethrough. Strikes-through any currently selected text. Returns True or False depending on current font is Strikethrough or not.

If some text is selected and its font is either not set, or of mixed character, this property returns ATX_UNDEFINED.

Usage

ALLText.FontStrike = n%
n% = ALLText.FontStrike

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

True	current font is Strikethrough,
False	current font is not Strikethrough.
ATX_UNDEFINED	current font is undefined

Default

False

FontSubSup Property

Description

Sets super/subscript attributes for text entered at current cursor location, or within select region, to super (if >0) or sub (if <0) - script. Other details remain unchanged. Returns super/subscript value for current font.

Usage

AllText.FontSubSup = n%
n% = AllText.FontSubSup

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer (half points)

Values

-500 < n% < 500	- NOTE: only EVEN numbers are available.
n% > 0	- current font is superscript,
n% < 0	- current font is subscript.
ATX_UNDEFINED	- selected text contains various super-subscript degrees, possibly including plain text.

Default

0

Remarks

If out of range, a trappable error is generated.

FontTableSize Property

Description

Returns number of entries in the Font Table.
Setting to 0 updates the Font Table, removing fonts not in use.

Usage

n% = ALLText.FontTableSize
ALLText.FontTableSize = 0

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

1 <= n% <= ~3,500

Default

1

Example

```
n% = ALLText.FontTableSize
ALLText.FontTableSize = 0
m% = ALLText.FontTableSize
if n%<>m% then
        print "Unused Fonts have been removed"
        End If
```

Remarks

There is always at least one font in the Font Table. This font is entry number 0, the "System" font of default size. It is not necessary to save and load this font.

Setting FontTableSize to 0 removes any fonts not in use. Upon the next reading of FontTableSize, the value read will reflect the number of fonts left in the table. Setting FontTableSize to any other value will trigger a trappable error.

FontUnder Property

Description

Sets or returns the underline style at the current caret position or selected text region. ALLText supports a very flexible underlining scheme whereby various styles of underlining may be set (Color, single or double, solid, dashed or dotted).

Note that settings of True/False may be used for the simple case of a single black underline. More complex styles are associated with other values. ALLText API Functions, *Make_Underline* and *Get_Underline* have been provided to handle such values.

Usage

n% = AllText.FontUnder

AllText.FontUnder = n%

x%=Get_Underline(ALLText.FontUnder, shape%, dblflag%, ncolor%)

ALLText.FontUnder=Make_Underline(shape%, dblflag%, ncolor%)

Access

	R	**W**
Design	**-**	**-**
Runtime	+	+

Data Type

Integer

Values

Defined by value set returned by make_underline function.

True corresponds to a single black solid underline.

False corresponds to no underlining.

Default

0 - FALSE

If Out of Range

VB Overflow error, property value remains unchanged.

FontWidth Property

Description

Sets or returns the current font width at the cursor location, or within an active select region. A value of ATX_UNDEFINED is returned if an active select region contains text of mixed width.

The FontWidth property is intended only to serve for support of special effects. Usually the value of this property is best left as ATX_DEFAULT (0) allowing the system to choose the actual width of the font itself. Use this property in those rare situations when you need to achieve high narrow characters.

Usage

ALLText.FontWidth = n%
n% = ALLText.FontWidth

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer (points)

Values

0 ..ATX_MAXFONTSIZE

Default

ATX_DEFAULT

Remarks

This property is intended only to serve for support of special effects. Usually the value of this property is best left as ATX_DEFAULT allowing the system to choose the actual width of the font itself. Use this property in those rare situations where you need to achieve narrow or wide characters.

FooterFText Property (HT/Pro only)

Description

Contains document Footer that set by the user or read from document file.

Access

	Read	Write
Design time	-	-
Run time	+	+

Values

Reading

```
 S$ = atx.FooterFText(0)     'returns Footer for document
S$ = atx.FooterFText(1)     'returns Footer for left (even) pages
S$ = atx.FooterFText(2)     'returns Footer for right (odd) pages
S$ = atx.FooterFText(3)     'returns Footer for first page
```

Setting:

atx.FooterFText(n%) = s$

Set footer for the corresponding page (Doc, Left pages, Right pages or First page)

Example

See Example in PRINT\tst.mak file.

Constants (See ATX4hAPI.bas)

```
Global Const ATX_FOOTER_SIMPLE = 0
Global Const ATX_FOOTER_LEFT     = 1 (even pages)
Global Const ATX_FOOTER_RIGHT    = 2 (odd pages)
Global Const ATX_FOOTER_FIRST    = 3
```

FormatPaste Property

Description

The Format Paste property determines whether ALLText includes and interprets font formatting codes when pasting text from the clipboard.

Usage

ALLText.FormatPaste = n%

n% = ALLText.FormatPaste

Access

	Read	Write
Design time	+	+
Run time	+	+

Data Type

Integer (points)

Values

True, FormatPaste = -1	Any \cf or \f strings embedded in the clipboard text will be interpreted as formatting codes when pasting into ALLText.
Off, FormatPaste=0	ALLText will treat any embedded formatting code as simple ASCII text during a paste operation. The pasted text will take on the properties set for the current cursor position into which the paste occurs.

Defaul

True

FText Property

Description

Sets or returns the first 30,000 characters of an AllText control as a string including or interpretting embedded codes as defined by the SelFType property. If the text length of the control exceeds 30,000 characters, then only the first 30,000 characters are returned.

Usage

ATX.FText = s$

s$ = ATX.FText

Access

	Read	Write
Design time	-	-
Run time	+	+

Remarks

Note that the formatting codes included in, or interpretted by FText are determined by the setting of the SelFType property. Set SelFType to ATXS_Mask or ATX_FMask for compatibility with ALLText 3.

See Also

SelFType - property

Text - property

HeaderFText Property (HT/Pro only)

Description

The HeaderFText property contains a formatted header text string for a document.

Note that this is a list property, headers can be defined for WholeDocument, OddPages, EvenPages, FirstPage.

Access

	Read	Write
Design time	-	-
Run time	+	+

Values

Reading: Get's the formatted header string

```
S$ = atx.HeaderFText(0)     'returns header for document
S$ = atx.HeaderFText(1)     'returns header for left pages
S$ = atx.HeaderFText(2)     'returns header for right pages
S$ = atx.HeaderFText(3)     'returns header for first page
```

Setting: Set the formatted header string

```
atx.HeaderFText(n%) = s$      Set header string for the corresponding page
( Doc, Left pages, Right pages or First page)
```

Example:

See Example in PrintHF\tst.mak file.

Constants (See ATX4hAPI.bas)

```
Global Const ATX_HEADER_SIMPLE = 0
Global Const ATX_HEADER_LEFT     = 1 (even pages)
Global Const ATX_HEADER_RIGHT    = 2 (odd pages)
Global Const ATX_HEADER_FIRST    = 3
```

HTAG Property (HT/Pro Version)

Description

The HTag property associates an interger valued Hypertext Tag with a string within ALLText. It is set and read in a manner analagous to Font properties (FontBold, etc).

Setting AllText.Htag=x& assigns a tag value of x& to the selected area. But if AllText.Select is FALSE then tag x& is assigned to current caret position - characters then inserted into this caret position are of tag x&.

Reading x&=AllText.Htag results in:

> If no text is selected and current position falls into some tag area then tagged area is automatically selected, tag number is returned. If current position is out of any tag then nothing happens, return 0.
>
> If some text segment is selected and it fully falls into some tagged area then selection is automatically extended up to tagged area, tag number is returned. If selected segment occupies more then tagged area or more than one tag then nothing happens, property returns ATX_TAG_UNDEFINED.

Access

Runtime only, Read/Write

Remarks

Setting a HTag value other than 0 creates a Hypertext region. The mouse displayed over such a region is controlled via the *MouseHPointer* property. Moving into or out of such a region triggers the *ATXChange* event.

Specific Tagged regions may be found using the *Find_HTag* function

Setting a HTag value of 0 undoes these effects.

HTagL & NTagL Properties - (HT/Pro edition Only)

Description:

HTagL and NTagL perform the same function as Htag or NTag, but on reading they return the value of the previous character tag (character before caret) rather than the tag of the character following the caret. Write operations are the same as for HTag.

Usage

ATX.htagl = l&

l& = ATX.htagl

Remarks

HTagL and NTagL were introduced for the following reason. When clicking on a character, the caret is moved to the start of that character. You can then read the HTag property to determine its tag value. If you start typing now however, new text entered will take on the characteristics of the prior character (HTag value as well as other characteristics such as FontName and FontColor). While FontName and FontColor always return the value of the prior character (what you would now be typing in) HTag returns the value of the next character (the one you clicked).

Data Type

Long

Defaul =0

LeftIndent Property

Description

Sets or returns Left indent for Paragraph

Usage

ALLText.LeftIndent= n&
n& = ALLText.LeftIndent

Access

	Read	Write
Design	-	-
Execution	+	+

Data Type

Long (twips)

Values

0 ≤ XTwipsToPixels(n&) ≤ (ALLText.DocWidth-ALLText.RightIndent)

Default

0

Example

```
AllText1.LeftIndent=    .25 * 1440      'set to 1/4 inch
```

Remarks

The resolution of this property is dependent on the physical device. ALLText holds such information internally in Pixels such that

```
ATX.LeftIndent = Screen.TwipsPerPixelX + 1
```

is equivalent to

```
ATX.LeftIndent = Screen.TwipsPerPixelX
```

LineNumber Property

Description

Defines the number of lines from the beginning of text to the current caret location (see SelStart, CurChar, CurPar).

Syntax

ATX.LineNumber = l&

l& = ATX.LineNumber

Remarks

On setting this property the caret is moved to the appropriate line of text and SelStart and other Selection properties are set accordingly.

LineSpacing Property

Description

Sets or returns the linespacing (in Twips) used within a paragraph.

Usage

ALLText.LineSpacing = n&
n& = ALLText.LineSpacing

Access

	Read	Write
Design	-	-
Execution	+	+

Data Type

Long (twips)

Default

0

Values

0 <= n% <= PixelsToTwips(ATX_MAXSPACING)

Out of Range

Trapable error is generated, Value remains unchanged

Example

```
AllText1.LineSpacing = .25 * 1440       'set to 1/4 inch
```

MouseHPointer (HT/Pro Version)

Description

Determines which mouse pointer is displayed when the mouse is over an area marked by a non-zero Htag value.

Usage

AllText.MouseHPointer = n%
n% = AllText.MouseHPointer

Access

	Read	Write
Design	+	+
Execution	+	+

Data Type

Integer

Values

0 < n% < 11 standard mouse pointers
(the same as for MousePointer property.)

11 < n% < *ATX_MAXMOUSEHPOINTER* nonstandard mouse pointers
(used by AllText HT/Pro only)

Default

0

Remarks

Note: If out of range then nothing happens, error message is generated.

Constants presenting property values are specified in file ATX4sAPI.BAS or ATX4hAPI.BAS For HT/PRO edition

Setting MouseHPointer to a value different than the MousePointer property setting, enables the triggering of the *ATXChange* event upon the movement of the mouse into or out of a Hypertext tagged region.

NewHtag Property (HT/Pro Version)

Description

The NewHtag property is a read-only property returning the lowest possible tag value not yet in use within the current content of the control.

Usage

n& = AllText.NewHtag

Access

	Read	Write
Design	**-**	**-**
Execution	+	**-**

Data Type

Long

Values

1 < n& < *ATX_MAXNEWHTAG*

Default

1

Example

```
Alltext1.CurChar = 0
Alltext1.SelToChar=0
for i%=0 to Alltext1.NumOfParagraphs - 1
Alltext1.CurPar = i%
Alltext1.SelToPar = i+1
t& = Alltext1.NewHtag
Alltext1.Htag = t&
Next
```

Remarks

If all available tags are already used in the Text than returns 0.

Note: the limitation on variety of tags per Text by value *ATX_MAXNEWHTAG* affects only on this property. If programmer is not satisfied this limitation he can not use NewHtag property without any restrictions but then he shouldorganize his own control of what tags are in use.

NTag Property (HT/Pro edition only)

Description:

NTag sets or returns a long integer tag value associated with the following character or selected text. NTag functions exactly like HTag except that text tagged with an NTag will not trigger a change in the mousepointer.

Syntax

ATX.NTag = l&

l& = ATX.Ntag

Remarks

NTag was introduced to allow invisible tagging of characters without triggering a change in the mouse pointer. NTagged text may also be added to the content using the embedded code \ATXnt#### and ending with \ATXnt0

Example

ALLText.NTag = Some of this \ATXnt5 text is tagged \ATXnt0.

Data Type

Long

Default

0

NumParagraphs Property

Description

Returns number of Carriage Return/Line Feed delimited paragraphs in the Text.

Usage

ALLText.NumParagraphs = n&
n& = ALLText.NumParagraphs

Access

	Read	Write
Design	-	-
Execution	+	-

Data Type

integer

Values

0 ≤ n& ≤ *MaxNumberOfParagraphs*

OLEcode Property (HT/Pro edition Only)

Description

Defines what character code will be generated as a place holder for external objects when READING the Text or SelText properties. If this is set to 0 then no character will be generated.

Usage

n = AllText.OLEcode

AllText.OLEcode = n

Access

	Read	Write
Design	-	-
Execution	+	-

Data Type

Integer

Values

0 <= n <= 254

Defaul

@

If Out of Range

generates trappable error

OLEObject Property (HT/Pro version)

Description

Setting this property causes ALLText to call the specified External Object DLL to create and embed an External Object at the current caret location, overwriting any currently selected portion of the document. The type of object created is specified by the assigned string, ObjName$.

When read, this property returns a string indicating the name of the External Object DLL (ATXPic, ATXOLE,...) supporting the object and an object number. If the object is an OLE object and the OLEVerb is set to anything other than ATX_Verb_Undefined, the OLE Server will be launched to executes this verb (ATX.OLEVERB) on this object. Note that this is true only for objects found immediately after caret, *or an object marked with ink grabber frame when using ALLText/Pen*.

Access

	R	**W**
Design	**-**	**-**
Runtime	+	+

Usage ' (see further notes below under Values)

AllText.OleObject = s$

s$ = AllText.OleObject

Data Type

STRING

Values

On reading, ALLText returns a string identifying the Exteral Object DLL and the specific instance of the external object, ex: "ATXOLE ATXObj. #00001" or "ATXPic 0006"

When setting, the string s$ may be one of several forms depending upon the type of external object being embedded.

Ole Object - ALLText.OLEObject = OleServerName & "*" & FileName.

Ole Object - ALLText.OLEObject = External Object DLL Name & : & FileName

For OLE Objects, s$ should be of the form : [ServerName$]& [* & OleDocName$]

ServerName$ - the name of an existing OLE server to be called for object embedding.
(If left out, ALLText will attempt to identify an OLE server based on the entry in the Windows Registration database)

*OleDocName$ - the full path of the file to embed as an OLE object
(if left out, the OLE server will be called to create a new object)

For embedded pictures, s$ should be of the form : "ATXPic: & ObjectFileName$

where ObjectFileName$ is the name of a valid image file (BMP, WMF, JPG, DIB or JPG).
(if left out, a blank picture object will be inserted as a placeholder)

Default

""

If Out of Range

Trapable error is generated, nothing happens.

Examples

```
ATX.OLEObject = "ATXPIC:\Fred.BMP      'Embeds a picture of Fred using the ATXPIC DLL
ATX.OLEObject = "ATXPIC:               'Creates an empty picture object
ATX.OLEObject = "Pbrush*bfly.bmp"      'load bfly.bmp as an OLE Object using Pbrush as OLE server
ATX.OLEObject = "*bfly.bmp"            'load bfly.bmp using default OLE server
ATX.OLEObject="WordDocument*C:\Sample.Doc"  ' Insert a word document
ATX.OLEObject = "Excelworksheet"       'Creating and inserting a NEW OLE object using Excel
```

Remarks

The name of this property is a hold over from earlier versions of ALLText. In truth, the OLEObject property only initiates an OLE Link when calling the ATXOLE.DLL External Object DLL. Other other external object libraries do not rely upon of OLE technology.

For OLE Objects, The Server Name should be present in the [embedding] section of the WIN.INI file.

Setting the OLEObject property to embed an OLE object without providing the name of a datafile to be embedded, launches the OLE server to create a new object. After exiting the server the object will be inserted at the current caret position.

OLEVerb Property (HT/Pro version)

Description

The OleVerb property specifies an operation to perform when an OLE object is activated by double click or by the reading of the OleObject property when the cursor is position at an OLE object or an OLE object is selected.

Usage

AllText.OleVerb = 1

n% = AllText.OleVerb

Access

	R	W
Design	+	+
Runtime	+	+

Data Type

Integer

Values

Depends on the OLE server.

OleVerb = ATX_VERB_UNDEFINED is special value. Objects with this setting not trigger their server application in response to a double click.

See OleObject property description for details.

Default

0

OverType Property

Description

The OverType property determines whether keyboard entry overwrites the current control contents or inserts new text at the current cursor position.

Usage

ALLText.OverType = n%
n% = ALLText.OverType

Access

	Read	Write
Design time	+	+
Run time	+	+

Data Type

Integer (points)

Values

True/False

Default

False

PageHeight Property

Description

Determines the height of the page. If the value of the PageHeight property is greater than maximum value for the current printer it is set to the maximum value supported by the current printer.

Usage

ATX.PageHeight = l&

l& = ATX.PageHeight

Default

MAX_LONG

Remarks

If text was not formatted using the printer device context (PrinterDC=False), the value of this property is equal to MAX_LONG.

PageLineNumber Property

Description

Determines the cursor position in terms of a given line on the current page. The number of the first line on a page is equal to 0.

Usage

ATX.PageLineNumber = l&

l& = ATX.PageLineNumber

Remarks

If text was not formatted using the printer device context (PrinterDC=False), AllText assumes that the content of the control occupies a single page whose length is equal to **MAX_LONG.** In this case the value of PageLineNumber property is equal to number of lines from the start of the document to the current cursor position (ie the LineNumber property).

On setting this property the caret is positioned to the specified line number on the current page (see PageNumber property).

PageNumber Property

Description

Sets or Returns the current page number. Page numbering begins from 0.

Usage

ATX.PageNumber = n%

n% = ATX.PageNumber

Remarks

If text is not formatted using the printer device context (PrintDC=FALSE) , AllText assumes that the contents of the control occupy a single page with length equal to **MAX_LONG.** In this case, the value of the PageNumber property is always equal to zero.

Setting this property repositions the caret to the specified page number.

PrinterDC Property

Description

Determines whether ALLText uses the Printer Device Context for formatting text on the screen. If this property is set to TRUE the hard copy output and the display will be the same (True WYSIWYG). If set to FALSE, ALLText uses the proper device context for the screen resulting in an enhanced on-screen presentation. (this may cause the on-screen word wrapping to be different from printed word wrapping

Usage

ATX.PrinterDC = False

b% = ATX.PrinterDC

Defaul

FALSE

Data Type

Boolean

Remarks

The following properties are affected by setting of the PrinterDC property.

PageHeight - always set to Max_Long, if PrinterDC = False

PageLineNumber - is equal to LineNumber, if PrinterDC = False

PageNumber - is set to 0 , if PrinterDC = False

PTag (HT/Pro edition Only)

Description

A long integer tag value associated with the current paragraph or selected paragraphs.

Usage

ATX.PTag = l&

l& = ATX.Ptag

Remarks

PTag was introduced to allow invisible tagging of an entire paragraph. (It may be useful in building a style sheet.)

Data Type

Long

See Also

Default =0

RightIndent Property

Description

Sets or returns Right indent for Paragraph being designed.

Usage

ALLText.RightIndent= n&

n& = ALLText.RightIndent

Access

	Read	Write
Design	-	-
Execution	+	+

Data Type

Long (twips)

Values

0 ≤ XTwipsToPixels(n&) ≤ (ALLText.DocWidth - ALLText.LeftIndent)

Default

0

Example

```
AllText1.RightIndent = .25 * 1440       'set to 1/4 inch
```

Remarks

The resolution of this property is dependent on the physical device. ALLText holds such information internally in Pixels such that

```
ATX.LeftIndent = Screen.TwipsPerPixelX + 1
```

is equivalent to

```
ATX.LeftIndent = Screen.TwipsPerPixelX
```

ScrollBarH and ScrollBarV properties

Description

Sets the presentation of the horizontal and vertical scroll bars as either Automatic (ALLText displays the scroll bar when text does not fit in the windows), Invisible, or Visible.

Usage

AllText1.ScrollBarH = n%AllText1.ScrollBarV = n%
n% = ALLText1.ScrollBarH n% = ALLText1.ScrollBarV

Access

	Read	Write
Design	+	+
Execution	+	+

Data Type

Integer

Values

0- Auto	Scrollbar appears as and when necessary
1 - Not Show	Scrollbar is not shown
2 - Show	Scrollbar is shown

Default

Auto

ScrollHorz and ScrollVert Properties

Description

Scrolls the text without affecting the current cursor position. Changes or returns difference between the document's left margin and the left edge of the control window (ScrollHorz), or the documment's upper margin and the control window's upper edge (ScrollVert).

Usage

ALLText.ScrollHorz= x& ALLText.ScrollVert= y&
x& = ALLText.ScrollHorz y& = ALLText.ScrollVert

Access

	Read	Write
Design	+	+
Execution	+	+

Data Type

Long (twips)

Values

0 ≤ XTwipsToPixels(x&) ≤ *MaxTextWidth*
0 ≤ YTwipsToPixels(y&) ≤ *MaxTextHeight*

Default

0

Example

```
Sub Command1.Click()
    AllText1.ScrollHorz = 0
    AllText1.ScrollVert = 0
End Sub
```

Remarks

The caret position within the Text is not changed

If out of range then these properties are reset to 0.

Select Property

Description

The Select property may be used as a flag. It returns a value of TRUE when a select region is active and FALSE if no region is currently selected.

The property may also be set in order to turn off a select region. On setting Select to 0 caret goes to the start of selected area and text is unselected. On setting Select to 1 the caret goes to the end of the selected area and after that text is unmarked.

Usage

n% = ALLText.Select

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

TRUE	Some text is selected
FALSE	No text is currently selected

Example

```
if AllText1.Select<>TRUE then
    AllText1.SelToPar = AllText1.CurPar+1
    AllText1.SelToChar = 0
    end if
```

Example

Select will be turned off whenever the SelLength has a value of 0. Examples SelSTart=SelStart+SelLength, or setting CurPar = SelToPar and CurChar=SelToChar.

SelFText Property

Description

The SelFText Property is ALLText's formatted version of the SelText property. It may be similarly used with the exception that reading of SelFText will include embedded formatting codes, and setting of SelFText will interpret any embedded formatting codes.

SelFText replaces selected text by **s$** if Select property is True or inserts text specified by **s$** into caret position otherwise. In either instance, any embedded string sequences which may be interpreted as formatting codes such as font and color information codes are so interpreted

Returns string of characters containing the current selected text including any embedded formatting codes.

Note that the formatting codes included in, or interpretted by SelFText are determined by the setting of the SelFType property. Set SelFType to ATXS_Mask or ATX_FMask for compatibility with ALLText 3.

Usage

ALLText.SelFText = s$
s$ = ALLText.SelFText

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

String

Default

""

Example

Some_String$ = AllText1.SelFText

Remarks

Use SelText if you do not want formatting characters interpretted or included in the returned string.

Note that after setting the SelFText property, the SelLength is NOT reset to 0, but rather is set to include all the newly inserted text. (this is different from ALLText 1.x and 2.x versions). You may wish to set Select = False, or SelLength =0 after setting SelFText.

SelFType Property

Description

Determines what RTF data (embedded formatting codes) will be included when reading either SelFText or FText properties. Also determines how RTF strings will be interpretted by AllText when setting either of these properties.

Visual Basic

ATX.SelFType = n%

n% = ATX.SelFType

Visual C

Data Type

Integer

Remarks

Values for SelFType may be constructed by ORg elements from the following table. (Global constants are defined in ATX4hAPI.BAS and ATX4sAPI.BAS

ATX_FNTTBL_MASK = 1	Include font table
ATX_CLRTBL_MASK = 2	Include color table
ATX_PRGSET_MASK = 4	Save text and paragraph settings (alignment, indenting, etc.)
ATX_OLEOBJ_MASK = 8	Expand embedded ole objects with full description for proper handling. If this mask is absent then OLE object is presented only as a code \ATXoNX, where X is the OLE object number within the AllText window.
ATX_RTFSTD_MASK = 16	Save as real 7 bit RTF with symbol conversion This mask causes AllText to generate an RTF output including initial ("{\rtf1\ansi\deff0\deflang1033") and final ("}") for RTF compatibility. In addition all special symbols such as endash, emdash, non breaking space, etc. will be converted into the standard RTF form. The output generated without this mask cannot be read by foreign RTF readers.
FULLRTF_MASK = 31	Based on logical OR of ATXF_MASK, ATX_CLRTBL_MASK, and ATX_RTFSTD_MASK
ATXS_MASK = 4	Same as ATX_PRGSETMASK, but name intended to show compatibility with ALLText 3 ATX_S format.
ATXF_MASK = 13	Compatible with ALLText 3s ATX_F Format. Based on logical OR of ATXS_MASK, ATX_FNTTBL_MASK, and ATX_OLEOBJ_MASK.

If you set the value of the SelFType property to 0 only plain text will be returned by the SelFText and FText

properties and all RTF embedded codes will be treated as simple ascii text when setting either property.

Note that the formatting codes included in, or interpretted by SelFText are determined by the setting of the SelFType property. Set SelFType to ATXS_Mask or ATX_FMask for compatibility with ALLText 3.

See Also

ExtDataType -- property

SelFText -- property

FText - property

Defaul

ATXS_MASK

SelLength Property

Description

The SelLength property may be used to read or set the length of a select region. It differs from the standard SelLength property primarily in its affect on the properties SelToChar and SelToPar

Usage

ALLText.SelLength = n%
n% = ALLText.SelLength

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

0 <= n% <= ATX_MaxParag*ATX_MaxChar

Default

0

Example

AllText1.CurPar = 0
AllText1.CurChar = 0
AllText1.SelLength = 100

Remarks

ALLtext Copy, Cut, and Paste operations are limited to 32,000 characters. Attempting larger clipboard manipulations will generate a trappable VB error.

Setting SelLength will automatically update the SelToPar and SelToChar properties, likewise settting SelToPar and SelToChar will automatically cause SelLength to change. Setting SelLength to 0 automatically resets the Select property to FALSE.

Use of the SelToPar and SelToChar properties is faster than setting SelLength.

SelStart Property

Description

The SelStart property sets or returns the current cursor/caret position as defined by character count offset from the beginning of the text content. Setting SelStart automatically resets the CurPar and CurChar properties.

Usage

AllText.SelStart = n&

n& = AllText.SelStart

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Long

Values

0 < n& <ATX_MAXPARAG * ATX_MAXCHARS

n& <= SelStart+SelLength

Default

0

Example

```
' Text property emulation
SelStart = 0
SelLength = ATX_MAXPARAG * ATX_MAXCHARS
s$ = SelText
```

Remarks

The SelStart property should never be set to a value greater than SelStart + SelLength. Incorrectly setting SelStart to a larger value will result in a trappable error. To always avoid such errors, you may set Select to False before moving the cursor with SelStart.

Note: SelStart considers the end of a paragraph as two characters. (chr$(13) & Chr$(10))

Note: If n& > ATX_MAXPARAG * ATX_MAXCHARS the caret is positioned into last position of the Text.

If n& < 0 then caret is positioned into first position of the Text.

Setting SelStart to the end of the select region (as specified by SelStart+SelLength or SelToPar and SelToChar) results in a SelLength value of 0 and Select being set to FALSE.

Use of the CurPar and CurChar properties is faster than SelStart.

SelText Property

Description

ALLText's SelText property generally works as per the standard text box with a few exceptions related largely to the formatted nature of ALLText.

Setting SelText replaces selected text by **s$** if Select property is True or inserts text specified by **s$** into caret position otherwise. In either instance, embedded formatting are not interpreted but are treated as standard text.

Reading SelText returns a string of characters containing the current selected text, without including any font and color control information.

Usage

ALLText.SelText = s$
s$ = ALLText.SelText

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

String

Default

""

Remarks

Use SelFText if you want to insert formatted text.

Note that after setting the SelText property, the Select region is not reset (this is different from ALLText 1.x and 2.x versions). This is useful if you then wish to set the formatting for newly entered text using FontName, FontSize, FontColor, LeftIndent, etc.

SelToChar Property

Description

Defines and returns the final edge of the select region as the nth Character within Paragraph specified by the SelToPar property.

Usage

ALLText.SelToChar = n%
n% = ALLText.SelToChar

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Values

0 <= n% <= Number of Characters in the Paragraph specified by CurPar property

Default

0

Remarks

If out of range this property is set to last Character of the Paragraph specified by SelToPar property. This behavior may be usefully employed: setting SelToChar to ATX_MAXCHARS will set SelToChar to the last character in the paragraph.

This property is reset to 0 by setting SelToPar.

SelToPar Property

Description

Sets the paragraph number designating final edge of selected selection region., automatically resetting SelToChar property to 0, and setting SelLength as appropriate.
Returns the paragraph number of the final edge of a select region.

Usage

ALLText.SelToPar = n&
n& = ALLText.SelToPar

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Long

Values

0 <= n& < ALLText.NumParagraphs

Default

0

Example

```
' get number of last paragraph of selected text.
   i& = AllText1.SelToPar
' get number of last character of selected text.
   j& = AllText1.SelToChar
```

Remarks

If out of range, this property is set to first (if<0) or last (if>0) Paragraph of the Text.

ShowHidden Property (HT/Pro only)

Description

Specifies whether ALLText displays hidden text (see FontHidden Property). Text with the Hidden attribute wil be visible only when the ShowHidden property is True. Otherwise such text is not visible.

Usage

ALLText.ShowHidden = True / False

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer

Default

FALSE

TabAdd Property (HT/Pro only)

Description

The TabAdd property adds a new absolute tab definition for the current or selected paragraphs using the current default tab alignment (see *TabAlignment*).

Usage

ALLText.TabAdd = newtabloc&

Access

	R	W
Design	-	-
Runtime	-	+

Data Type

long

Values

0 to maxlong

If Out of Range

If < 0 trapable error is generated.

If distance between new tab location and nearest existing tab < 16 pixels
then nearest tab changes its location to new tab location instead setting of new tab.
No error condition is generated.

TabAlignment Property (HT/Pro only)

Description

The TabAlignment property is used to set or read the alignment of any absolute tab positions defined for a paragraph. The property holds an array of tab alignments, indexed from 0 to TabCount-1, for the selected paragraphs. Each element in the array corresponds to one of the absolute tab positions for the paragraph.

Using TabAlignment, pointing to a negatively indexed array element, sets or reads the default alignment for the *TabAdd* property but does not affect any existing tab settings.

Usage

alignment& = alltext.TabAlignment(index%)

alltext.TabAlignment(index%) = alignment&

Access

	R	W
Design	-	-
Runtime	+	+

Data Type

integer

Values

ATX_TAB_LEFT	left aligned tab
ATX_TAB_CENTRED	centered tab
ATX_TAB_RIGHT	right tab
ATX_TAB_UNDEFINED	undefined tab, value returned upon reading TabAlignment while several paragraphs are selected having different tab alignments.

Default

ATX_TAB_LEFT

If Out of Range

A trapable error is generated.

Example

```
'change first tab alignment to right, set default to right also
alltext.TabAlignment(0) = ATX_TAB_RIGHT
'add new right tab
alltext.TabAdd = 3000
' add new centered tab
alltext.TabAlignment(-1) = ATX_TAB_CENTRED 'sets default alignment for the TabAdd property)
alltext.TabAdd = 2000
```

Remarks

(Constants may be found in file ATX4sAPI.BAS or ATX4hAPI.BAS)

TabCount Property (HT/Pro only)

Description

Returns the number of absolute tab locations set in the selected or current paragraph.

Setting this property to 0 clears all tabs in selected paragraphs of text.

Usage

nt% = alltext.TabCount

alltext.TabCount = 0

Access

	R	W
Design	-	-
Runtime	+	+

Data Type

integer

Values

0 to 2000

Default

0

If Out of Range

A trappable error is generated if an attempt is made to set TabCount to a value other than 0.

TabDel Property (HT/Pro only)

Description

Setting TabDel will remove the absolute Tab definition for the nearest Tab location, but only if such a tab setting exists within 16 pixels of the specified value.

Usage

alltext.TabDel = tabloc&

Access

	R	W
Design	-	-
Runtime	-	+

Data Type

long

Values

0 to maxlong

If Out of Range

If < 0 then trapable error is generated.

If there is no tab closer than 16 pixels no tabs settings are deleted - Note that in this case there is NO trappable error is generated.

TabEnabled Property

Description

Determines how ALLText handles the Tab Key. With TabEnabled set to a value of True, ALLText accepts the Tab character for insertion into the text. With TabEnabled set to a value of FALSE, ALLText treats the TabKey as a focus change mechanism similar to other controls.

Usage

ALLText.TabEnabled=False

Access

	Read	Write
Design	+	+
Execution	+	+

Data Type

Logical

Values

TRUE	The tab key inserts a tab character into the document.
FALSE	The tab key serves as a change of focus mechanism.

Default

TRUE

If Out of Range

Value is set to True

Remarks

If TabEnabled is set to FALSE ALLText will lose focus upon entry of the tab key. In this case the developer may wish to trap the <control> <tab> sequence in the KeyDown or KeyUp events and use this to insert a tab character.

TabLocations Property (HT/Pro only)

Description

The TabLocations property is an array of long integer values used to set or return the absolute tab locations for current or selected paragraphs. TabLocations(0) holds the tab location for the leftmost tab setting. TabLocations(1) holds the value for the next location.

Usage

li& = alltext.TabLocations(i%)

alltext.TabLocations(i%) = li&

Access

	R	W
Design	-	-
Runtime	+	+

Data Type

long

Values

0 <= li& <= MAXLONG

0 <= i% <= alltext.TabCount-1

If Out of Range

A trappable error is generated if there is no tab corresponding to the array index specified.

A trapable error is also generated if a negative index is specified.

Remarks

TabLocations(i) returns a value of ATX_TAB_UNDEFINED if more than one paragraph is selected, and the value for TabLocation(i) is different for paragraphs in the set

Tab locations can not be within 16 pixels of each other. If the distance between a tablocation specified by setting this property and the nearest existing tab is less than 16 pixels a trappable Error will be generated, and the tab location will NOT be changed.

TabStep Property

Description

Sets or returns Tabulation step size in twips. Note that ALLText rounds this value to the nearest 16 pixels - Read back the TabStep property after setting to verify the actual tabsize.

Usage

AllText.TabStep= n%
n% =AllText.TabStep

Access

	Read	Write
Design time	-	-
Run time	+	+

Data Type

Integer (Twips)

Values

XPixelsToTwips(ATX_MINTABSET) ≤ n% ≤ ATX_MAXTABSET

Default

XPixelsToTwips(ATX_MINTABSET)

Remarks

The ASCII code for a character is 9. ALLText recognizes chr$(9) as a tab when found in the text string. ALLText also interprets keyboard input of the <Cntrl> <Tab> combination as a tab.

The first tab location will be measured from the left margin of the ALLText window, not from the left margin of the paragraph - thus if the paragraph margin is set at 1/4 inch and the tab setting is for 1/2 inch, the first Tab location will be 1/4 inch into the paragraph. The next tab will be a half inch beyond that.

Note: If set to an out of range value, ALLText will reset this property is to the nearest margin.

Text Property

Description

Holds the unformatted textual content of the first 32,767 characters of the control.

Usage

n$ = ALLText.Text
ALLText.Text="Please spread the word about ALLText"

Access

	Read	Write
Design	-	-
Execution	+	+

Data Type

String

Remarks

Setting the Text property, has the effect of removing any text beyond the first 32,767 characters from the control content. Setting the Text property also results in Unformatted text. To add Formatted text use SelFText = s$

TextFormatted Property

Description

Text modifications that cause reformatting of the entire text can take a lot of time if you have a large document. For this reason AllText formats only the visible part of text immediately upon loading, while the rest of the text is being formatted as a background task after the first screenful is presented.

After a FileLoad operation, ALLText sets the value of the TextFormatted property to specify the number of paragraphs which have been fully formatted internally. When formatting has been completed, the property is then set by ALLText to TRUE (-1).

You can check the value of TextFormatted at any time. You can also set TextFormatted = TRUE in order to force ALLText to complete interpretation of the formatting before proceeding to the next task. If you set TextFormatted property to some other value, AllText will format text from the beginning of the document to the paragraph number indicated. While text is being formatted no another action can be performed by AllText.

Usage

ATX.TextFormatted = l&

l& = ATX.TextFormatted

Default

-1 - The whole text is formatted

Remarks

While this property can be set, it is best to handle it in a read only mode. This will help you to avoid possible problems .

Setting the ChangeEventMask to the Global constant ATXFORMATED_CHANGED_MASK will cause ALLText to trigger an ATXChanged event in response to changes in the TextFormatted property. In this way you can easily identify progress in the background loading operation.

Properties and functions which rely upon the formatting of the complete document may not be reliable unless the TextFormatted property is True. For instance, the DocHeight property indicates the total height (in twips) of the document. This can not be determined until formatting has been completed.

Example

```
ret% = ALLText.ClearAll
ALLText.FileLoad = ATX_IO_LOWLEVEL
ALLText.TextFormatted = True
Label1.Caption  = Str$( ALLText.DocHeight)
```

TextLength Property

Description

Returns the number of characters currently held in the control (length of the textual content).

Usage

n% = ALLText.TextLenth

Access

	Read	Write
Design	-	-
Execution	+	-

Data Type

Integer

Remarks

This is a read-Only property.

Carriage Return/LineFeed combinations delimiting paragraphs are counted as two characters.

Example

Insert text at the end of the control

ALLText.Select = false turn off selection

ALLText.SelStart = ALLText.TextLength

TopIndent Property

Description

Sets or returns indent distance between the top of the paragraph and the bottom of previous Paragraph. (gets added to BottomIndent of previous paragraph)

Usage

ALLText.TopIndent= n&
n& = ALLText.TopIndent

Access

	Read	Write
Design	-	-
Execution	+	+

Data Type

Long (twips)

Values

0 ≤ YTwipsToPixels(n&) ≤ *MaxTextHeight*

Default

1

Example

```
AllText1.TopIndent = .25 * 1440      'set to 1/4 inch
```

Values

The resolution of this property is dependent on the physical device. ALLText holds such information internally in Pixels such that

```
ATX.TopIndent = Screen.TwipsPerPixelX + 1
```

is equivalent to

```
ATX.TopIndent = Screen.TwipsPerPixelX
```

UnDoAction Property

Description

Controls the Undelete buffer.
Setting UnDoAction either undoes the last action (set to 1) or clears the buffer (set to 0).
Reading the UnDoAction property returns the number of possible Undo Steps stored in the buffer.

Access

	Read	Write
Design	-	-
Execution	+	-

Settings

ATX_UNDO_RESET (0) '- clears undo buffer

ATX_UNDO_STEP (1) '- undo last action

Syntax:

```
n% = atx.UnDoAction      ' check number of available Undo Steps.
atx.UnDoAction = 0       'Empty undo buffer
atx.UnDoAction = 1       'Perform one step undelete operation
```

Note:

In order to perform several undo operations, set ALLText.UnDoAction=1 several times.

WriteProtect Property

Description

Enables or disables end-user editing actions upon the control.
Several levels of write protection are available depending on the value set.

Usage

ALLText.WriteProtect = n%

n% = ALLText.WriteProtect

Access

	Read	Write
Design	+	+
Execution	+	+

Data Type

Integer

Values

0 ATX_PROTECT_NONE	The caret is shown, all editing actions are enabled.
-1 ATX_PROTECT_CHANGES	The caret is shown, all actions are enabled except those changing the text.
1 ATX_PROTECT_KEYBOARD	The caret is hidden, selection is reset, all changes are available through properties only.
2 ATX_PROTECT_SCREEN	Just like ATX_PROTECT_KEYBOARD with the addition feature that changes to the text made through properties are not visible on the screen while in this setting.

Default

False

Remarks

True may be used instead of PROTECT_CHANGES, False may be used instead of PROTECT_NONE

Note: If out of range then this property is set to PROTECT_CHANGES

It is a good idea to set WriteProtect when loading and saving a document, when parsing text, or when performing any lengthy manipulation which the end-user should not be permitted to interupt.

ALLText Functions

The following API functions are supported by ALLText. Functions requiring the HT/Pro edition are noted.

ATX_CurToXY Function - translates from a cursor position to X&, Y& coordinates .

ATX_Print Function - prints specified text, and initiates printer events

Atx_PrintCancel Function - aborts printing operation

Atx_Print_Finish Function - terminates ALLText's connection to the printer.

ATX_Print_Region Function - prints to specified region of a page

Atx_Print_Start Function - initializes ALLText link to printer device

ATX_Print_Title Function - provides control over print dialog text

ATX_ToLower Function - this function converts all Selected text in the textbox to Lower Case.

ATX_ToUpper Function - this function converts all Selected text in the textbox to Upper Case.

ATX_XYToCur Function -translates from X&, Y& coordinates to paragraph& and character% location.

ATX_RegNewExtern Function - **(HT/Pro)**registers external object DLL

Find_Htag Function - **(HT/Pro)**searches for the count%-th occurrence of a particular HotSpot region

Find_Ntag Function- **(HT/Pro)**searches for the count%-th occurrence of a particular BookMark region

Find_Phrase Function - searches for a specified text string within the controls content

Find_PhraseEX Function - searches for a specified text string of specified format

FontTableGet Function - reads an entry from the Font Table

FontTablePut Function - sets an entry in the Font Table

Get_Border Function - **(HT/Pro)**translates a border configuration value into workable component features

Get_LineTopHeight - Gets line formatting details for a specified line

Get_ParDet - Gets all Paragraph formatting details for a given paragraph

Get_Shadow Function - translates a Shadow value into workable component features.

Get_Underline Function - translates an Underline value into workable component features.

Make_Border Function - **(HT/Pro)** creates border value from components.

Make_Shadow Function - creates shadow value from components.

Make_UnderLine Function - creates underline value from components.

Print_AText function- prints specified paragraph range

Print_ATextPages function - prints specified pages

ATX_CurToXY Function

Description

Translates from a cursor position specified by paragraph& and character% parameters into X&, Y& coordinates in twips designating displacement of upper left corner of character specified from upper left corner of text.

Declaration

ATX_CurToXY (ATXName As Any, ByVal paragraph as Long, ByVal character as Int, X as Long, Y as Long) As Integer

Usage

x% = ATX_CurToXY (ATXName, paragraph&, character%, X&, Y&)

Returns

The ASCII code for the character located at the position specified if such a character exists, or -1 otherwise.

Input Parameters

ATXName - AllText control name

paragraph& - paragraph where position is placed

character% - character of paragraph where position is placed

Output Parameters

X&, Y& - coordinates in twips designating displacement of upper left corner of character specified from upper left corner of text.

Remarks

This function may be moved to scroll the window to show a particular character in a given location. For instance to scroll the control so that the first character of paragraph 17 is at the top.

See Also

ATX_XYToCur Function

ATX_Print Function

Description:

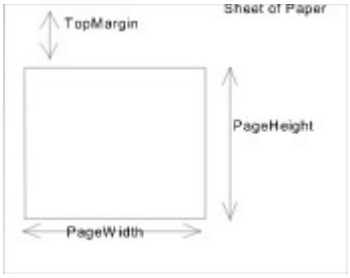
This function initiates printing of the content of an ALLText control from a given paragraph to a destination paragraph.

Syntax

x%= ATX_PRINT (ALLText , StartTop&, TopMargin&, PageHeight&, LeftMargin&, PageWidth&, ParFrom&, ParTo&, Flags%)

Parameters

hctl	- ALLText control containing source document.
StartTop&	- Upon return, ALLText resets this parameter to a value specifying the location (in twips) of the bottom of the last printed line in twips.
TopMargin&	- Top indent
PageHeight&	- Vertical space for the text on the page
LeftMargin&	- Left margin in twips
PageWidth&	- Document width in twips
ParFrom&	- Specifies first paragraph in range to be printed
ParTo&	- Specifies at last paragraph in range to be printed
Flags%	- Not currently used in this function.



Notes:
This function may only be called AFTER the ATX_PRINT_START function and before ATX_Print_Finish.
After calling this function, the following events are triggered sequentially for each page.

PrintThisPage Event
PrintStartPage Event
PrintEndPage Event

To insure that printing is confined to a specific region on a given page, use the ATX_Print_Region function instead of this function.

Atx_Print_Cancel Function

Description:

This function aborts the printing method and should be followed by the Printer.EndDoc method. It may be useful in trapping errors as in the following routine:

Usage

X% = ATX_PRINT_Cancel ()

Parameters

None

Example:

```
Sub PrintSomething(...)
        On Error Goto Printing_Error
        .... ' Print something with ALLTEXT, Printer.Print etc
        Exit Sub
Printing_Error:          ' Error occured in the Printer object
        AtxPrintCancel atxControl
End Sub
```

Atx_Print_Finish Function

Description:

This function terminates ALLText's connection to the printer.

Usage

X% = ATX_PRINT_FINISH ()

Parameters

None

Notes:

This function should generally be followed by the Printer.EndDoc method to inform VB that printing is over and the page should be ejected.

ATX_Print_Region Function

Description:

This function prints text on the current page inside an area defined by parameters. It differes from the ATX_Print Function in that it will not generate any events and will not continue printing on another page if the text to be printed does not fit within the specified area of the printed page.

Usage

x% = ATX_PRINT_REGION (ALLText, StartTop&, PageHeight&, LeftMargin&,
PageWidth&, ParFrom&, ParTo&, Flags%)

Parameters

hctl	- ALLText control containing source document.
StartTop&	- Top indent.
PageHeight&	- Vertical space for the text on the page
LeftMargin&	- Left margin in twips
PageWidth&	- Document width in twips
ParFrom&	- Specifies first paragraph in range to be printed
ParTo&	- Specifies at last paragraph in range to be printed
Flags%	- Not currently used in this function.

Notes

If the Text to be printed does not fit within the specified area, ALLText prints what can fit, and the function returns the error code 109, ATX_PRINT_ERR_REGIONTOOSMALL.

Note that the ATX_Print_Region function is particularly useful when printing to a device with now page feed ability such as a picturebox. This will insure that the text fits within the desired window.

Atx_Print_Start Function

Description:

This function initializes ALLText's link to the proper output device.

Usage

X% = ATX_PRINT_START (FormWnd%, hdc%, log_n%, rel_n%)

Parameters

FormWnd%,	-	Handle to the Form on which ALLText is located
hdc%	-	Handle of print output destination
log_n%	-	Not Currently used
rel_n%	-	Not Currently used, set to 1

Notes:

Parameters log_n% and rel_n% are not used here and should always be set to 1.

The FormWnd% parameter refers to the Forms control handle. This is used to disable the specified form during printing to prevent unexpected user actions, and to set the parent for the print progress dialog box. The form is re-enabled after completing the ATX_Print_Finish function. Setting FormWnd% to 0 prevents disabling of the form and surpresses the PrintProgress Dialog box.

After calling this function with a valid FormWnd, ALLText displays the print dialog box.

ATX_Print_Title Function

Description:

This function provides control within VB to change the text in the dialog box which appears during printing.

Usage

Call ATX_PRINT_TITLE (ByVal TitleString$)

Parameters

TitleString$ - String to be displayed as a title in the PrinterProgress Dialog Box.

Notes:

This function may only be called AFTER the ATX_PRINT_START function and before ATX_Print_Finish.

The printer progress dialog box only appears if the document is to be printed on a physical printer device - not when printing to the screen or a bitmap.

To prevent display of the PrinterDialog box the FormWnd parameter of ATX_Print_Start function should be set to 0.

ATX_ToLower Function

Description

This function converts all Selected text in the textbox to Lower Case.

Syntax

Call ATX_TOLower (CntrolName$)

Input Parameters

Cntrl Name - the name of the ALLText control to be acted upon

Example

```
Sub Cmd_Click()
    Call ATX_TOLower(ALLText)
    End Sub
```

ATX_ToUpper Function

Description

This function converts all Selected text in the textbox to Upper Case.

Syntax

Call ATX_TOUPPER (CntrolName$)

Input Parameters

Cntrl Name - the name of the ALLText control to be acted upon

Example

```
Sub Cmd_Click()
    Call ATX_ToUpper(ALLText)
    End Sub
```

ATX_XYToCur Function

Description

Translates X&, Y& coordinates (in twips) from upper left corner of document into cursor position specified by paragraph& and character% .

Syntax

x% = ATX_XYToCur (ATXName , X&, Y&, paragraph&, character%)

Returns

The ASCII code for the character located at the position specified by paragraph& and character% parameters if such a character exists or -1 otherwise.

Input Parameters

X&, Y& - coordinates in twips designating displacement of upper left corner of character needed from upper left corner of text.

Output Parameters

ATXName - AllText control name

paragraph& - paragraph where position is placed

character% - character of paragraph where position is placed

Remarks

This even may be very useful during MouseMove events. Note however that MouseMove priovides X,Y coordinates from Top Left of control window, while the function expects coordinates with regard to top of text document. Offsets between the two systems are specified by ScrollVert and ScrollHorz properties.

ATX_RegNewExtern Function (requires HT/Pro edition)

Description

Loads specified external Object DLL and registers it with ALLText.

Syntax

x% = ATX_REGNEWEXTERN (noMsg%, lpstrModule$)

Parameters

noMsg% = FALSE - Display message box if the specified DLL is not found in the PATH.

noMsg% = TRUE - Show no error message

lpstrModule$ - Path to the ExtObjDLL for example "ATXPIC.DLL" or "C:\tmp\ATXPIC.DLL".

Example:

```
Sub Form_Load ()
    X = ATX_REGNEWEXTERN(ByVal False, "ATXOLE.DLL")
    X = ATX_REGNEWEXTERN(ByVal False, "ATXOLE.PIC")
     .....
    End Sub
```

Returns

0	if OK
< >0	if an error occurred

Remarks

External object DLLs must be registered before they can be used. They only need to be registered once each time the ALLText control is loaded into memory (not for each instance, or even for each form containing ALLText).

Find_Htag Function (requires HT/Pro edition)

Description

Searches for the count%-th occurrence of a particular tagged region, as specified by the tag value (Htag&). If found, the caret is repositioned and count% is returned. If there is no occurrence of Htag& within the Text - then nothing happens, the function returns 0. If an insufficient count of the sought phrase exists in the document - the caret is positioned on the last occurrence of Htag&, and the actual occurrence number is returned.

Note that the search takes place starting from the current position and looks forward in the text.

Syntax of call

%=Find_Htag(CntrlName, Htag&, count%)

See the file ATX4sAPI.BAS or ATX4hAPI.BAS for the appropriate function declaration

Parameters

CntrlName - the control name. Find_HTag can operate on any ALLText control in the application.

Htag& - Tag # to find. .

count% - counter of tags, designates what occurrence of Htag& is needed. Can be of values {1..32000} or {-1..-32000}. If count% > 0 then searches forward, else backward.

Remarks

Searching for the 0th occurance of the current tag (the value of HTag at the current cursor position) will act to select the complete tagged phrase, ex: *x% = Find_HTag(atx1, Atx1.HTag ,0)* .

Example

Let's say there are 100 sentences marked by Htag=4 and 30 phrases marked by Htag=9 placed arbitrarily within the text. With the caret initially positioned between 50th and 51th phrases of Htag=4 and between 10th and 11th phrases of Htag=9:

Find_HTag(atx1, 4,5) positions caret on 55th phrase of Htag=4, returns 5.

Find_HTag(atx1, 9,11) positions caret on 21th phrase of Htag=9, returns 11.

Find_HTag(atx1,4,200) positions caret on last phrase marked by Htag=4, NOTE: returns 50.

Find_HTag(atx1,9,-100) positions caret on first phrase of Htag=9, NOTE: returns -10

Find_HTag(atx1,9,1) positions caret on 11th phrase of Htag=9, returns 1.

Find_HTag(atx1,9,-1) positions caret on 10 phrase of Htag=9, returns -1.

Find_HTag(atx1,0,1) positions caret forward on first tagged area without regard of tag # (skipHtag forward), returns 1.

Find_HTag(atx1,0,-1) positions caret backward on first tagged area without regard of tag # (skipHtag backward), returns -1

Find_Ntag Function (requires HT/Pro edition)

Description

Searches for the count%-th occurrence of a particular tagged region, as specified by the tag value (Ntag&). If found, the caret is repositioned and count% is returned. If there is no occurrence of NTag& within the Text - then nothing happens, the function returns 0. If an insufficient count of the sought phrase exists in the document - the caret is positioned on the last occurrence of NTag&, and the actual occurrence number is returned.

See Find_HTag for further details

Find_Phrase Function

Description

The FIND_PHRASE function searches for a specified text string within the controls content. The search begins at the current caret position and proceeds forward or backward according to the Direction% parameter.

If found, the function returns a positive long value or 0, designating the character count offset of the search string from the beginning of the AllText content; else returns ATX_UNDEFINED.

Syntax

Pos& = Find_phrase(CntrlName, FindWhat$, Direction%, CaseSensitive%)

Input Parameters

CntrlName	- Name of the ALLText control window being searched
What$	- String containing the phrase to find.
Direction%	- Integer (Boolean) value. If True, then search forward else search backward.
CaseSensitive%	- Integer (Boolean) value. If True, then case sensitive search else case ignored.

Example

```
Sub SearchText (ATX1 As Control, CaretGoToFlag%, PhraseMarkFlag%)
        ss& = ATX1.SelStart
        sl& = ATX1.SelLength
        wp% = ATX1.WriteProtect
        ATX1.WriteProtect = ATX_PROTECT_SCREEN
        posfound& = find_phrase(ATX1, TextToSearch, SearchForward, SearchCaseSen)
        If posfound& < 0 Then
                MsgBox "phrase not found"
          Else
                If NOT CaretGoToFlag% then exit Sub
                ss& = posfound&
                sl& = 0
                If PhraseMarkFlag% then sl& = Len(TextToSearch)
          End If
        ATX1.SelStart = ss&
        ATX1.SelLength = sl&
        ATX1.WriteProtect = wp%
End Sub
```

Find_PhraseEX Function

Description

This is an enhanced search function supporting a search based on the formatting of the text being sought.

Syntax

Res& = FIND_PHRASE(ALLText, FindWhat, SearchDirection, SearchCaseSen, SearchFontName, SearchSize, SearchWidth, SearchBold, SearchItalic, SearchUnderline, SearchStrike, SearchShadow, SearchSubSup, SearchHidden, SearchColor, SearchHTag,StartPos, StopPos)

Return Value

TRUE 1 - if search is successful
UNDEF = - 4096 = &HFFFFF000&) - phrase not found

Input Parameters

alltext - Specifies which ALLText control the search should be performed upon.
FindWhat - Text to be found. If this string is empty (=) Find_PhraseEX will look for any text with the specified formatting attributes.
Direction - Search direction (Forward if 1 or Backward if 0).
CaseSensFlag - Case sensitivity flag (Case sensitive if 1 and Case insensitive if 0).

NOTE: The following input parameters specify font formatting to be used as a filter during the search. The search will ignore a particular formatting parameter if the string value (SearchFontName) is set to an empty string, or the numberic value is set to ATX_UNDEFINED (= -4096).

SearchFontName - Font name of the search string. The font will be ignored during the search if this parameter contains an empty string.
SearchSize% - Font Height
SearchWidth% - Font Width.
SearchBold% - Font bold.
SearchItalic% - Font Italic
SearchFontUl% - Font underline.
SearchStrike% - Strikedtrough font
SearchShadow% - Font shadow
SearchSubSup% - Super or Subscript font
SearchHidden% - This value is not used in current version of AllText.
SearchColor& - Foreground font color.
SearchFontHid% - (HT/Pro edition only, ignored for S)
Font Hidden Property. If ATX_UNDEFINED (= -4096 = &HFFFFF000&)
SearchHTag& - (HT/Pro edition only, ignored for S)
The value of Htag to find

Output Parameters

FoundPos& - Starting location where phrase was found,
FoundLen& - Length of the phrase found

Example

```
Dim ATX As Control
Set ATX = form.ATX
TextToSearch = "String"
SForward = 1  ' Search Forward
SCaseSen = TRUE        ' Case Sensitive search
SFontName = "Times New Roman"            ' Find only Times New Roman text
```

```
SFontHt = 10:     SBold = 1           ' Find only Only 10pt bold text
Ignore all other character related attributes
SWidth = &HF000:          SItalic = &HF000:          SUnderl = &HF000:       SStrike = &HF000:
SShadow = &HF000:      SScript = &HF000:          SHidden = &HF000:         SColor   = &HFFFFF000
SHid = &HFFFFF000&
SHTag     = &HFFFFF000&

res& = find_phrase(ATX, TextToSearch, SForward, SCaseSen, SFontName, SFontHt, SearchFontWd, SBold,
SItalic, SUnderl, SStrike, SShadow, SScript, SHidden, SColor, SHid, SHTag, begPos, finPos)
If res& < 0 Then
      MsgBox "phrase not found"
      End If
```

REMARKS

Setting the Numeric values for formatting parameters to ATX_UNDEFINED will cause ALLText to disregard that formatting when conducting its search.

FontTableGet Function

Description

This function fills in its parameters with corresponding values from the specified ALLText font table entry number and returns TRUE if EntryNumber exists and FALSE otherwise.

Syntax

X= FontTableGet(ControlName$, EntryNum%, Family%, CharSet%, FontName$, FontSize%, FontWidth%, FontBold%, FontItalic%, FontUnder%, FontStrikeOut, FontShadow%, FontSubSup%)

Return Value

True if entry specified exist, False otherwise.

Input Parameter

ATXName	- AllText control name
EntryNum%	- Number of font table entry.

Output Parameters

family%	- Font family
CharSet%	- Font character set
FontName$	- Font name
FontSize%	- Font size
FontBold%	- True if font is bold
FontItalic%	- True if font is Italic
FontUnder%	- Value characterizing font's underlining.
FontStrikeOut%	- True if font is Striked out
FontShadow%	- Value characterizing font's Shadowing.
FontWidth%	- Font width
FontSubSup%	- Value characterizing Super-SubScript.

FontTablePut Function

Description

This function sets the specified ALLText Fonttable Entry based on the values of each parameter and returns the Fonttable entry number that was actually updated.

The set of Font Table Entry numbers must be continuous. Attempting to set an entry number by reference to a value greater than FontTableSize will result in ALLText renumbering that entry and returning the actual entry number. For example if you try to set parameters for the 10th entry in a fonttable that has only five entries, ALLText will update the last fonttable entry in this case and return value will be 5-1=4

Syntax:

x% = FontTablePut (ATXName, EntryNum%, family%, CharSet%, FontName$, FontSize%, FontWidth %, FontBold%, FontItalic%, FontUnderline%, FontStrikeOut%, FontShadow%, FontSubSup%)

Returns

Entry number where it is actually set (see Remark).

Input Parameters

ATXName	- AllText control name
EntryNum%	- Number of font table entry.
family%	- Font family
CharSet%	- Font character set
FontStrikeOut%	- True if font is Striked out
FontName$	- Font name
FontSize%	- Font size
FontWidth%	- Font width
FontBold%	- True if font is Bold
FontItalic%	- True if font is Italic
FontUnder%	- Value characterizing font's underlining.
FontShadow%	- Value characterizing font's Shadowing.
FontSubSup%	- Value characterizing Super-SubScript.

Example

```
'-get details for 3-rd entry
  FontTableGet (ALLText1, 3, family%, CharSet%, Name$, Size%, Width%, Bold%, Italic%, Underline
                                          %,      StrikeOut%, Shadow%, SubSup%)
'-change some parameters
  Bold%=Not Bold%   ' Toggle Bold
  Italic%=Not Italic%' Toggle Italic
'-Add entry to font table
  SetasIndex% = FontTablePut (ALLText1, 32767, family%, CharSet%, Name$, Size%, Width%, Bold%,
                                          Italic%, Underline%, StrikeOut%, Shadow%, SubSup%)
```

Remarks

- If EntryNum% is less than 0 then 0-th entry is assumed.

- If EntryNum% is greater than last existing entry number then the font specified by input parameters is added to the font table. The Function returns its index in font table.

Get_Border Function (HT/Pro Version Only)

Description

The Get_Border function translates a border configuration value as used by the Border Property into individual side%, shape% and shadow% values.

Syntax

Call get_border(border&, side%, shape%, shadow%)

See the file ATX4sAPI.BAS or ATX4hAPI.BAS for the appropriate function declaration.

Input Parameters

border& - paragraph border value (see border property).

Output Parameters

side% - The sides of paragraph rectangle where bordering lines are set:
ATX_BORDER_NONE or ATX_BORDER_BOX
or superposition of constants:
ATX_BORDER_LEFT, ATX_BORDER_RIGHT,
ATX_BORDER_TOP, ATX_BORDER_BOTTOM.

shape% - bordering shape:
ATX_BORDER_SHAPE_STANDARD or
ATX_BORDER_SHAPE_THICK or
ATX_BORDER_SHAPE_DOUBLE.

shadow% - border is shadowed or not:
ATX_BORDER_NOTSHADOWED or
ATX_BORDER_SHADOWED

Returns

none

Example

```
b& = alltext1.border

call get_border( b&, side%, shape%, shadow%)
```

Get_LineTopHeight Function

Description

Gets line formatting details for a specified line within a specified paragraph

Usage

Get_LineTopHeight(alltext, Paragraph&, Line&, top&, height&)

Input Parameters

alltext	- ALLText control
Paragraph&	- paragraph number within ALLText
Line&	- line number within Paragraph&

Output Parameters

top&	- Line top (twips)

height& - Line height (twips)

Get_ParDet Function

Description

Gets paragraph formatting details for a given paragraph

Usage

Get_Pardet(alltext, Paragraph&, top&, height&, FirstLineIndent&, LineSpace&, SpaceAfter&, SpaceBefore&)

Input Parameters

alltext - ALLText control
Paragraph& - paragraph number within ALLText

Output Parameters

top& - Paragraph top (twips)
height& - Paragraph height (twips)
FirstLineIndent& - Paragraph FirstLineIndent (twips)
LineSpace& - Paragraph LineSpace (twips)
SpaceAfter& - Paragraph SpaceAfter (twips)
SpaceBefore& - Paragraph SpaceBefore (twips)

Example

```
Call GET_PARDET(AllText1, (AllText1.CurPar), part, parh, a1, a2, a3, a4)

parb = part + parh    ' distance from top of document to bottom of paragraph
```

Get_Shadow Function

Description

This function is intended to facilitate FontShadow property manipulations, translating a ShadowValue as read via the FontShadow Property, into it's more workable component features.

Syntax

x%= Get_Shadow(ALLText1.FontShadow, delta1%, delta2%, color1%, color2%)

See the file ATX4sAPI.BAS or ATX4hAPI.BAS for the appropriate function declaration

Input Parameters

shadow% - Integer value received by readyng FontShadow property.

Output Parameters

delta1% - Top shadow displacement, to upper left from base
delta2% - Down shadow displacement, to lower right from base.
color1% - Top shadow color.
color2% - Top shadow color.

Return

None

Example

```
' Set shadowing of second Paragraph equal to first.
AllText1.Cur Par = 0
get_shadow(AllText1.FontShadow, d1%,d2%,c1%,c2%)
AllText1.Cur Par = 1
AllText1.Sel Par = 1
AllText1.Sel Char = 32000
AllText1.FontShadow = make_shadow(d1%,d2%,c1%,c2%)
```

Equivalent Visual Basic code

```
Sub get_shadow (shadow%, delta1%, delta2%, color1%, color2%)
    delta1% = (shadow% And (7 * 256 * 16)) / (256 * 16)
    delta2% = (shadow% And (7 * 256)) / (256)
    color1% = (shadow% And (15 * 16)) / (16)
    color2% = (shadow% And (15))
End Sub
```

Get_Underline Function

Syntax

x% = Get_underline (ALLText.FontUnder, shape%, dblflag%, ncolor%)

See the file ATX4sAPI.BAS or ATX4hAPI.BAS for the appropriate function declaration

Description

Sets output parameters to underlining details defined by UnderValue%

Parameter values may be taken from ATX4sAPI.BAS or ATX4hAPI.BAS for HT/Pro version

Input Parameters

UnderValue - underline value

Output Parameters

shape% - ATX_NONE | ATX_SOLID | ATX_DASH | ATX_DOT | ATX_DASHDOT | ATX_DASHDOTDOT.
dblflag% - ATX_SINGLE | ATX_DOUBLE.
ncolor% - ATX_Black | ATX_Blue | etc.

Example

```
' toggle single/double underlining
    GetUnderline(AllText1.FontUnder, Shp%, Dbf%, Clr%)
    If Dbf%=ATX_SINGLE then Dbf%=ATX_DOUBLE else Dbf%=ATX_SINGLE
    AllText1.FontUnder = make_underline (Shp%, Dbf%, Clr%).
```

Make_Border Function (HT/Pro Version Only)

Description

The Make_Border function creates a value acceptible to the Border Property from the individual border components: side, shape and shadow.

Syntax

b& = make_border(side%, shape%, shadow%)

See the file ATX4sAPI.BAS or ATX4hAPI.BAS for the appropriate function declaration.

Input Parameters

side% - The sides of paragraph rectangle where bordering lines are set: ATX_BORDER_NONE or ATX_BORDER_BOX or superposition of constants: ATX_BORDER_LEFT, ATX_BORDER_RIGHT, ATX_BORDER_TOP, ATX_BORDER_BOTTOM.

shape% - bordering shape: ATX_BORDER_SHAPE_STANDARD or ATX_BORDER_SHAPE_THICK or ATX_BORDER_SHAPE_DOUBLE.

shadow% - border is shadowed or not: ATX_BORDER_NOTSHADOWED or ATX_BORDER_SHADOWED

Output Parameters

returns long value specifying resulting border

Example

```
b& = alltext1.border
call get_border( border&, side%, shape%, shadow%)
alltext1.border = make_border( side%, shape%, 1)
```

Make_Shadow Function

Description

This function is intended to facilitate FontShadow property manipulations, translating desired component features of a shadow font into a value acceptable to the FontShadow property.

Syntax

ALLText1.FontShadow=Make_Shadow% (delta1%, delta2%, color1%, color2%)

See the file ATX4sAPI.BAS or ATX4hAPI.BAS for the appropriate function declaration

Input Parameters

delta1%	Top shadow displacement - to upper left from base.
delta2%	Down shadow displacement - to lower right from base.
color1%	Top shadow color.
color2%	Top shadow color.

Returns

An integer value used to set FontShadow property (see FontShadow property description).

Example

AllText1.FontShadow = make_shadow(1,1,0,15)

Equivalent Visual Basic Code:

```
Function make_shadow% (delta1%, delta2%, color1%, color2%)
    If delta1% < 0 Or delta1% > 15 Then Exit Function
    If delta2% < 0 Or delta2% > 15 Then Exit Function
    If color1% < 0 Or color1% > 15 Then Exit Function
    If color2% < 0 Or color2% > 15 Then Exit Function
    make_shadow% =256*16*delta1% +256*delta2% +16*Color1% +color2%
End Function
```

Make_UnderLine Function

Description

Returns integer, defining underline mode, provided by parameters.

Syntax

AllText1.FontUnder = make_underline (shape%, dblflag%, ncolor%)

Input Parameters

shape%	-	ATX_NONE | ATX_SOLID | ATX_DASH | ATX_DOT | ATX_DASHDOT | ATX_DASHDOTDOT.
dblflag%	-	ATX_SINGLE | ATX_DOUBLE.
ncolor%	-	ATX_Black | ATX_Blue | etc.

See the file ATX4SAPI.BAS or ATX4HAPI.BAS for the appropriate function declaration

Example

AllText1.FontUnder = make_underline (ATX_SOLID, ATX_SINGLE, color_index(alltext1.fontcolor).

Print_AText function

Description

The Print_AText function provides a simple method of printing a range of paragraph with a single function call.

Syntax

i% = Print_AText(CntrlName, pTop&, pBottom&, pLeft&, pWidth&, PrintDoc_from_par%, PrintDoc_to_par%, PrintDoc_Flag%)

Input Parameters

CntrlName	- the Name of the source control.
pTop&, pBottom&	- Top and bottom margins in Twips.
pLeft&, pWidth&	- Left Margin and Line Length in Twips.
PrintDoc_from_par%	- Specified start of paragraph range.
PrintDoc_to_par%	- Specifies end of paragraph range.
pFlag%	- set to 1 to include the background bitmap, 0 otherwise.

Remarks

While easy to use, this function does not provide any control over the printer dialog box, and always forces a page eject after printing each page.

Note that the DocWidth property setting does not affect the text wrapping width when printing using the built in printer functions. This is independently set by the parameters of the print functions.

Also note that unless the PrinterDC property is TRUE, even with the width settings exactly the same, due to differences in printer resolution the output may line break a word prior to, or after the break shown in the control window.

Print_ATextPages function

Description

The Print_ATextPages function provides a simple method of printing specified pages with a single function call.

Syntax

i% = Print_ATextPages(CntrlName, pTop&, pBottom&, pLeft&, pWidth&, pPages$, pFlag%)

Input Parameters

CntrlName	- the Name of the source control.
pTop&, pBottom&	- Top and bottom margins in Twips.
pLeft&, pWidth&	- Left Margin and Line Length in Twips.
pPages$	- A string indicating which pages are to be printed. ALLText scans this string and prints pages corresponding to non-blank elements in the string. ie: if the string consists of two blanks followed by a character, another blank and two characters, ALLText will print pages 3,5 and 6.
pFlag%	- set to 1 to include the background bitmap, 0 otherwise.

Remarks

While easy to use, this function does not provide any control over the printer dialog box, and always forces a page eject after printing each page.

Note that the DocWidth property setting does not affect the text wrapping width when printing using the built in printer functions. This is independently set by the parameters of the print functions.

Also note that unless the PrinterDC property is TRUE, even with the width settings exactly the same, due to differences in printer resolution the output may line break a word prior to, or after the break shown in the control window.

ALLText Events

The following Events are supported. Events requiring the HT/Pro or forms edition are so noted:

ATXChange	- triggered by a change of state
ATXGet	- triggered by the setting of the FileLoad property. Supports Low level I/O
ATXPut	- triggered by the setting of the FileSave property. Supports Low level I/O
ATXHScrollClick	- triggered by click on Horizontal Scroll bar
ATXVScrollClick	- triggered by click on Vertical Scroll bar
DragDrop	- standard Visual Basic Event
DragOver	- standard Visual Basic Event
DropFileStart	- **(HT/Pro)** triggered by the dropping of a file on the control.
DropFile	- **(HT/Pro)** triggered by the dropping of a file on the control.
ExternOLEAction	- **(Forms)** provides data exchange with embedded external objects.
PrintThisPage	- triggered prior to printing each page.
PrintStartPage	- triggered prior to the start of each each printed page .
PrintEndPage	- triggered upon completion of the printing of each page of text

ATXCHANGE Event

Description

This event is triggered by changes in the state of the ALLText control. The ChangeEventMask property is used to control which state changes trigger the event.

Uses - This event can be used to update a ruler, status bar, or button bar. It may also be used to pop up a window or a help Bubble when moving the mouse over a key phrase. It is intended to support the display of current parameters pertaining to caret position or selected text region (eg: cursor location or current font name).

Syntax

ATXChange(ChangedMask As Long, Param1 As Long, Param2 As Long)

Parameters

ChangedMask As Long - Presents reason for the event to occur.

Param 1 as Long - State of ALLText **before** change

Param2 As Long - State of ALLText **before** change

See the following table for specific details.

Global Constant	**Event Triggered in response to Change in:**	**Param 1 Contains**	**Param2 Contains**
ATXTAG_CHANGED = 1	Htag Value	New HTag	Old HTag
ATXMOUSE_CHANGED = 2	Mouse cursor	New Mouse Cursor	Old Mouse Cursor
ATXFONT_CHANGED = 3	Font, including font characteristics such as underline	New FontIndex	Old FontIndex
ATXCOLOR_CHANGED = 4	Font color	New Color	Old Color
ATXCURSORLOCATION_CHANGED = 5	Caret location	0	0
ATXSELECTION_CHANGED = 6	Selected area	1 if selected, 0 if not	0
ATXLOCATIONX_CHANGED = 7	X location of the AllText window	New X	Old X
ATXLOCATIONY_CHANGED = 8	Y location of the AllText window	New Y	Old Y
ATXSELPARAGS_CHANGED = 9	Selected paragraph area	0	0
ATXPARALIGNMENT_CHANGED = 10	Paragraph Alignment	New Align	Old Align
ATXMARGIONX_CHANGED = 11	Paragraph Margins: LeftMarg, RightMarg or FirstLineIndent	0	0
ATXSCROLLH_CHANGED = 12	Horizontal scroll	0	0
ATXSCROLLV_CHANGED = 13	Vertical scroll	0	0
ATXPAGE_CHANGED = 14	Current page number	New Page #	Old Page #
ATXPAGELINE_CHANGED = 15	Current Page Line Number	New line	Old Line
ATXLINE_CHANGED = 16	Line number.	New line	Old Line
ATXFORMATED_CHANGED = 17	The value of the Textformatted property has changed - this may indicate progress in the background formatting process.	Par #	Par #
ATXCHANGED_CHANGED = 18	DataChanged property.	New	Old

Remarks

It is possible to prevent generation of ATXChange event by setting corresponded bits of the ChangeEventMask to zero.

All events allowed by ChangeEventMask will be triggered in following cases:

New Window was created.

New data is loaded from a bound database.

ChangeEventMask property was changed.

A File is loaded with FileLoad property.

NewDoc property was changed.

This event occurs after completing some changes in the ALLText **while it is idle** that is not busy of some work like processing fast keyboard input.

See Also

ChangeEventMask Property

ATXGet Event

Description

The ATXGet event presents the developer an opportunity to set or modify the input stream being fed to ALLText. It is triggered during I/O when the ***FileLoad*** property is set into ATX_IO_STANDARD or ATX_IO_LOWLEVEL.

In Standard Mode, the user will first set a filename and open the file. ALLText will read in some portion of the file (currently 2000 characters, but this is subject to change), and then trigger the event. The developer may then modify the input string UserSTR which is added to the ALLText document content upon return from the event subroutine.

In LowLevel mode ALLText does not read from a file at all, but triggers the ATXGet event to allow the user to set the UserSTR parameter (possibly from a serial I/O stream).

The ATXGet event subroutine uses a flag parameter (FLAG) to provide the programmer with the possibility to interrupt loading process.

In standard I/O mode the FLAG is automatically initialized to 1 upon entry to the subroutine each time there is data read from the file. When the end of the file is reached, the event is triggered once more with UserStr parameter set to 0 indicating an EOF condition. Manually set the flag to 0 in order to interrupt the loading process.

In low level mode, the FLAG is automatically initialized to 0 (stop) upon each entry to the event procedure and must be programmatically set to 1 before exiting the procedure in order to process the input text and continue triggering additional ATXGet events. No text will be processed during an ATXGet event where the FLAG is 0 upon exit from the routine.

Syntax

Sub Alltext_ATXGet(flag As Integer, UserSTR As String)

Input Parameters

flag As Integer - 1 - file contains data, 0 - file is empty

Output Parameters

flag As Integer - 1 - continue file loading, 0 - break file loading

UserSTR As String - the string read from file (and modified if necessary) that will be taken by AllText.

Remarks

The *FileSave* and *FileLoad* properties cannot be called within ATXGet event subroutine.

Example 1: STANDARD LOADING

```
...
alltext1.DataType = file_format%   ' set the file type
alltext1.FileName = file_name$     ' set the file name
alltext1.FileLoad = ATX_IO_STANDARD
....

Sub AllText1_ATXGet (flag As Integer, UserSTR As String)
    If Len(UserStr)>ATX_MAXCHARS Then ' interruption of file loading
        flag = 0
        GoTo ATX_GET_err
        EndIf
    UserSTR = UserSTR + Chr$(13) + Chr$(10)
Exit Sub
```

```
ATX_GET_err:
    Exit Sub
End Sub
```

Example 2: Low Level LOADING

```
Sub load_from_file (file_name$, file_format%)
    wp% = alltext1.WriteProtect:    alltext1.WriteProtect = 2
    i% = alltext1.ClearAll          ' Clear the control content
    alltext1.FontTableSize = 0   ' Remove unused fonts from font table
     ....   take actions to initialize modem, sending appropriate ESC sequences ....
    alltext1.DataType = file_format%
    alltext1.FileLoad = ATX_IO_LOWLEVEL    '   generates an ATXGet event for each chunk loaded
     ....   take actions to reset the modem, sending appropriate ESC sequences. ...
    alltext1.WriteProtect = wp%
End Sub

Sub AllText1_ATXGet (flag As Integer, UserSTR As String)
    UserStr = AModemReadingFunction()
    Select Case Len(UserStr)
        Case is > ATX_MAXCHARS      'ALLText paragraph size limit reached
            flag = 0      ignore any UserStr data and terminate I/O
            ' handle condition as desired.
        Case =0          ' No input from modem
            flag = 0      ignore any UserStr data and terminate I/O
        Case Other
            UserSTR = UserSTR + Chr$(13) + Chr$(10)
            flag =1 ' Add UserStr content to ALLText and continue receiving input
        End Select
End Sub
```

PrintEndPage Event

Description

This event is triggered after completing the printing of each page of text.

Within this event the programmer can print additional information on the page before it is ejected. This can be done by means of Visual Basic functions, WINAPI functions or ATX_PRINT_REGION function. In particular, the ATX_Print_Region function can be used here to print Footers.

Syntax

ALLText_PrintEndPage(PageNumber As Integer, SkipEndPageFlag As Integer)

Input Parameters

PageNumber - the current page number
SkipEndPageFlag - On entry into this event, the flag is set to 1 for the last page, or 0 otherwise.

OutPut Parameters

SkipEndPageFlag - determines whether to surpress the end-of page pagebreak. If you set this flag to TRUE printing will continue on the same sheet otherwise a page break will be inserted and printing will continue on the next page.

Remarks

Parameter SkipEndPageFlag is used by AllText to determine if it has to print the next page on the same sheet of paper as it just completed printing. If you want to print several small documents on the same page you should set SkipEndPage flag to TRUE for the final page of each document.

It is possible to identify the last page of a document by checking the SkipEndPage flag. This will be initialized as 1 for the last page.

PrintStartPage Event

Description:

This event is triggered for each page. It is triggered after the PrintThisPage event, but just before printing each page. It can be used to change location of the printed page on printing media.

Synatx

ALLText_PrintStartPage(PageNumber As Integer, TopIndent As Long, LeftIndent As Long)

Parameters

PageNumber - the current page number

TopIndent - The top margin for the current page (this can be changed within the event)

LeftIndent - The left margin for the current page (this can be changed within the event)

Remarks

This event can be used to set left and top indents for the current page. For example you can print your document as multiple columns on the page or you can just print Header of the page while processing this event.

This event is NOT triggered if the SkipPageFlag was set True in the PrintThisPage event.

PrintThisPage Event

Description:

This event is triggered for each page after calling the ATX_Print Function and before the ATX_PrintStartPage event. It may be used to skip the printing of a particular page.

Syntax

ALLText_PrintThisPage(PageNumber As Integer, DialogString As String, SkipPageFlag As Integer)

Parameters

PageNumber - the current page number to be printed

DialogString - contains the string to be displayed in the dialog box when printing.

SkipPageFlag - if True, indicates that the current page should be skipped (not printed).

Remarks

This event can be used in order to skip printing of the current page. For example you can use this event if you want to print only odd or only even pages. This can be done by setting **SkipPageFlag** to **TRUE.**

On input Dialog String contains the value set by the ATX_Print_Title function.

Setting DialogString to an empty string will prevent display of the printer dialog box

Setting SkipPageFlag to True prevents triggering of the ATX_PrintStartPage and ATX_PrintEndPage events

ATXPut Event

Syntax

Sub ALLText1_ATXPut (Flag As Integer, UserSTR As String)

Description

In a manor analogous to the ATXGet event, ATXPut presents the developer an opportunity to set or modify the output stream being read from ALLText. It is triggered during I/O when the FileSave property is set into ATX_IO_STANDARD or ATX_IO_LOWLEVEL.

In Standard Mode, the user will first set a filename and open the file. ALLText will trigger the event after preparing to write some chunk of content (currently up to 2000 characters, but this is subject to change), passing that content to the event procedure in the parameter UserStr. The developer may then modify the string. ALLText then write out to the file upon return from the event. This continues until the document has been fully processed or until the control flag is set to 1 according to the VB code.

In LowLevel mode ALLText does not write to a file at all, but triggers the ATXPut event to allow the user to handle the output UserSTR parameter as desired (possibly directing it to a serial I/O stream).

The ATXGet event subroutine uses a flag parameter to provide the programmer with the possibility to interrupt loading process. Each time during loading process flag=1. When the end of file is achieved the ATXGet event will be triggered with flag=0. If the programmer wishes to interrupt a loading process he should set flag=0 manually.

Input Parameters

Flag As Integer

1 - if Userstr is not empty, 0 - Userstr=""

UserSTR As String - String containing part of AllText content in format depending on DataType.

Output Parameters

Flag As Integer

1 - continue file saving, 0 - break saving process.

Example: LOW LEVEL SAVING

```
Sub save_to_file (file_name$, file_format%)
    wp% = alltext1.WriteProtect:      alltext1.WriteProtect = 2
  ' Select text to save
    alltext1.SelStart = 0 :    alltext1.SelLength = 2000000

    RTFfn% = FreeFile
    Open file_name$ For Output As RTFfn%
    On Error GoTo 0

    alltext1.DataType = file_format%
    alltext1.FileSave =ATX_IO_LowLevel
    Close RTFfn%

    alltext1.WriteProtect = w%
End Sub

Sub AllText1_ATXPut (flag As Integer, UserSTR As String)
    crlf$ = Chr$(13) + Chr$(10)
```

```
    ulen% = Len(UserSTR)

    Print #RTFfn%, UserSTR;
    On Error GoTo li_err
    If Right$(UserSTR, 2) = crlf$ Then
      blklen% = 0
    Else
      blklen% = ulen% + blklen%
      If blklen% > 200 Then
            Print #RTFfn%, crlf$;
            On Error GoTo li_err
            blklen% = 0
      End If
    End If
    goto li_ok
li_err:
    flag = 0
    Resume li_ok
li_ok:
End Sub
```

ATXVScrollClick & ATXHScrollClick Events

Description

These events are triggered when the end-user clicks on the Vertical or Horizontal Scrollbar.

Syntax

ALLText_ATXVscrollClick (ScrollValue As Long)

ALLText_ATXHscrollClick (ScrollValue As Long)

Parameters

ScrollValue - value characterizing new position of the scrollbar

(the same as can be got reading ScrollHorz or ScrollVert property).

DragDrop Event

Description

Standard Visual Basic Event

Syntax

Sub ATX_DragDrop (Source As Control, X As Single, Y As Single)

References

DragOver Event

Description

Standard Visual Basic Event

Visual Basic

ATX_DragOver (Source As Control, X As Single, Y As Single, State As Integer)

DropFileStart Event

Description

This event occurs every time you drop several files from File Manager before you get any DROPFILE event. It is triggered only if allowed by the DropFileMode property.

Visual Basic

Sub ALLText_DropFileStart(X As Long, Y As Long, FNumber As Integer)

Remarks

This event gives to user X and Y coordinates of AllText window point where Left mouse button was released while performing Drag-n-Drop operation. The value of parameter FNumber is equal the number of files taken from File Manager. You can change FNumber if your application does not support multiple file Drag-n-Drop operation. (For example you can set FNumber to 0)

See Also

DropFile Event

DropFileMode -- property

DropFile Event

Description

This event occurs each time AllText window receives DROPFILE event. It is triggered only if allowed by the DropFileMode property.

Visual Basic

Sub ALLText_DropFile(X As Long, Y As Long, FName As String, FIndex As Integer)

Remarks

This event gives to user X and Y coordinates of AllText window point where Left mouse button was released while performing Drag-n-Drop operation. Parameter FName contains file name being dropped to the AllText window. Parameter FIndex can be used to break Drag and Drop processing if an error occurred. This can be made by setting Findex flag to (1).

See Also

DropFileStart -- event

DropFileMode -- property

ExternOleAction Event (for ALLText/Forms edition)

Description

This is event provides data exchange between an AllText ExtObject and the application. The use of this event depends on details of the particular ExtObject DLL. It is applicable only to the ALLText/Forms edition.

TroubleShooting

I Lost the Pictures in my document when I transferred using the clipboard.

This is most probably due to improperly registering or not registering the ATXPIC.DLL. This DLL provides Picture handling support to ALLText.

Make sure you have registered the DLL using the ATX_RegNewExtern function in your VB Project:

```
Declare Function ATX_REGNEWEXTERN Lib "atx40h" (ByVal noMsg%, ByVal lpstrModule$)
As Integer

X% = ATX_REGNEWEXTERN  (0, ATXPic.DLL)
```

I Lost the formatting during cut & paste.

This may be the result of copying from an application which did not support RTF to the clipboard. Applications such as MS Write can not exchange formatted data with ALLText, but will only copy the raw text.

Another possible source of the problem is the improper use of SelFText to move strings between ALLText controls. The formatting codes included in SelFText strings rely on the FontTable definitions associated with a particular instance of the control. Copying a string from one ALLText control to another by using SelFText may lose or change the formatting due to differences in the Font Table definitions. Either use the clipboard (ClipboardAction Property) or use FontTableGet and FontTablePut functions to synchronize the two font tables.

Im losing Carriage Returns.

ALLText is an RTF oriented control. When setting SelFText or FText, or when reading in a file with the datatype set to anything other than 0, all carriage return/line feed combinations (CHR$(13) + CHR$(10)) will be ignored. The correct RTF or ATX format code for a paragraph break (new line) is \par .

I am losing my fonts when working with a database.

This is usually due to having an incorrect DataType setting when using ALLText as a bound control. A DataType of 2 is usually most appropriate. If you know in advance all the fonts and font attributes your application may need, you may wish to use the FontTableGet and FontTablePut functions to hard code the font table. Then you can use DataType of 1 and save some space in the database.

I dont want to see ALLTexts Print Dialog Box. What can I do?

Set the first parameter of the *ATX_Print_Start* Function to 0.

My Doc Width is changing when I change Database records or read in a new file.

Depending on how youve saved your data, it is likely that the setting of the DocWidth property is stored with the data. When you change to a new record, the DocWidth is therefore updated by the new data. If you want to insure a specific Word Wrap width, reset the DocWidth after reading the file or in the Data.Reposition event

Various properties such as DocHeight seem unreliable after I load a large file or paste a large section from the clipboard.

This is probably due to ALLTexts dynamic formatting. In order to speed the initial display of large documents, ALLText initially formats only enough of the document to allow proper viewing within the control window. Additional formatting is carried on as a background task.

ALLTexts dynamic text formatting leads us to the result some ALLText properties may have incorrect values for a short period. This makes possible occurrence of the ATXChange events that are not concerned with real user actions. For example on loading huge text into ALLText window and pressing CTRL+END we will get several ATXChange events with ATXSCROLLV_CHANGED flag set.

To prevent this situation set the TextFormatted property to -1 (forcing ALLText to complete the formatting operation) or wait for the property to reach this value (If TextFormatted >0 it reflects the number of formatted paragraphs).

I am having trouble controlling scrolling

Getting Technical Support

Send us a Sample Project illustrating the problem. We cant help unless we can replicate.

Free Technical Support is Limited to 30 minutes per call, with a Total of 4 hrs during first 30 days.

Embedded Format Codes

The codes described below are used by ALLText in reading and writing ATX formatted files, and in support of direct formatted content manipulation through the SelFText property. Note that numeric codes are terminated by a space. Note too, that where a single backslash is needed in the text, it is read and written by ALLText as a double backslash sequence.

\ATXsh<N> - Switch shadowing on (N<>0) or off (N=0)

\ATXts<N> - Set tabulation step for paragraph

\ATXul<N> - Switch underlined on (N<>0) or off (N=0) - See the MakeUnderline function for more details.

\ATXwd<N> - Set current font width.

\b<N> - Switch bold on (N<>0) or off (N=0)

\i<N> - Switch italic on (N<>0) or off (N=0)

\strike<N> - Switch strikethrough on (N<>0) or off (N=0)

\up<N>,\dn<N> - Switch superscript, subscript on (N<>0) or off (N=0).

\f<N> - Set current font to N-th of ATX font table (Not RTF!)

\cf<N> - Set current color to N-th of ATX color table (Not RTF!)

\ql,\qr,\qc,\qj - paragraph alignment settings

\li,\ri,\fi - indents: left, right, firstline

\sl,\sb,\sa -vertical spacings: Line spacing, Space before, Space after

\tx<N>,\tqr,\tqc - positioned tabulation settings for paragraph. (HT/PRO edition Only)

\paperw<N> - Set document width to N twips

\pard - reset paragraph settings to default.

\plain - reset current character settings to default.

\par - paragraph break

\line - soft line break

\page - page break

\~ - nonbreaking space

\ATXbrdr<N> - Switch border on (N<>0) or off (N=0) (HT/PRO edition only)

\ATXht<N> - Switch Htag on (N<>0) or off (N=0) (HT/PRO edition only)

\object, \objemb, - support of embedded ole objects. (HT/PRO edition only)

\objname, \objdata - support of embedded ole objects. (HT/PRO edition only)

BorderStyle Property

Description

Determines the border style for an object. For forms and text boxes, read-only at run time.

Usage

atx.BackBorderStyle = x%

Setting

The BorderStyle property settings for a form are:

Setting	Description
0	None
1	Fixed Single.(Default)

Data Type

Integer (Enumerated)

Default

1

DataField Property (HT/Pro)

Description

Binds the ALLText control to a Memo or Binary Object Field.

Read/write at both design time and run time.

Usage

ALLText.DataField [= fieldname]

Remarks

ALLText can not be bound to a Text field, only to Memo or Binary Object (OLE) fields. Use MEMO fields for under 64 K, or OLE fields for larger amounts of data.

Data Type

String

DataSource Property (HT/Pro)

Description

Determines the data control through which the current control is bound to a database. Read/write at design time; not available at run time.

Usage

Available at Design Time only.

Enabled Property

Description

Determines whether ALLText Control can respond to user-generated events.

Usage

ALLText.ENABLED = <true/False>

Setting

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

BOOL

Default

TRUE

Hwnd Property

Description

Standard Property - Returns a handle to the ALLText control.

Visible Property

Description

Determines whether ALLText is displayed on the form.

Usage

ALLText.ENABLED = <true/False>

Data Type

BOOL

Default

TRUE

Remarks

It is not possible to print from an invisible ALLText control. Instead set the controls height and width to small values and hide the control behind some other control.

TabStop Property

Description

Standard Property - Determines whether a user can use the Tab key to set the focus to a control.

Usage

ALLText.TabStop = <true/False>

Data Type

BOOL

Remarks

Note that once ALLText has the focus, the TabEnabled property determines how the Tab key works. Unless TabEnabled is FALSE, hitting the Tab key will not move the focus out of the ALLText control.

TabIndex Property

Description

Standard Property - Determines the tab order of ALLText within its parent form.

Usage

ALLText.TabIndex[= index]

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.

Data Type

Integer

Remarks

Note that once ALLText has the focus, the TabEnabled property determines how the Tab key works. Unless TabEnabled is FALSE, hitting the Tab key will not move the focus out of the ALLText control.

Index Property

Description

Standard Property - A number identifying the control within a control array. Available only if the control is part of a control array; read-only

Usage

I% = ALLText[(integer)].Index

Data Type

Integer

Remarks

ALLText control.

Tag Property

Description

Standard Property - Hidden data not shown on the control .

Usage

ALLText.Tag = Tag$

Data Type

String

MousePointer Property

Description

Standard Property - Determines the MousePointer in use over the ALLText control.

Usage

ALLText.MousePointer = iMouseP%

Data Type

Integer <Enumerated>

Remarks

Note that users of the HT/Pro edition may specify a distinct mouse pointer for use over hotspots by setting the MouseHPointer property.

Top and Left Properties

Description

Standard Properties - determine the distance between ALLText and the Top Left corner of its container.

Usage

ALLText.Top = Y

ALLText.Height = H

Data Type

Single

Width and Height Properties

Description

Standard Properties - determines the height and width of the ALLText control.

Usage

ALLText.Width = W

ALLText.Height = H

Data Type

Single

Remarks

The word wrapping width within the ALLText control is separately controlled by the DocWidth property. The height of the document within the ALLText control may be read by the DocHeight property.

HelpContextID Property

Description

Specifies the Help context number for the ALLText control.

Usage

ALLText.HelpContextID = b%

Data Type

Integer

MouseMove Event

Description

Triggered as the user moves the mouse over the control, unless another object has captured the mouse.

Syntax

Sub ATX_MouseMove ([Index% As Integer,] Button As Integer, Shift As Integer, X As Single, Y As Single)

Parameters

The MouseMove event uses these arguments:

Argument	Description
Index	Uniquely identifies a control if it is in a control array.
Button	The state of the mouse buttons. This is a bit flag value. If a bit is set it indicates that the button was down. LEFT_BUTTON 1 RIGHT_BUTTON 2 MIDDLE_BUTTON 4
Shift	This is a bit flag indicating the state of the Shift, Ctrl and Alt keys. The bit is set if the key is down. The following BitMasks may be used SHIFT_MASK 1 CTRL_MASK 2 ALT_MASK 4
X ,Y	The current location of the mouse pointer. X and Y are always expressed in terms of the coordinate system set by the ScaleHeight, Scale Width, ScaleLeft, and ScaleTop properties of the object.

MouseDown and MouseUp Events

Description

Occur when the user presses (MouseDown) or releases (MouseUp) a mouse button while over the ALLText control

Syntax

Sub ATX_MouseDown ([Index%,]Button As Integer, Shift As Integer, X As Single, Y As Single)

Sub ATX_MouseUp ([Index %,]Button As Integer, Shift As Integer, X As Single, Y As Single)

Parameters

The MouseDown event uses these arguments:

Argument	Description
Index	Uniquely identifies a control if it is in a control array.
Button	The button which triggered the event. This is a bit flag value. If a bit is set it indicates that the button was down. LEFT_BUTTON 1 RIGHT_BUTTON 2 MIDDLE_BUTTON 4
Shift	This is a bit flag indicating the state of the Shift, Ctrl and Alt keys. The bit is set if the key is down. The following BitMasks may be used SHIFT_MASK 1 CTRL_MASK 2

	ALT_MASK	4
X ,Y	The current location of the mouse pointer. X and Y are always expressed in terms of the coordinate system set by the ScaleHeight, Scale Width, ScaleLeft, and ScaleTop properties of the object.	

KeyPress Event

Description

Occur when the user presses or releases an ANSI key while ALLText has the focus.

Syntax

Sub ALLText_KeyPress ([Index As Integer,]KeyAscii As Integer)

Parameters

Argument	Description
Index	Uniquely identifies a control if it is in a control array.
KeyAscii	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 that the object receives no character.

KeyDown and KeyUp Events

Description

Occur after the KeyDown and Before the KeyUp event when the user presses a key.

Syntax

Sub ALLText_KeyDown ([Index As Integer,]KeyCode As Integer, Shift As Integer)

Sub ALLText_KeyUp ([Index As Integer,]KeyCode As Integer, Shift As Integer)

Parameters

The KeyUp and KeyDown events use these arguments:

Argument	Description
Index	Uniquely identifies a control if it is in a control array.
KeyCode	A key code, such as, KEY_F1 (F1 key) and KEY_HOME (Home key). To specify key codes, use the constants in CONSTANT.TXT, a Visual Basic file that specifies system defaults.
Shift	This is a bit flag indicating the state of the Shift, Ctrl and Alt keys. The bit is set if the key is down. The following BitMasks may be used SHIFT_MASK 1 CTRL_MASK 2 ALT_MASK 4

Remarks

KeyDown and KeyUp are not invoked in response to the Enter key if the form has a command button with the Default property set to True.

Lost Focus Event

Description

Standard Event - Occur when the control loses focus.

Syntax

Sub ALLText_LostFocus ([Index As Integer])

GotFocus Event

Description

Standard Event - Occur when the control gets focus.

Syntax

Sub ALLText_GotFocus ([Index As Integer])

DblClick Event

Description

Standard Event - Occurs when the user quickly clicks twice in succession.

Syntax

Sub ALLText_DblClick ([Index As Integer])

Click Event

Description

Standard Event - Occurs when the user quickly clicks on the control.

Syntax

Sub ALLText_Click ([Index As Integer])