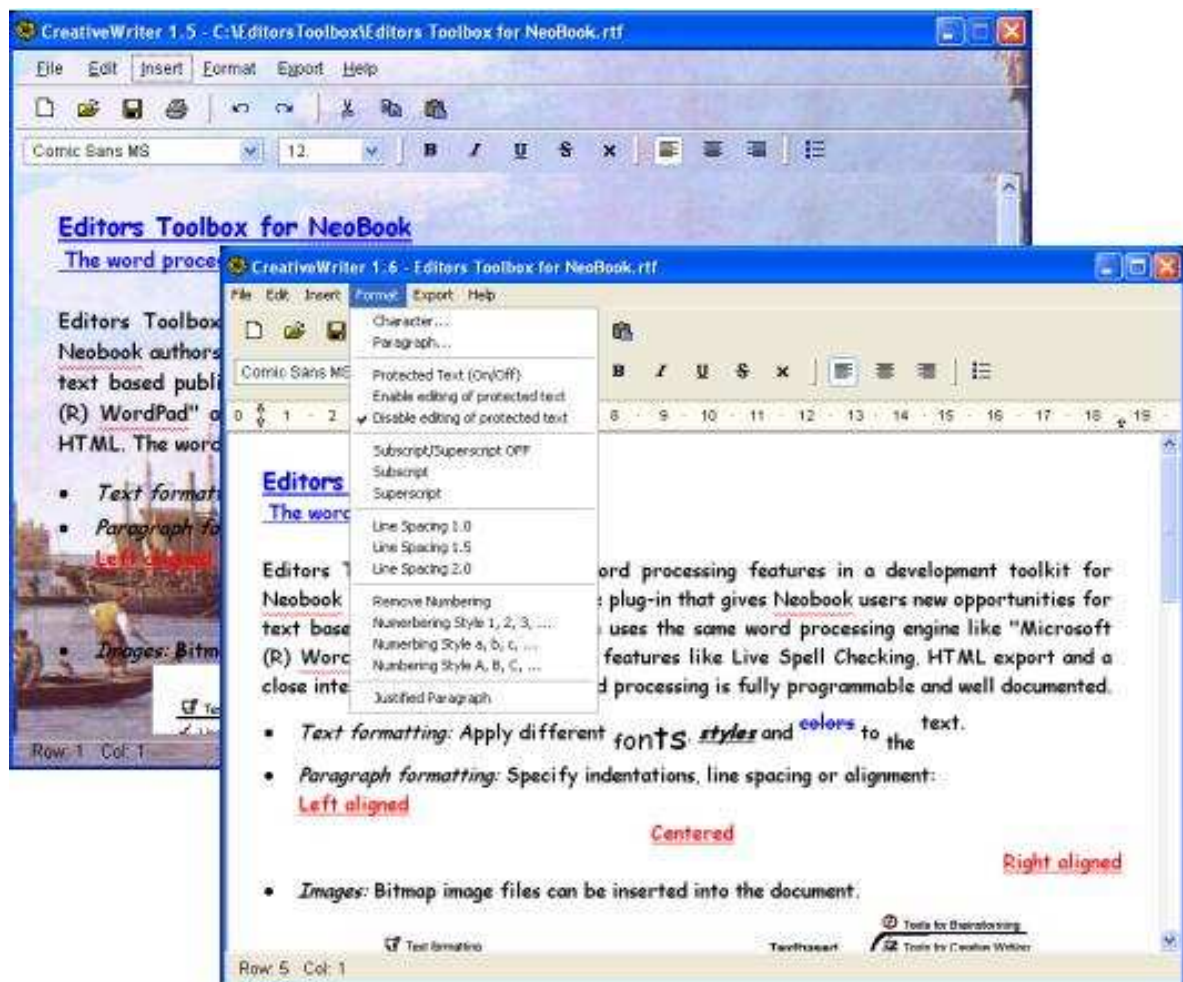


Add advanced word-processing to NeoBook

Editors Toolbox for NeoBook

Plug-in Version 1.75 for NeoBook 4.1.3a or higher
(NeoBook Version 5.5+ is recommended)



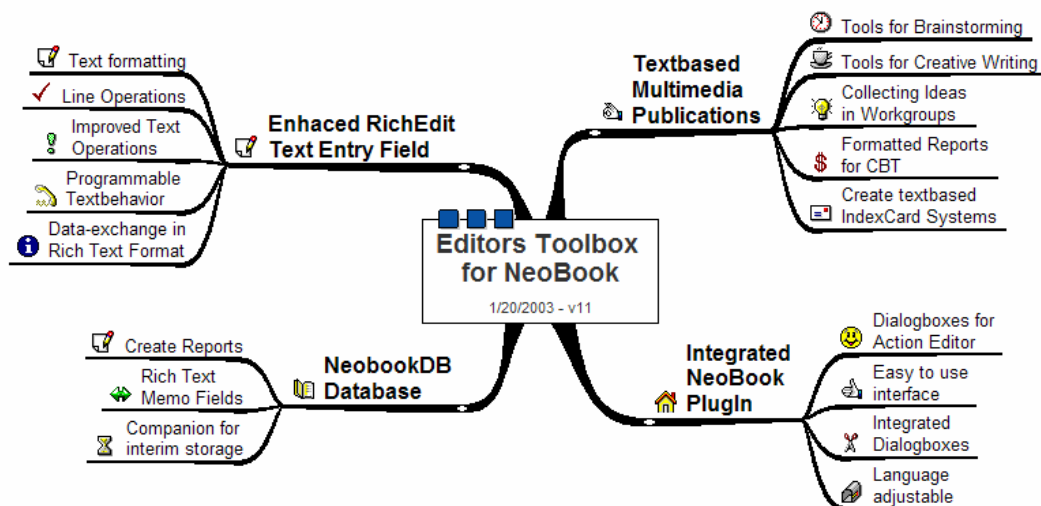
For NeoBook visit Neosoft Corp. at: www.NeoSoftWare.com

Editors Toolbox for NeoBook

The word processing plug-in

Editors Toolbox adds access to word processing features in a development toolkit for Neobook authors. It is an easy to use plug-in that gives Neobook users new opportunities for text based publications. The plug-in uses the same word processing engine like "Microsoft (R) WordPad" and offers advanced features like Live Spell Checking, HTML and Adobe Acrobat (R) PDF export. The word processing is fully programmable and well documented.

- *Text formatting:* Apply different **fonts**, **styles** and **colors** to the text.
- *Paragraph formatting:* Specify indentations, line spacing or alignment:
Left aligned
Centered
Right aligned
- *Images:* Bitmap image files can be inserted into the rich text document.



- *Embedded objects:* Objects like Microsoft Excel (R) worksheets can be inserted into the document with the clipboard pasted command or via dialog.
- *Live Spell Checker:* Your Editors Toolbox text window offers live spell

checking with red wiggly lines like Microsoft (R) Word. Free dictionaries in more than 20 languages are available from Addictive Software (R). You can also utilize Microsoft Word (R) custom dictionaries. Use the (optional) word list compiler from Addictive Software (R) to create your own dictionaries.

- *Export:* Your rich text documents and reports can be exported to a Neobook variable. Choose the output format with the options unformatted text, Rich Text source code, Line Mode and HTML. Export your rich text document as an "Adobe Acrobat (R) PDF" file. (Export filters do currently not support Images in HTML and PDF documents).
- *Database support:* Substitute the Neobook text fields with Editors Toolbox and store formatted rich text in a Database like NeobookDB (TM).
- *Cut/Copy/Paste operations/ Drag and Drop:* Formatting is preserved during these clipboard operations. Exchange contents with other programs with drag and drop. Paste special content which is linked to other applications into the text window.
- *Printing:* Allows printing of documents. Specify the page size and margins.
- *User Interface:* Smart programming interface for easy and powerful access to all word-processor functions. Use line commands to insert or manipulate rich text format documents programmatically to create reports. Call Neobook action script commands from within the rich text document using the Hyperlink function of the toolbox. Create read only text windows for displaying rich text files. Combine NeoBook object functions like HideObject and ShowObject with visual effects to show or hide the editor window.
- *Totally integrated into Neobook:* Dialog supported action commands for all toolbox functions. Built-in dialog boxes for load, save, print and formatting functions. Multi-language support. Shares Neobooks easy-to-use philosophy.
- *Documentation:* Editors Toolbox includes sample applications that demonstrate almost all functions of the toolbox. An extensive User Guide is included as an Adobe Acrobat (R) PDF-document.

Disclaimer - Agreement and using a free plug-in

Users of Editors Toolbox for NeoBook must accept this disclaimer of warranty:

“Editors Toolbox for Neobook is supplied as is. The author disclaims all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The author assumes no liability for damages, direct or consequential, which may result from the use of Editors Toolbox for NeoBook. Information in this document is subject to change without notice. You may distribute your NeoBook publications created with Editors Toolbox for NeoBook for any number of publications.”

Editors Toolbox for Neobook is free for registered Users of Neobook. It was created for our own text based NeoBook publications. Now it is offered to the Neobook community without charge and we hope you will have fun with the plug-in.

The CreativeWriter sample application is very basic and was intended as a reference publication to demonstrate the basic features of the plug-in. It was not intended as a standalone application. If you want to extend your action script knowledge and gather some more programming know-how on Neobook you can extend the sample application with some more word processing functions.

Editors Toolbox for Neobook was created using the ® Delphi programming language from ® Borland. It uses native code only. There is no Spyware or Adware or anything like this in the Editors Toolbox for NeoBook plug-in. It was developed and tested on Windows XP only.

You must check if Editors Toolbox for Neobook is suitable for your publication. If you want to use the plug-in in a commercial publication you must test the plug-in and the desired functions extensively before you use it in your publication.

As long as we use the plug-in for the development of our own commercial applications we will continue to fix problems and anomalies

that effect our publications. But we might stop development at any time so you cannot rely that there is support for the plug-in in the future. You might invest a lot of time using Editors Toolbox before you encounter an unsolvable problem or unacceptable limitation.

This UserGuide is created on the fly and is intended as a reminder basically for our own purposes. We cannot guarantee that the information provided in the guide is always up to date together with the plug-in and free of mistakes.

We strongly recommend that you read the “ReadMe” file which contains the latest information available to the plug in. You will find inside this file a list of enhancements and bug fixes as well as a list of known issues. We included also a list of frequently asked questions.

Table of Content

Disclaimer - Agreement and using a free plug-in	4
Introduction to word-processing with NeoBook.....	8
About NeoBook and Editors Toolbox	8
NeoBook Plug-In Interface	9
Microsoft ® Rich Edit Engine	9
Event Handling in a NeoBook Plug-in.....	10
Printing, Page size and Margins	11
Installation of Editors Toolbox for Neobook	12
General Programming Overview	12
CreativeWriter sample application	14
The sample publication and trouble shooting on start up.....	14
Programming Basics.....	15
Set-up a Rich Text Edit Control	16
Update the Format Bar and handling OnFormat Events	18
Call a popup menu with a click on the right mouse button.....	20
Store formatted text into a Database with the NeoBookDB plug-in	21
HTML conversion and NeoBook WebBrowser support	21
Hyperlinks to NeoBook and handling OnLinkClicked Events	22
Other programming samples.....	25
Global Variables	26
Overview of the Action Command Reference	30
Action Command Reference	34
teAddLine.....	34
teBackStyle	34
teBiDiMode	36
teCheckState	37
teClear	38
teClipboard	38
teClose	38
teDeleteLine	39
teExportToPDF.....	39
teFindText.....	40
teFirstIndent	41
teFontDialog.....	42
teGetAlignment.....	42
teGetFontList	43
teGetFontName.....	43
teGetFontSize.....	44
teGetFontStyle.....	44
teGetLine	45
teGetLineFromChar.....	45
teGetNumbering.....	45

teGetText.....	46
teGetTextLength	49
teGetTotalLines	49
teHideSelection.....	50
teInsertImage	50
teInsertObjectDialog.....	50
teInsertSpecial	51
teLeftIndent.....	51
teLineSpacing	52
teLink.....	53
teLoadFileDialog.....	54
teModified	55
teOpen	56
tePageSetupDialog	58
teParagraphDialog.....	59
tePlainText	59
tePrintDialog	60
tePrinterSetupDialog	60
teReAlign.....	61
teRedraw.....	61
teRichEditVersion	62
teRightIndent	62
teSaveFileDialog.....	63
teSelLength.....	64
teSelStart	64
teSelText.....	65
teSetAlignment	66
teSetFocus.....	66
teSetFontName	66
teSetFontSize	67
teSetFontStyle	67
teSetNumbering	68
teSetText	69
teSpellCheck	69
teSpellOptionDialog.....	72
teSpellSuggest	72
teTextColor.....	73
teUndo	74
teWordWrap.....	74
teWriteMode	75
Rich Edit Shortcut Keys.....	76
Copyright and Trademarks	78

Introduction to word-processing with NeoBook

About NeoBook and Editors Toolbox

NeoBook is a multimedia authoring system from NeoSoft Corp. to create and publish 32-bit Multimedia Windows Applications, Screen Savers or Internet Explorer ActiveX Controls.

Users who are familiar with other programming languages like C, Delphi or Visual Basic will appreciate the easy-to-use philosophy of NeoBook and its action script commands. NeoBook is not only helpful for creating multimedia applications with drag and drop of multimedia content. It includes a very easy programming language called “action scripts”. These action script commands are much easier to use than regular programming languages and you don’t have to care about classes, objects or the initialization of variables. It allows you to create multi-user database applications with a few action commands and a very easy interface to access database records. NeoBook compiles handy applications in one file, creates a setup program and is perfect for publishing over the internet.

Most Multimedia and Authoring Tools offer only basic text editing tools. The combination of authoring tools with advanced word-processing features offers interesting opportunities for multimedia applications and writing tools. NeoBook provides in its tool palette a multi-line text entry field for unformatted text with a limited programming interface. Many applications require more sophisticated word processing features and a full programming interface.

Editors Toolbox for NeoBook is a plug-in for NeoBook 4.1.3a or higher that allows adding advanced word-processing capabilities to NeoBook. The plug-in supports text formatting and extended features which are helpful to access and change formatted text documents programmatically. It offers a link interface to access NeoBook action commands and the NeoBook tool palette. Editors Toolbox has integrated dialog boxes for all standard features of a word-processor like printing, page set-up, format font, format paragraph, printer set-up, search and replace etc. It supports many functions for advanced text processing such as line operations and the integration of objects, such as excel worksheets and HTML conversion of the text document to a NeoBook variable.

Editors Toolbox comes with a user friendly programming interface that shares the easy-to-use philosophy of NeoBook. All action commands of the plug-in are supported by a dialog-box that makes it easy to use these functions. A sample text editor demonstrates how to integrate Editors Toolbox into a NeoBook Publication. Features like “toolbars”, “search and replace dialogs”, link interface, HTML or

Adobe ® Acrobat ® PDF file export (no images) or background pictures are realized with NeoBook objects or action script commands.

With NeoBook and Editors Toolbox it is fun to create word-processors, writing tools and text-based multimedia applications. It takes only a few mouse clicks to add word processing to an application. You can built editors that look like a classical application or use the advantage of the NeoBook multimedia features to built fancy editors.

NeoBook Plug-In Interface

NeoSoft introduced the plug-in interface with Version 4.0 of NeoBook. This plug-in interface was not originally designed to display visual content like database tables or a word-processor. In version 4.09 NeoBook extended the plug-in interface to use a "Rectangle" as a frame for visual content. The plug-in interface was improved in Neobook version 4.1.1a to support keyboard input and other features. Neobook version 4.1.1d added features for easier resizing of objects. Finally Editors Toolbox requires the plug-in interface of NeoBook 4.1.3a or higher which includes some bug fixes. This makes it possible to add word-processing to NeoBook through a plug-in tool like Editors Toolbox.

The main idea of Editors Toolbox is to offer an easy to use programming interface with a flexible set of action commands to access formatted text. This is provided in a single plug-in file that can be compiled to a standalone application with NeoBook.

Microsoft ® Rich Edit Engine

Editors Toolbox has no word-processing engine itself. The plug-in is based on the Microsoft Rich Edit Dynamic Link Libraries which are part of the Microsoft Windows installation. Editors Toolbox uses the same word-processor engine as Microsoft ® WordPad. This enables a relative small size of the Editors Toolbox plug-in.

Microsoft published different versions of the Rich Text Edit Control with different functionality. Editors Toolbox links itself automatically to one of the following Microsoft ® Rich Text Edit Control.

File names of the installed Rich Edit version DLL

Version	DLL-Name
1.0	Riched32.dll
2.0	RichEd20.dll
3.0	RichEd20.dll (same Name as Rich Edit 2.0)
4.1	MsftEdit.dll

Editors Toolbox checks on startup if an advanced Microsoft ® Rich Text Edit Control version like 2.0 or 3.0 or 4.1 is installed, if not the plug-in uses Microsoft ® Rich Text Edit Control version 1.0. Built-in dialog boxes of Editors Toolbox like format paragraph expand their functionality automatically to the current Rich Text Edit Control. Only the commands for line spacing, numbered lists, transparent background and non visible page breaks require Rich Edit 2.0 or 3.0 or 4.1 controls. For more details please refer to the action command reference. Note that the CreativeWriter sample application expands the menu options according to the installed Microsoft ® Rich Text Edit Control.

The following list describes which Rich Edit versions are included in which releases of Microsoft Windows:

Windows XP SP1	Rich Edit 4.1, 3.0 and Rich Edit 1.0 emulator
Windows XP	Rich Edit 3.0 and Rich Edit 1.0 emulator
Windows Me	Rich Edit 3.0 and Rich Edit 1.0
Windows 2000	Rich Edit 3.0 and Rich Edit 1.0 emulator
Windows NT 4.0	Rich Edit 2.0 and Rich Edit 1.0
Windows 98	Rich Edit 2.0 and Rich Edit 1.0
Windows 95	Rich Edit 1.0

At least you can rely on the fact that a Rich Edit 1.0 control is installed on the system that runs NeoBook publications. For example if you want to use the Rich Edit 2.0 Control you need Microsoft Windows 98 or higher on the target system. However you can update the Microsoft ® Rich Text Edit Control on the target system. Please take care that you don't violate the Copyright of ® Microsoft! See the Microsoft ® Website for more details to different Rich Edit versions.

Event Handling in a NeoBook Plug-in

Regular word-processing programming libraries need to respond to event messages generated in background by the control when the user edits text. These are events like paragraph change, format change, text change, input position change etc. The command interpreter of NeoBook is not designed to process large volumes of action commands as it would be necessary to process if the publication wants to respond to such events. Therefore the plug-in interface of Editors Toolbox supports only one regular event handler. This is the

format change event that is necessary for the “format toolbar” as you see it in the CreativeWriter sample application.

Note that you can respond to text change events indirectly if you set a variable of a NeoBook Text Entry Field to the same variable of a Rich Text Edit Control. You would do this in the Editors Toolbox `teOpen` command. The plug-in updates this variable on every text change. So, you can use the text change event of the text entry field to respond on a text change in the Editors Toolbox text window. Still you will find that it is not a good place to process a large number of action commands.

Printing, Page size and Margins

The text width depends normally on the dimensions of the control size. When you print a text document or export it to an Adobe® Acrobat®-PDF file you will notice that the text width is adjusted to the page size and page margins of the printer. Editors Toolbox offers a page set-up dialog where you can specify the page size and margins to the page border for printout. The plug-in calculates the number of pages which results from the page settings and displays the total number of pages in the print dialog. Unfortunately the Win32 Rich Text Edit Controls do not allow storing the page settings inside the text when the file is saved. The plug-in allows you to specify global variables to setup the page size and page margins that can be stored within the document with a work around. Read the section for global variables for more details.

Installation of Editors Toolbox for Neobook

Editors Toolbox includes an easy-to-use setup program. Usually you have downloaded the Editors Toolbox installation file from an internet website. The setup program assumes that the NeoBook Application Version 5.5+ is installed in your programs folder at “Neobook 5” or “Neobook 4” if you are using this older version. You can change the default installation folder of Neobook in the setup program. If you accept the default installation folder for Editors Toolbox then the setup program will install the plug-in into the Neobook “Plug-ins” folder. You can use the setup program also to remove the installation of Editors Toolbox.

Before you can use Editors Toolbox action commands in NeoBook you must install the Editors Toolbox plug-in into NeoBook. You would do the installation in NeoBook at “Options - Install plug-ins”. This NeoBook menu function opens a Windows file selector for the Neobook plug-in folder. If you have accepted the default installation folders for Neobook and Editors Toolbox you should see the folder “EditorsToolbox” in the file selector. Click on this folder and install the “EditorsToolbox10” plug-in file. If you have problems please consult the NeoBook user manual.

General Programming Overview

Once the plug-in is installed and visible in the action command list, the integration of the Rich Text Edit Control into a NeoBook publication is very easy. Just draw a rectangle on the screen. Then add the *teOpen* command of the plug-in to a button or to the Page Actions of the current publication page to open a word-processor window. The plug-in will display a dialog for the *teOpen* command where you must specify the name of the rectangle from a NeoBook object list and other (optional) settings depending which type of word processing you need. If you run the publication you can use the Rich Text Edit Control as if it were a native object of NeoBook.

The *teOpen* command allows you also to specify a variable that is used to store the text content of the control. This variable is updated when the user makes changes to the current document. Note that using this online variable slows down the system performance when the size document increases significantly. If you want to use Editors Toolbox to process large text documents up to 16.7 million characters you can not use a variable which is automatically updated while the user types text on the keyboard. Also you should not use links to Neobook action commands when you process large files because currently when you

load a file the initial start up process for links takes too much time. Please consult the *teOpen* command for more details how to set up Editors Toolbox for large files.

If you want to insert text through the programming interface of the plug-in you would use the *teSelText* or *teAddLine* functions. Use the file load function to load a complete text file in Rich Text Format or *teSetText* to set the complete text of the control.

You can store the current document into a NeoBook variable with the *teGetText* command and you can specify the output format like HTML (no images) or Rich Text Source Code. The option for rich text source code allows you to integrate formatted text into a database like the free NeoBookDB database plug-in. You can also build your own load and save procedures for your document with extended text features based on the rich text source code (read the section Global Variables on this). For more information on different file formats and how to store formatted text in a database see the *teGetText* command reference.

You can determine if the document was changed with the *teModified* variable. You can save a document with the *teFileSaveDialog* command and display an optional dialog box.

You can print the current document with the *tePrint* action command and display an optional dialog box. You can use the *tePageSetupDialog* to choose a paper format or change the page borders. The print function will calculate the resulting pages for the printout.

You can set the cursor to a text position with *teSelText*. Use line operations like *teAddLine* or *teDeleteLine* to handle text operations with text lines instead of handling groups of characters with *teSelStart*, *teSelLength* or *teSelText*.

Use the command *teWriteMode* to set up the control for *read only* operations or *lock* the visual update to perform text operations in background.

Apply a paragraph numbering style with the *teNumbering* command or a background picture with to the document. Set the line-spacing with the *teLineSpacing*. These commands require Rich Edit 2.0 or higher. Please consult the action command reference.

Use *teSpellCheck* to enable live spell checking of your documents while the user is typing text or use classical dialog boxes with multi language support for you dialog based spell checking.

Use *teExportToPDF* to export your document to an Adobe® Acrobat®-PDF file (Images are currently not supported). You will find that the export converter can produce very tiny pdf-documents.

CreativeWriter sample application

The sample publication and trouble shooting on start up

Editors Toolbox includes a NeoBook sample publication for a word-processor in the publication file “CreativeWriter.pub”. The sample demonstrates how to integrate a basic Rich-Text-Editor into NeoBook and gives you a lot of programming hints. In addition to the main CreativeWriter sample application is a very basic publication “BasicEditor.pub” that demonstrates basic features to create an editor window.

Note that you should increase the image cache size in NeoBook 4.1.1 or higher at “Book - Book Properties - Misc.” because the standard bar and the format bar include a large number of small images. You must check which value gives you a good performance on your system. We use a cache value of 60 on our (fast) testing system.

If you run the CreativeWriter sample publication you should see the “Editors Toolbox for Neobook” text file with a few words (red) underlined by the live-spell-check engine that might be unknown to the dictionary or just misspelled. If you don’t see the sample text file then this text file is not located in the Neobook publication folder [PubDir].

In case you see the sample text file with every word underlined with a red wiggly line from the live-spell-check engine this means that the default English language dictionary was not found. The sample publication expects the dictionary in the publication folder as it is stored in the NeoBook [PubDir] global variable. You can right click with the mouse on a misspelled word and locate your dictionary folder (which should currently reside in your Editors Toolbox installation folder). This will show a popup menu where you can change the dictionary folder with the “options” button. See the command *teSpellCheck* for more details.

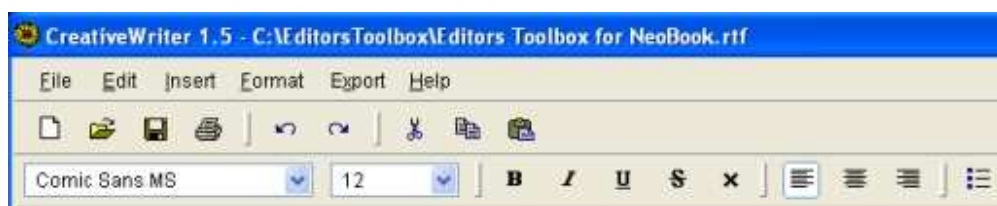
The CreativeWriter sample application is intended to demonstrate the programming interface of the plug-in and might help you when you find a bug or strange behavior in your own publications built with Editors Toolbox for NeoBook. It covers a wide range of word-processing requirements. The sample publication for Neobook v5+ is more advanced than the sample pub for Neobook v4.

It is strongly recommended that you first recompile your existing publication created with Editors Toolbox for NeoBook when you want to use an updated plug-in. We found an issue in Neobook 5+ that might be related with the debugger. Recompiling your publication with a new plug-in version of Editors Toolbox for Neobook seems to solve the problem. It also sets the [PubDir] directory to the dictionary files.

Programming Basics

You find different versions of the *teOpen* command in the CreativeWriter sample application in the action command section of the page properties. The default setting of the *teOpen* command is optimized to support very large text files. Check out a different *teOpen* command for a close interface to NeoBook with hyperlinks or just disable the “OnLinkClicked” event handler in the *teOpen* command to process large files.

The word-processor window is not visible at design time because plug-ins like Editors Toolbox will attach their visual content to a NeoBook Rectangle only at runtime.



Before we start to review the special action commands of Editors Toolbox we start with some basic explanations on how to simulate a Menu and a Format Bar with NeoBook.

Once you have loaded the sample publication into NeoBook you will notice that you cannot directly access all NeoBook objects on the screen because some are combined as groups of objects. This makes it much easier to keep an overview about the number of object on the screen. If you want to modify a single button, a combo box or add a new button to the menu you might need to call the Object List at “Options Show Object List”. The names of the objects should help to find the desired object. Click on the desired object in the list with the right mouse button and select “Object Properties”. If you start to make extended changes to the design of CreativeWriter you need to ungroup the object groups. Please remember the names of the groups.

When you want to modify the position of a grouped object on the screen you would select the object in the Object List and call the Object Size dialog on the main menu at “Edit Object Size”. Here you can adjust the size and screen position of a grouped object. If one object is on top of another, the Object List is the only way of accessing these non-visible objects.

There are different ways to design a Button Bar like the buttons for Bold, Italic or underline. When a button is clicked it stays selected until another click on the button releases the selection. NeoBook does not support such buttons directly. To achieve a fancy Windows XP look we place a white rectangle below a NeoBook Push Button. This rectangle is not visible. When the user clicks on the button we set the

rectangle to visible state. The Push Button itself has a transparent background and displays the rectangle when it gets visible.

Set-up a Rich Text Edit Control

Now we start to explain the CreativeWriter programming sample. Once you have loaded the sample publication you should take a look at the page properties. Press button F5 or click on the page property button on the button bar of NeoBook. In page properties the set-up of the Rich Edit Window and the Format Bar will be done.

The following *teOpen* action command attaches a Rich Text Edit Control to a NeoBook Rectangle.

```
teOpen "teRectangle" "[teRectangle_Text]"
"teRectangle_OnFormatChange" "" "" "" "Scrollbar"
"Inches" "Ruler"
```

The text content is stored in the variable [teRectangle_Text]. When the user makes changes to the text this variable will be updated. Therefore this *teOpen* setup is not intended for very large text files.

We have defined a subroutine at Book Actions Subroutines with the name "teRectangle_OnFormatChange". This subroutine is called by the plug-in and updates the Format Bar when a change of the current formatting is detected. This happens when the user changes the format or moves the cursor to another position in the text with different formatting.

During the set-up procedure we don't want to process any keyboard or mouse events. So we use the Suspend action.

```
Suspend "True"
```

Now we start to setup our Format Bar with the necessary information. First we read the installed system fonts and send the string list to a NeoBook Combobox with the Name teFontNameList on the Format Bar.

```
teGetFontList "teRectangle" "teFontNameList"
```

The NeoBook Combobox "teFontSizeList" was filled with font sizes in the action dialog of the ComboBox.

Next we read the current FontName and FontSize which was set internally by the plug-in with the default value "Arial - 12 pt" and send the value to the ComboBoxes. Note that we use the global variable [teFontName] and [teFontSize]. These variables are set in the

ComboBoxes as “Variable to Store selected Item”. When we change the value of these variables the selected items in the Comboboxes will be changed.

```
teGetFontName "teRectangle" "[teFontName]"
teGetFontSize "teRectangle" "[teFontSize]"
```

We load a sample document into the Rich Text Edit Control without showing an OpenFile Dialog. You must check if the file exists before loading the document. The sample uses the FileExists command. We use the global variable [PubDir] for the publication directory.

```
FileExists "[PubDir]Editors Toolbox for
NeoBook.rtf" "[Result]"
If "[Result]" "=" "TRUE"
    teLoadFileDialog "teRectangle"
"[PubDir]Editors Toolbox
for NeoBook.rtf"
endif
```

Finally we activate keyboard and mouse events and process the mouse and keyboard buffer.

```
Suspend „False+NoBuffer“
```

Next we call our subroutine to read the formatting information in the sample document at the current input position to update the Format Bar.

```
GoSub "teRectangle_OnFormatChange"
```

You will notice that we never directly call the Rich Text Edit Control. Instead we call the Neobook Rectangle which hosts the Rich Text Edit Control.

```
teSetFocus "teRectangle"
```

This is it! All other action commands are not active and demonstrate special aspects of the programming interface. You can activate this action commands to learn more about the programming of Editors Toolbox.

Update the Format Bar and handling OnFormat Events

When the user changes the current input position or the formatting of the text we need to check if the Format Bar in NeoBook needs to be updated. The plug-in internally keeps track of those events and calls the NeoBook subroutine "OnFormatChange" which can be easily accessed from the subroutines item of the Book menu.

While the Subroutine is working the plug-in takes care that keyboard input don't leads to an action buffer overflow. For example if the user scrolls with the cursor keys through the document every time a format change is detected the subroutine "OnFormatChange" is called. Using the Neobook Suspend command might not work correct (see ReadMe File - Known Issues) therefore the plug-in catches keyboard input while the subroutine is processing.

During design mode using the cursor keys in the editor window might be slow because of the debugging functions of Neobook. But after compiling the keyboard input shows an acceptable speed.

Next we have to check if a Rich Text Edit Control is attached to a NeoBook rectangle. If not we will exit the subroutine. Otherwise an error occurs for each Editors Toolbox action command in the subroutine. That will look very lousy. (Note that you can switch off the error handling in NeoBook at Book Properties).

```
SetVar "[teSelLength]" "-1"
teSelLength "teRectangle" "[teSelLength]"
If "[LastError]" ">" ""
    Suspend „False+NoBuffer“
    Return
endif
```

If text is selected it is not possible to update the format bar! This is very important and related to the selection change events of the NeoBook Comboboxes and Listboxes. We use global variables to update the current file name and file size on the format bar. A run of the OnFormatChange subroutine would change the selected text to the character formatting at the current input position. So we have to leave the subroutine if we find selected text.

```
If "[teSelLength]" "<>" "0"
    Return
Endif
```

In Neobook 5.0 or higher you can use the new IfEx command.

```
IfEx "[LastError]" > [#34][#34] or [teSelLength] <> 0"
    Return
```

```
endif
```

Now we will update the font name and the font size on the format bar. Therefore we have to update the global variables “[teFontName]” and “[teFontSize]”. Unfortunately these global ComboBox variables call a “selection change” event in the NeoBook ComboBox action. We don’t want this event when we update the Format Bar. Otherwise “teRectangle_OnFormatChange” is called several times. We use the global variable “[teNoComboBoxUpdate]” to inform the Combobox to suppress the “selection change” event.

```
SetVar "[teNoComboBoxUpdate]" "True"
teGetFontName "teRectangle" "[teFontName]"
teGetFontSize "teRectangle" "[teFontSize]"
```

Next we store the alignment of the current paragraph in the variable “teGetAlignment” and check for each button if it is the current alignment style. If not, we hide the white selection rectangle under the button. If we have found the alignment button we show the white selection rectangle.

```
teGetAlignment "teRectangle" "[teGetAlignment]"
If "[teGetAlignment]" "=" "Left"
    ShowObject "teLeftSel" "None" "0"
else
    HideObject "teLeftSel" "None" "0"
Endif
. . .
```

We will check the FontStyle at the current input position in the same way as the alignment style.

```
teGetFontStyle "teRectangle" "Bold"
"[teGetFontStyle]"
If "[teGetFontStyle]" "=" "Bold"
    ShowObject "teBoldSel" "None" "0"
else
    HideObject "teBoldSel" "None" "0"
Endif
. . .
```

Finally we examine the numbering style which is currently only bullets on the Format Bar because bullets is the only numbering style which is supported by all Rich Text Edit Controls (because it requires Rich Edit 1.0). The menu of the sample application adjusts itself to the current installed control and hides invalid features. We have to check

the keyword which is returned if we check the current paragraph numbering style. If it is “Bullets” then we show the white rectangle under the button. In not we will hide the white rectangle.

```
teGetNumbering "teRectangle" "[teGetNumbering]"
If "[teGetNumbering]" "=" "Bullets"
    ShowObject "teBulletsSel" "None" "0"
else
    HideObject „teBulletsSel" "None" "0"
Endif
```

You might find a solution to make the updating of the “FormatBar” faster with less action commands in 5+. It might look like the following example but this result looks flickering:

```
teGetAlignment "teRectangle" "[teGetAlignment]"
IFEx "[teGetAlignment] = Left or [teGetAlignment] = Center
or [teGetAlignment] = Right"
    HideObject "teLeftSel" "None" "0"
    HideObject "teCenterSel" "None" "0"
    HideObject "teRightSel" "None" "0"
    ShowObject "te[teGetAlignment]Sel" "None" "0"
endif
```

Before we leave the subroutine we need to set the global variable “[teNoComboBoxUpdate]” to false to allow to select a font name and font size on the Format Bar.

```
SetVar "[teNoComboBoxUpdate]" "False"
```

Changing the OnFormat Event subroutine is very tricky and may have unexpected results. Note that the teWriteMode command has to options (EnableEvent, DisableEvent) to stop OnFormat change subroutine calls.

Call a popup menu with a click on the right mouse button

This example gives a hint how to create a basic popup window with a click on right mouse button. The example shows a list of suggestions for a misspelled word with the Neobook MenuEx command. If live spell checking is active then the live spell check dialog is suppressed and this self made subroutine is called.

If you want to create your own popup menu then you must first set the global variable “Rectangle.RightButtonAction” with an action command. This action command is called each time the user clicks the

right mouse button within the editor window. This variable must be set before calling the `teOpen` command.

```
SetVar "[teRectangle.RightButtonAction]" "GoSub  
[#34]RightMouseDown[#34]"
```

A right mouse click will call the subroutine "RightMouseDown". Now you must create this subroutine. In Neobook call "Book - Subroutine - New" and enter "RightMouseDown". Neobook creates the header and footer for this subroutine.

See the creative writer sample application for details. You will find a basic example that shows the principal. Note that you must switch on a custom popup menu at "Page - Properties" first. Look out for the "RightButtonAction" variable which is disabled by default.

The sample code uses the Neobook popup menu command `MenuEx` to create a popup menu. Please note that this Neobook menu command is limited for a popup menu. Unlike a normal popup menu the Neobook menu commands need an extra mouse click to get closed. So you would need two mouse clicks to jump from one misspelled word to another misspelled word. This also confuses the Rich Edit Control and it can happen that the selected word will be moved.

Store formatted text into a Database with the NeoBookDB plug-in

The free NeoBookDB ® plug-in allows creating a Database with Neobook. Editors Toolbox makes it possible to store formatted text into a Database Memo or Textfield.

Substitute the Neobook Textbox with a textbox created with Editors Toolbox. Use the command `teGetText` to store the current text of the Editors Toolbox textbox as Rich Text Source Code. This rich text source code can be placed into a Memo or Text Field of a Database. Please see the action command reference of the `teGetText` command for more information and programming examples.

HTML conversion and NeoBook WebBrowser support

If you want to export your current document to a HTML file you must use the `teGetText` command. The result is stored in a NeoBook variable. You can save the html content of this variable with the `WriteLine` command with the file extension *.htm. This html-file can be shown in the NeoBook WebBrowser Tool.

Editors Toolbox allows you to add HTML commands to your formatted rich text document. When you convert your document to HTML this commands are preserved in the HTML document.

This gives you limitless possibilities to enrich your document with HTML elements like tables, pictures, links, outlining or calls of NeoBook action commands. The Creative Writer sample application shows examples in the “Creative” menu to insert tables, pictures and calls of NeoBook action commands.

Note that the character “<” is interpreted by the HTML converter as a start of a HTML command.

To insert a picture to the document just add the following line. Note that HTML expects pictures in the format GIF, JPG or PNG:

```
<IMG SRC="picturename.ext">
```

The following example adds a link with a subroutine call to NeoBook. If the user clicks in the NeoBook WebBrowser on the linked text the subroutine “HTML_CallSubroutine” is called and displays an AlertBox with a message:

```
<a href="neobook:GoSub
%22HTML_CallSubroutine%22">Call a subroutine in
NeoBook:</A>
```

Note that you need to use “%22” for inner quotes. If you want to add this call with the `teAddLine` command from within an action script the call would look like this. You must use [#34] for the outer quotes:

```
teAddLine "Rectangle1" "<a
href=[#34]neobook:GoSub
%22HTML_CallSubroutine%22[#34]>Call a subroutine in
NeoBook:</A>"
```

Editors Toolbox and the NeoBook WebBrowser Tool gives an easy-to-use access to the word of HTML.

Hyperlinks to NeoBook and handling OnLinkClicked Events

Editors Toolbox supports a link interface to NeoBook from within the document. You can add hyperlinks programmatically to selected text with the `teLink` command. Linked text is recognized by the blue text color and the underlined font style when it is loaded from an extern file. Please note that this parsing for hyperlinks on file load operations takes some time on long documents.

If the user hovers with the mouse pointer over the linked text the mouse pointer changes to a “hand”. If the linked text is clicked the plug-in calls a subroutine in NeoBook. This subroutine name must be specified in the *teOpen* command.

When linked text is clicked the subroutine receives information about the caller of the subroutine (mouse buttons or HTML converter), the linked text and in case of a mouse button call the position of the linked text in the document and which button was clicked. For more details please consult the *teLink* command.

This example adds a link to a document. It assumes that the user has selected some text in the document. After text was converted to linked text the selection must be set to zero.

```
teLink "teRectangle" "SELECTION"  
SetVar "[teSelLength]" "0"  
teSelLength "teRectangle" "[teSelLength]"
```

If you want to use links to jump to another position in the document (targets) you can only use the limited functionality to search for the text position with the *teFindText* command when a link was clicked. In this case it is better to add a HTML link to a target and convert you document to HTML to be shown and executed in the WebBrowser object.

If a text document is converted to HTML (see *teGetText* command) the converter calls this subroutine when a link was detected during conversion. This allows to decide what happens with linked text on conversion. Change linked text to HTML commands, delete it or just ignore it. In this case the *Source* variable contains the keyword “converter”. The “HTML Converter Demo” in the CreativeWriter sample application changes the linked text of the sample document to linked text that responds to a mouse click in the WebBrowser Tool of NeoBook. See the following details of the subroutine that can be found in NeoBook at “Book - Subroutine - *teRectangle_OnLinkClicked*”:

First we check the variable “*teRectangle_OnLinkClicked.Source*” which was set by the plug-in to inform who called the subroutine. That can be a click on a mouse button or the HTML converter.

This following code checks if the HTML converter has found linked text. The Variable “*teRectangle_OnLinkClicked.Text*” contains the linked text that was found. We want to convert the NeoBook action command for a custom window to HTML: CustomWindow “Demo Link to NeoBook” “-1” “-1” “*teLinkTextDialog*” “ToolWindow+Exclusive”

Note that you can not use quotes directly. You must use [#34] for the outer Quotes and %22 for the inner quotes. This is required by NeoBook. We use the Variable “*teWebLink*” to store the HTML code. This makes it easier if you want to review your result in a NeoBook

message window (AlertBox). Finally we set the new value for the linked text variable *“teRectangle_OnLinkClicked.Text”*. When we leave the subroutine the text value is passed to the converter by the plug-in.

```

    If "[teRectangle_OnLinkClicked.Source]" "=" "Converter"
        SetVar "[teWebLink]" "<a href=[#34]neobook:CustomWindow
%22Demo Link to NeoBook%22 %22-1%22 %22-15%22
%22teLinkTextDialog%22
%22ToolWindow+Exclusive%22[#34]>[teRectangle_OnLinkClicked.Text]<
/A>"
        SetVar "[teRectangle_OnLinkClicked.Text]" "[teWebLink]"

```

Next we check if the call comes from within the rich text document when the user clicked with a mouse button on the linked text. If the variable *“teRectangle_OnLinkClicked.Source”* contains “0” the left mouse button was clicked or “1” for right and “2” for middle mouse button.

A click on the left mouse button calls a NeoBook CustomWindow and a click on the right mouse button displays the linked text in a (NeoBook) text entry field to edit or delete the link. First we check for the left mouse button.

```

    else
        If "[teRectangle_OnLinkClicked.Source]" "=" "0"
            CustomWindow "Demo Link to NeoBook" "-1" "-1"
            "teLinkTextDialog" "ToolWindow+Exclusive"
            . Set the selected text manual to 0 for Rich Edit 1.0
            SetVar "[teSelLength]" "0"
            teSelLength "teRectangle" "[teSelLength]"
            ShowObject "teRectangle" "None" "0"

```

Next we check for the right (or middle) mouse button to call a NeoBook dialog in a custom window to edit the linked text. We use the link variables *“start”* and *“end”* to select the linked text. The variable of the text entry field is set to the variable for the linked text *“teRectangle_OnLinkClicked.Text”*.

Finally we set the focus to the Rich Text Edit Control and leave the subroutine.

```

    else
        teSelStart "teRectangle"
        "[teRectangle_OnLinkClicked.Start]"
        Math "[teRectangle_OnLinkClicked.End] -
[teRectangle_OnLinkClicked.Start]" "0" "[teSelLength]"
        teSelLength "teRectangle" "[teSelLength]"
        CustomWindow "Edit Link" "-1" "-1" "teLinkGroup"
        "ToolWindow+Exclusive"
    endif
    teSetFocus "teRectangle"
endif

```

Return

Other programming samples

You will find a sample for a “find dialog” and a “search and replace dialog” in the CreativeWriter sample application. The grouped dialog boxes are located at the page “dialogs”. The functions demonstrate how to integrate these elements with action commands and Neobook Tools.

If you call the help menu you will find a demonstration of more programming samples on different topics. With the demonstrations and with this documentation you should be able to integrate advanced word-processing into Neobook.

Another source for more information how to use Editors Toolbox is the included ReadMe file. It contains a section for frequently asked questions that show some nuts and tricks.

Global Variables

The plug-in of Editors Toolbox defines some global variables and subroutines for different purposes:

1. Variables created when using hyperlinks. A subroutine must be specified to handle link events. See the action commands `teLink` and `teOpen` for details.

“SubroutineName.Start”: Start position of linked text in the document

“SubroutineName.End”: End position of linked text in the document

“SubroutineName.Text”: Linked text

“SubroutineName.Source”: Information about the caller of the link. If it is the HTML converter that found a link on conversion the variable returns the keyword “Converter”. If the mouse button was clicked on linked text the variable contains 0 for left mouse button, 1 for right mouse button and 2 for middle mouse button.

2. Variables created with the `teOpen` command.

“RectangleName.Row”: Line number of the current input position

“RectangleName.Column”: Column number in the line of the current input position. See Status Bar in the creative writer sample application.

„RectangleName.PluginVersion“: Release version of the plug-in

3. Variables created with `teSaveFileDialog` and `teLoadFileDialog` commands when a dialog box is displayed.

“RectangleName.FileName”: Name of selected file when the user makes a valid selection in a save or load file dialog box.

“RectangleName.CurrentDir”: Stores the name of the current directory folder selected by a user. Both dialogs check initially if the NeoBook global variable “CurrentDir” has a value. If so they use the directory specified in this variable. See the CreativeWriter sample application for details how to set the file dialogs to the current directory.

4. Variables related to the Ruler. You must set a value to this variables before using the command `teOpen`.

“RectangleName.Ruler.Flat”: Use a flat ruler (true/false)

- “RectangleName.Ruler.Adjustable”: Enable/disable adjusting the page margins with the mouse (true/false).
- “RectangleName.Ruler.PageWidth”: Set a value in metric millimeters for the PageWidth.
- “RectangleName.Ruler.RightMargin”: Set a value for the ruler’s right margin. If you set this value for right margin then the value is ignored for printing or page setup dialog. It is visual on screen only. Set values in metric millimeters.
- “RectangleName.Ruler.LeftMargin”: Set a value for the ruler’s left margin. If you set this value for left margin then the value is ignored for printing or page setup dialog. It is visual on screen only. Set values in metric millimeters.
- “RectangleName.Ruler.RulerColor”: Set the color of the Ruler. Values for red, green, blue expected in one string separated with a commas.
- “RectangleName.Ruler.MarginColor”: Set the color of the page margins. Values for red, green, blue expected in one string separated with a commas.
- “RectangleName.Ruler.BackgroundColor” : Set the color of the background of the ruler. Default color is the windows menu color. Values for red, green, blue expected in one string separated with a commas.
- “RectangleName.Ruler.LineMarkerColor” : Set the color for a marker for the start and end of a line. Values for red, green, blue expected in one string separated with a commas.

```
. /// Example to set the Ruler Colors:
SetVar "[Rectangle.Ruler.BackgroundColor]" "236,233,216"
SetVar "[Rectangle.Ruler.MarginColor]" "248,236,236"
SetVar "[Rectangle.Ruler.RulerColor]" "248,240,240"
SetVar "[Rectangle.Ruler.LineMarkerColor]" "248,230,230"
```

5. Variables created with teOpen and tePageSetupDialog for the page size and the margins of the current document (for printing or Adobe® Acrobat® PDF export). The variable expects values in metric millimeters.

- “RectangleName.PageWidth”: Page width
- “RectangleName.PageHeight”: Page height
- “RectangleName.PageTop”: Page top margin to text
- “RectangleName.PageLeft”: Page left margin to text
- “RectangleName.PageRight”: Page right margin to text
- “RectangleName.PageBottom”: Page bottom margin to text

You can write to these variables to specify the page size and margins of the current document. You must specify the values

before using the `teOpen` command. The command `teOpen` checks if values are assigned to these variables and reads the values. Otherwise the page size is "A4".

Example for "A4" Page size and 25 mm margins:

```
SetVar "Rectangle1.PageWidth" "210"  
SetVar "Rectangle1.PageHeight" "297"  
SetVar "Rectangle1.PageTop" "25"  
SetVar "Rectangle1.PageLeft" "25"  
SetVar "Rectangle1.PageRight" "25"  
SetVar "Rectangle1.PageBottom" "25"
```

You can also specify values before using the command "`teExportToPDF`" together with the "Default" value options of this command. See command `teExportToPDF` for more information.

Unfortunately the Microsoft ® Rich Edit Engine does not store the values for page size and page margins within the document. Following might be an idea for a work around: If you want to save these values within the document you have to control the process of loading and saving a document on your own. Use the command "`teGetText`" to get the Rich Text Format Source Code of the current document and save your text document in the variable to a file with NeoBook action commands. Before saving use the NeoBook string manipulation commands to store the page size value within the document. On loading use the NeoBook command to load the rich text content of the file into a variable, read and setup the page size values and margins and place the document into the Editor Window with the command "`teSetText`".

6. Misc global variables. You must set a value to this variables before using the command `teOpen`.

"RectangleName.AllowResize": When the Rectangle is resized to which Editors Toolbox is attached then the editor window is resized automatically. Set this variable to False to disable automatic resizing. To resize manually use the command "`teReAlgin`".

"RectangleName.WinHandle": Handle of the Rich Edit Control.

"RectangleName.SpellNoDialogOption": Suppress in a live spelling popup menu a menu option to call a spell checking dialog. Set this variable to True.

"RectangleName.SpellCheckHWND": You can specify the window handle of a Microsoft Rich Edit Control to be spell checked. This function is intended for advanced NeoBook developers and may

have unpredicted results. Note that you are not allowed to create your own spellcheck engine! See the “teSpellCheck” command for more details.

7. Event driven variables

“RectangleName.RightButtonAction”: Call a Neobook action command on right mouse click in the editor window. This action command can be used to call a popup menu.

“RectangleName.LeftButtonAction”: Call a Neobook action command on left mouse click in the editor window.

This example calls the subroutine “RightMouseDown” on a right mouse click. See “Popup Menu” section of this manual for more details:

```
SetVar "[teRectangle.RightButtonAction]" "GoSub  
[#34]RightMouseDown[#34]"
```

* Replace the word “RectangleName” with the name of the rectangle to which Editors Toolbox is attached.

Overview of the Action Command Reference

Open/ Close

teOpen	Attach a Rich Text Edit Control to an existing NeoBook Rectangle.
teClose	Close the Text Control window and remove it from an existing NeoBook Rectangle

Text Operations

teSetText	Write (and replace existing) text into the current rich text document. You can use Neobook variables.
teGetText	Store the complete text of the current Rich Edit document in a NeoBook variable or get the word at the current input position. You can specify the output format like HTML, RTF source code or unformatted text.
teSelStart	Set or get start position of selected text or get the current input position
teSelLength	Set or get the number of characters selected in the document
teSelText	Set or get selected text
teClipboard	Perform clipboard actions.
teCheckState	Check CanUndo, CanRedo, CanPaste or count the number of words in the text.
teUndo	Undo or redo the last text operations
teLink	Changes selected text to hyperlink text. Use this command to add hyperlinks between text in a document and NeoBook. When the mouse hovers over the linked text the mouse pointer changes to a "hand"
teFindText	Search for Text in the document.
teInsertImage	Insert a Jpeg image file (picture) into the document at the current input position
teInsertSpecial*	Insert special functions at the current input position. Currently the function only inserts a non visible page break.
teExportToPDF	Export the current document to an Adobe ® Acrobat ® PDF File. (Images are not supported).

Line Operations

teAddLine	Add a line of text at a specified line number.
teGetLine	Get text from a specified line number
teDeleteLine	Delete a specified line of text
teGetTotalLines	Get total number of lines in the text
teGetLineFormChar	Get line index of current input position

Character Formatting

teSetFontStyle	Set font style for current input position or text selection
teGetFontStyle	Get the font style for current input position
teSetFontName	Set the fontname for current input position or selection
teGetFontName	Get the name of the font at the current input position
teSetFontSize	Set fontsize for current input position or selection
teGetFontSize	Get the font size of the character at current input position
teGetFontList	Fill an existing NeoBook Listbox or Combobox with fontnames of the current computer system

Paragraph Formatting

teSetAlignment	Set paragraph alignment (Left, Center, Right)
teGetAlignment	Get the alignment style of the current paragraph
teSetNumbering*	Set numbering style for the paragraph at the current input position
teGetNumbering*	Get the paragraph numbering style for the paragraph at the current input position
teLineSpacing*	Get or set the line spacing for the paragraph at the current input position. The measurement unit is pixel
teLeftIndent	Get or set left indent of paragraph at the current input position. The measurement unit is pixel
teRightIndent	Get or set the right indentation for the paragraph at the current input position. The measurement unit is pixel
teFirstIndent	Get or set the indent of the first line of a paragraph at the current input position. The measurement unit is pixel.

Integrated Dialogboxes

teFontDialog	Display a Windows standard dialog to set the font style for the current input position
teParagraphDialog	Display a dialogbox to setup the paragraph formatting style
tePageSetupDialog	Call a Windows standard dialog to set-up the page settings for printing. The settings will not be stored inside the text file when the document is saved
tePrintDialog	Print current document and display an (optional) dialogbox
tePrinterSetupDialog	Display a Windows standard dialog box for printer set-up
teLoadFileDialog	Load a textfile and display an (optional) dialogbox
teSaveFileDialog	Save a textfile and display an (optional) dialogbox
teInsertObjectDialog	Call a Windows standard dialog to insert an object to the rich text document. Objects can be linked or embedded to the document

Spell Checking

teSpellCheck	Activate live spell checking and show red wiggly lines under misspelled words. A right mouse click on a misspelled word shows a popup menu with a wordlist for correction and more options like setup and activation of dictionaries. Alternatively call a spell checking dialog box.
teSpellSuggest	Get suggestions for a misspelled word or check if a word is correctly spelled.
teSpellOption Dialog	Call an option dialog to set up spelling options.

Generall Settings

teClear	Deletes the complete text
teModified	The plug-in internally keeps track if the user makes changes to the current document. Use teModified to get or set the current state of the modified variable
teWordWrap	Specify the word wrap style. Wrap words at the window border or handle text as one line
teWriteMode	Specifies the editing mode of the Text Control. Valid modes are read only mode, normal writing and editing mode, a locked window to prevent

	refreshing and redrawing of the screen, Unlock Window for refreshing and redrawing, Disable or Enable Keyboard Input, Disable or Enable Format Change Events calling the Subroutine 'OnFormatChange', Enable or Disable editing and deleting of protected text, Show or Hide the Ruler.
teHideSelection	Specifies whether a text selection is hidden when the Text Control loses the input focus
teRedraw	Redraw the contents of the Text Control
teReAlign	Aligns a Rich Text Edit Control to the Rectangle. Use this command after the Rectangle was resized with the NeoBook SizeObject command in Neobook versions prior to 4.1.1d. teReAlign does not work with CustomWindows. If you use NeoBook version 4.1.1d or higher the Rich Text Edit Control is resized automatically when the attached rectangle is resized. You don't need to call teReAlign. (See also global variable [Rectangle.AllowResize]).
teSetFocus	Set the keyboard input focus to the Text Control
tePlainText	Ignore the internal text format when loading or saving a text file and interpret all formatting information as normal text or rich text.
teTextColor	Set the text color
teBackStyle*	Set or get the background color of the Rich Text Edit Control or set a background picture
teGetTextLength	Store the number of characters in the current rich text document to a variable
teBiDiMode*	Set bi-directional mode according to the installed language system
teRichEditVersion	Store the active Rich Text Edit Control version in a variable. The Microsoft ® Rich Text Edit Control 1.0 has a limited functionality and does not support line spacing and numbered paragraphs. Use this function to determine which Rich Text Edit Control is running.

* This commands require Rich Edit 2.0 or higher. See reference for details.

Action Command Reference

teAddLine

Purpose: Add a line of text at a specified line number.

Category: Line Operations

Syntax: teAddLine "Rectangle" "LineNumber" "Text"

Specify the name of a NeoBook Rectangle. Specify a value for *LineNumber* to add text. Use the keyword -1 for *LineNumber* to append text to the end of the document. Specify a text string for *Text*.

Example: Append a line of text ("Hello World") to the end of the document.

```
teAddLine "Rectangle1" "-1" "Hello World"
```

teBackStyle

Purpose: Set or get the background color of the Rich Text Edit Control or set a background picture

Category: General Settings

Syntax: teBackStyle "Rectangle" "Color" "Variable"

Specify the name of a NeoBook Rectangle. Enter a color with RGB values for Red, Green and Blue. Use the action command dialog for teBackStyle to specify the color with the select color dialog. If the value for color is -1 then the variable returns the RGB color value for the text at the current input position.

You can specify a jpg or bitmap picture as a background for the Rich Text Edit Control. Note that Rich Edits doesn't support a transparent background regularly. This functionality is achieved with a trick for Rich Edit 2.0 or higher. You might need a redraw of the text window or a newly ShowObject for the Rectangle depending on your purpose.

You can set a special value for "Variable" to determine how a picture will be displayed. Some Keywords accept a number value to adjust the display style: FishEye, Emboss, Spray, Posterize, Saturation, Brightness, Contrast and PencilDrawing. Use an Empty Variable for a regular display style. Check out the Menu Option "Insert - Insert Background Image" and the "Idle-Event" in the Creative-Writer Sample Application.

Following you find the Keywords that you can set as a Option for “Variable”:

```
SetVar "[teBackStyle]" "Tile"
SetVar "[teBackStyle]" "Center"
SetVar "[teBackStyle]" "Stretch"
SetVar "[teBackStyle]" "Sharpen"
SetVar "[teBackStyle]" "Soften"
SetVar "[teBackStyle]" "SoftenLess"
SetVar "[teBackStyle]" "Greyscale"
SetVar "[teBackStyle]" "EdgeEnhance"
SetVar "[teBackStyle]" "Blur"
SetVar "[teBackStyle]" "Invert"
SetVar "[teBackStyle]" "AntiAlias"
SetVar "[teBackStyle]" "Laplace"
SetVar "[teBackStyle]" "HiPass"
SetVar "[teBackStyle]" "Chalk"
SetVar "[teBackStyle]" "DraftOutlines"
. +++ These Keywords accept an Value (1-255)
SetVar "[teBackStyle]" "FishEye=2"
SetVar "[teBackStyle]" "Emboss=2"
SetVar "[teBackStyle]" "Spray=2"
SetVar "[teBackStyle]" "Posterize=6"
SetVar "[teBackStyle]" "Saturation=6"
SetVar "[teBackStyle]" "Brightness=2"
SetVar "[teBackStyle]" "Contrast=2"
SetVar "[teBackStyle]" "PencilDrawing=2"
```

Unsupported and limited function not intended for final publication: To show the Editors Toolbox text window transparent use the Keyword “Transparent” for “Color”. This function is currently very limited and not recommended to be used in final publications. Formatting and editing is very limited. The mouse can not be used. Live spell checking while typing in text is not recommended. The screen refreshing is flickering and not satisfying. If you want to try out this function please note that you must also set the “Rectangle” to Hollow/Transparent to show the current Neobook Objects under the “Rectagle”.

```
teBackStyle "teRectangle" "Transparent" "[teBackStyle]"
. Refresh screen
ShowObject "teRectangle" "None" "0"
teSetFocus "teRectangle"
```

Example: The example gets the background color at the current input position and stores the result in the variable [teBackStyle]. If the background color is white (255,255,255) then the background color is changed to blue (0,0,255).

```
teBackStyle "Rectangle1" "-1" "[teBackStyle]"
If "[teBackStyle]" "=" "255,255,255"
```

```
teBackStyle "Rectangle1" "0,0,255"
"[teBackStyle]"
Endif
```

This example attaches a simple rich text window to an existing Neobook Rectangle and adds a background picture on startup of the current Neobook Page. Place this code in Neobooks Page Properties Action.

```
teOpen "Rectangle1" "" "" "" "" "" "" "Inches" ""
"Ruler"
ShowObject "Rectangle1" "None" "0"
teBackStyle "Rectangle1" "BitmapFile.bmp"
"[teBackStyle]"
```

This example adds a background picture to an existing rich text window when you press a button. Place this code in Neobooks Button action.

```
teBackStyle "Rectangle1" "BitmapFile.bmp"
"[teBackStyle]"
```

This example attaches a simple and colorful rich text editor to an existing Neobook Rectangle on startup, inserts some text and displays a background picture. After teBackStyle we need to redraw the Rectangle to display the text. Place this code in Neobooks Page Properties Action.

```
teOpen "Rectangle1" "" "" "0,64,128"
"255,255,255" "" "" "Inches" "" "Ruler"
teSetText "Rectangle1" "This is some sample
text."
teBackStyle "Rectangle1"
"C:\EditorsToolbox\pic1.bmp" "[teBackStyle]"
teRedraw "Rectangle1"
```

teBiDiMode

Purpose: Set bi-directional mode according to the installed language system

Category: General Settings

Syntax: teBiDiMode "Rectangle" "BiDiMode"

Specify the name of a NeoBook Rectangle. Use the keywords "LeftToRight", "RightToLeft", "RightToLeftNoAlign" or

“RightToLeftReadingOnly”. This command requires RichEdit 3.0 versions or higher.

Example: `teBidiMode "Rectangle1" "RightToLeft"`

teCheckState

Purpose: Check CanUndo, CanRedo, CanPaste or count the number of words in the text.

Category: General Settings

Syntax: `teCheckState "Rectangle" "CheckThis" "Variable"`

Specify the name of a NeoBook Rectangle.

With CheckThis you can define what you want to check:

CanUndo - Undo available

CanRedo - Redo available

CanPaste - Check if the clipboard contains content to paste.

Variable - Returns “TRUE” or “FALSE” for the options above.

WordCount - Count the number of words in the current text document.

Example: The example checks if the clipboard contains content to paste. With regard to the result it enables or disables the menu items.

```
teCheckState "teRectangle" "CanPaste"
"[teCheckState]"
If "[teCheckState]" "=" "FALSE"
    DisableMenuItem "MenuPaste"
else
If "[teCheckState]" "=" "TRUE"
    EnableMenuItem "MenuPaste"
endif
```

This example counts the number of words in the text and displays an Alertbox with the result.

```
teCheckState "teRectangle" "WordCount"
"[teCheckState]"
AlertBox "Count Words" "The text contains
[teCheckState] words."
teSetFocus "teRectangle"
```

teClear

Purpose: Deletes the complete text.

Category: General Settings

Syntax: teClear "Rectangle"

Specify the name of a NeoBook Rectangle.

Example: teClear "Rectangle1"

teClipboard

Purpose: Perform clipboard actions.

Category: Text Operations

Syntax: teClipboard "Rectangle" "Options"

Specify the name of a NeoBook Rectangle. Use the following keywords in options:

"Cut" - Cuts out selected text and copies it to the clipboard

"Copy" - Copies the selected text to the clipboard

"Paste" - Pastes text or objects from the clipboard

"PasteContent" - Calls a Windows dialog to paste content

"Clear" - Clears (deletes) the selected text or object.

"SelectAll" - Selects the complete text

Use the keyword "PasteContent" to paste special content from another application into the Rich Text Edit Control. For example you can paste a rich text document to the Rich Text Edit Control which is linked to another application, like Microsoft Word. If the content is not compatible or not detected then it is pasted normal.

Example: teClipboard "Rectangle1" "SelectAll"

teClose

Purpose: Close the Text Control window and remove it from an existing NeoBook Rectangle

Category: Open/ Close

Syntax: teClose "Rectangle" "Options"

Specify the name of a NeoBook Rectangle. Enter the keyword "HideErrors" in options to Hide Errors while removing the Text

Control window. To display Error messages use the keyword "ShowErrors". Normally you would use "HideErrors" for the final publication and "ShowErrors" only for testing purposes.

Example: teClose "Rectangle1" "HideErrors"

teDeleteLine

Purpose: Delete a specified line of text
Category: Line Operations
Syntax: teDeleteLine "Rectangle" "LineNumber"

Specify the name of a NeoBook Rectangle and the *LineNumber* to delete.

Example: The example deletes line number three.

```
teDeleteLine "Rectangle1" "3"
```

teExportToPDF

Purpose: Export the current document to an Adobe ® Acrobat ® PDF File.
 (Images are not supported).
Category: Text Operations
Syntax: teExportToPDF "Rectangle" "EmbedFonts" "FontEncoding"
 "PageMargins" "PageSize" "PrintOrientation" "PDF_FileName"

Specify the name of a NeoBook *Rectangle*.

EmbedFonts: Specify if fonts are included into the pdf document.
Options are None, Subset, Full. Note that including fonts will increase the file size significantly. None is recommended.

FontEncoding: Apply different types of font encoding. The options are WinAnsi, PDFDoc, Standard, MacRoman, MacExpert. The WinAnsi Option is suitable for most languages but does currently not support all characters. PDFDoc seems to be suitable for the English language.

PageMargins: Specify the values for the Top, Left, Right and Bottom Margins in the measurement unit PIXEL. If you choose "Default" then the page size is determined by the global variables for [RectangleName.PageTop], [RectangleName.PageLeft], [RectangleName.PageRight]

and [RectangleName.PageBottom] or by the settings made by the PageSetupDialog.

PageSize: Specify the size of the Page selecting a value from the listbox. If you choose "Default" then the page size is determined by the global variables for [RectangleName.PageWidth] and [RectangleName.PageHeight] or by the settings made by the PageSetupDialog.

PrintOrientation: Choose between Landscape and Portrait.

PDF FileName: Name of the PDF file to create.

Example: The example exports the current document into a PDF File. It uses the default values to setup the page size (from global Variables or Page Setup Dialog). After the pdf-document was created an Acrobat Reader is called to display the file.

```
teExportToPDF "Rectangle1" "None" "WinAnsi"
"Default" "Default" "Portrait"
"[PubDir]test.pdf"
FileExists "[PubDir]test.pdf" "[Result]"
If "[Result]" "=" "True"
    Run "[PubDir]test.pdf" "" "" "" ""
else
    MsgBox "Error: Export to PDF File" "No
file was exported into PDF Format!"
endif
teSetFocus "teRectangle"
```

teFindText

Purpose: Search for Text in the document.

Category: Text Operations

Syntax: teFindText "Rectangle" "Found-Variable" "FindString" "Start"
"Highlight" "MatchCase" "WholeWord"

"Rectangle": Name of a NeoBook Rectangle.

"Found-Variable": Returns the index of the position where the text searched for is found. If the specified string is not found then the Found-Variable returns -1.

"FindString": String to search for.

"Start": Index number that determines where to begin the search.

"Highlight": If Highlight is entered then a match appears highlighted (selected) in the text. You can use teSelText to

change the selected text. Leave Highlight empty ("") if the match shall not be selected.

"MatchCase": If MatchCase is entered then the string to search for must exactly match the text in the document. If you search for >NeoBook< then >Neobook< would not be found. Leave MatchCase empty ("") to ignore an exact match. In this case the search string >NeoBook< would find >Neobook< in the document.

"WholeWord": If WholeWord is entered then the match must be a single word. If you would search for the word >Book< then >Neobook< would not be found. Leave WholeWord empty to ignore the word case. In this case >Book< would find >Neobook< in the document.

You will find code samples for a "find dialog" and a "search and replace dialog" in the sample text editor. Look for the action commands in the find button or replace button. You can use the teWriteMode command to lock refreshing the screen when text was modified. This is important if you want to search and replace NeoBook Variables in the Text. Tip: Place all NeoBook Variables in a Text Entry Field. This causes a text change event to fire each time a variable is modified. Place code in the text change event to modify variables in the text. Somehow you must find the current value of your variables in the editor document.

Example: The example searches for the text "NeoBook". The found position will be highlighted. The search options MatchCase and WholeWord will be ignored.

```
teFindText "Rectangle1" "[teFindText]" "NeoBook"
"1" "Highlight" "" ""
```

teFirstIndent

Purpose: Get or set the indent of the first line of a paragraph at the current input position. The measurement unit is pixel.

Category: Paragraph Formatting

Syntax: teLeftIndent "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. If you want to get the value for the first line indent you must the set variable value to -1. The function returns the pixel value for the first line indent. Specify a positive value (or zero) for the variable to set the value for the first line.

The function returns the internal Win32 Rich Text Edit Control

values. These internal windows values for LeftIndent and FirstLineIndent are a little bit confusing. The paragraph dialog of Editors Toolbox recalculates the values for the regular usage of a paragraph formatting dialog. The relation is shown in this example:

$\text{MyFirstLineIndent} = \text{LeftIndent} + \text{FirstLineIndent}$

$\text{MyLeftIndent} = - \text{FirstLineIndent}$

Note that screen pixels, font pixels and printer pixels are dependent on different values. Therefore word-processors calculate 72 dots per inch as a standard unit. You get fairly good results with this value on different hardware.

Example: The following example stores the value of the first line indent in the variable [teFirstLineIndent].

```
SetVar "[teFirstLineIndent]" "-1"  
teFirstLineIndent "Rectangle1"  
"[teFirstLineIndent]"
```

The example sets the value of the first line indent to 72 which is the default editor measurement value for the measurement of one inch.

```
SetVar "[teFirstLineIndent]" "72"  
teFirstLineIndent "Rectangle1"  
"[teFirstLineIndent]"
```

teFontDialog

Purpose: Display a Windows standard dialog to set the font style for the current input position.

Category: Character Formatting

Syntax: teFontDialog "Rectangle"

Specify the name of a NeoBook Rectangle.

Example: teFontDialog "Rectangle1"

teGetAlignment

Purpose: Get the alignment style of the current paragraph

Category: Paragraph Formatting

Syntax: teGetAlignment "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Use a *Variable* to get and store the keywords of the current alignment style. The variable returns the keywords Left, Center, Right.
(Unsupported Functionality: The function teGetAlignment assumes for Rich Edit 1.0 that a paragraph is justified if it is not left, centered or right aligned. For Rich Edit 2.0 Controls a justified paragraph alignment is detected. In both case the Variable returns the keyword "Justify". See also teSetAlignment.

Example: The example checks if the alignment is centered and changes it to left alignment.

```
teGetAlignment "Rectangle1" "[teGetAlignment]"
If "[teGetAlignment]" "=" "Center"
    teSetAlignment "Rectangle1" "Left"
Endif
```

teGetFontList

Purpose: Fill an existing NeoBook Listbox or Combobox with fontnames of the current computer system.

Category: Character Formatting

Syntax: teGetFontList "Rectangle" "NeoBookListbox"

Specify the name of a NeoBook Rectangle. Enter the object name of a NeoBook Listbox or Combobox. The Listbox will be filled with the currently installed fonts. The function is basically intended for the format toolbar.

Example: teGetFontList "Rectangle1" "ComboBox1"

teGetFontName

Purpose: Get the name of the font at the current input position.

Category: Character Formatting

Syntax: teGetFontName "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Specify a variable to return the name of the font at the current input position.

Example: The example checks for the font "Times New Roman" and changes it to "Arial".

```
teGetFontName "Rectangle1" "[teGetFontName]"
If "[teGetFontName]" "=" "Times New Roman"
    teSetFontName "Rectangle1" "Arial"
Endif
```

teGetFontSize

Purpose: Get the font size of the character at current input position.

Category: Character Formatting

Syntax: teGetFontSize "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Specify a variable to return the size of the font at the current input position.

Example: The example checks for a large title font size and changes it to a size of 12 points.

```
teGetFontSize "Rectangle1" "[teGetFontSize]"
If "[teGetFontSize]" "=" "24"
    teSetFontSize "Rectangle1" "12"
Endif
```

teGetFontStyle

Purpose: Get the font style for current input position.

Category: Character Formatting

Syntax: teGetFontStyle "Rectangle" "FontStyle" "Variable"

Specify the name of a NeoBook Rectangle. Use the keywords Bold, Italic, Underline, Strikeout, Subscript, Superscript or Protected for the font style you want to check. The variable returns a keyword for the required fontstyle: Bold, Italic, Underline, Strikeout, Subscript, Superscript or Protected. The variable returns an empty string ("") if the input position is has no special format or if the text was selected. If you want to edit or delete protected text you must use the command teWriteMode with the option 'EnableEditProtected' to enable editing of protected text.

Example: The example checks if the character at the current input position is formatted bold.

```
teGetFontStyle "Rectangle1" "Bold"  
"[teGetFontStyle]"
```

teGetLine

Purpose: Get text from a specified line number
Category: Line Operations
Syntax: teGetLine "Rectangle" "Line" "Variable"

Specify the name of a NeoBook Rectangle. Specify a line number. The variable returns the text of the specified line.

Example: The example returns stores the text string of line number three in the variable [teGetLine].

```
teGetLine "Rectangle1" "3" "[teGetLine]"
```

teGetLineFromChar

Purpose: Get line index of current input position.
Category: Line Operations
Syntax: teGetLineFromChar "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Specify a variable to store the current input position.

Example: The example deletes the line at the current input position.

```
teGetLineFromChar "Rectangle1"  
"[GetLineFromChar]"  
teDeleteLine "Rectangle1" "[GetLineFromChar]"
```

teGetNumbering

Purpose: Get the paragraph numbering style for the paragraph at the current input position.
Category: Paragraph Formatting
Syntax: teGetBullets "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Use a variable to store the keyword for the current numbering style. The variable returns the following keywords:

“None”: No numbering style.

“Bullets”: Bullets as numbering style

“ArabicNumbers”: Arabic numbers (1,2,3,...)

“LoCaseLetter”: Lower case letters (a,b,c,...)

“UpCaseLetter”: Upper case letters (A,B,C,...)

The variable might return other unsupported values for a numbering style like *LoCaseRoman* or *UpCaseRoman*.

Rich Edit 1.0 (RichEd32.dll) only supports Bullets. Therefore all other numbering styles will be ignored. Rich Edit 2.0 stores a numbering style and returns a value but does not display it. Therefore paragraph numbering can only be used with Rich Edit 3.0 that displays and stores line spacing.

Example: The example checks if the numbering style is Arabic numbers and changes it to lower case letters. Set the indentation of the Letter numbering style to 20 pixels (points).

```
teGetNumbering "Rectangle1" "[teGetNumbering]"
If "[teGetNumbering]" "=" "ArabicNumbers"
    teSetNumbering "Rectangle1" "LoCaseLetter"
    SetVar "[teLeftIndent]" "20"
    teLeftIndent "Rectangle1" "[teLeftIndent]"
endif
```

teGetText

Purpose: Store the complete text of the current Rich Edit document in a NeoBook variable or get the word at the current input position. You can specify the output format like HTML, RTF source code or unformatted text.

Category: Text Operations

Syntax: teGetText "Rectangle" "Options" "Variable"

Specify the name of a NeoBook Rectangle.

Specify a *Variable* to store the current text of the document. Use a keyword as a value for *Variable* to specify the output format: Use the Keyword “Unformatted” to store the unformatted text paragraph by paragraph. All formatting and embedded objects will be ignored. This is the default output format.

Use the keyword “LineMode” in Variable to store text unformatted line by line with the same number of characters in one line as the current document in the editor.

(1) Use the keyword "RTF" for options to store the rich text format source code (rtf source code) in a variable. When you use the same variable in the Neobook Simple Text Tool you will see the formatted text. Use the rtf source code variable also to store formatted rich text in database records. Use teSetText together with the Variable to show formatted text.

Your basic database code for NeoBookDB might look like this:

```
. Attach Editors Toolbox textwindow to a Rectangle.
. Set the "Variable to store unformatted text" to the same
  Name as your Memo field in the Database.
    teOpen "Rectangle1" "[Database.cwTextMemoRTF]" "" "" ""
      "Border" "Scrollbars" "Inches" "" "Ruler"
    teSetFocus "Rectangle1"
. Create Database with Neosoftware's dbDatabase plug-in.
. You can define an OnFormatChange subroutine but this is
  not necessary.
    dbfCreate "[PubDir]Database.dbf"
"cwTitleStr,String,100|cwTextMemo,Memo,0|cwOptions,String,1
00" "Database_OnFormatChange"
. Define Alias
    dbfDefineAlias "[PubDir]Database.dbf" "Database"
. SetAutoEdit to automaticilly update the variables
    dbfSetAutoEdit "Database" "True"
. do the first update if information is in the Database
    teSetText "Rectangle1" "[Database.cwTextMemo]"
```

The action commands for your Database Add Button would look like this:

```
teGetText "Rectangle1" "RTF" "[teGetText]"
SetVar "[Database.cwTextMemo]" "[teGetText]"
dbfAddRecord "Database"
teSetText "Rectangle1" "[Database.cwTextMemo]"
```

The command for the Database First Button, Next Button or Previous button would look like this. Just use the "teSetText" command to place the formatted text from the memo-field of your database into the Editors Toolbox Textwindow:

```
dbfFirst "Database"
teSetText "Rectangle1" "[Database.cwTextMemo]"
```

(2) Use the keyword "HTML" for Options to store text in HTML format in a variable. The converter supports only a small subset of the RTF code which is (more or less) supported by Rich Edit 1.0 controls. Embedded objects and bitmap images are currently not supported. The command uses the Neobook global variable

[PageTitle] of the current page as the title of the HTML page or "Untitled" if [PageTitle] contains no value.

The following formatting is supported by the html converter: **bold**, underlined, ~~strikeout~~ and *italic* text, indentation of paragraph, text alignment (left, right, centered, justified), numbering style bullets, font size, font face and font color. All other rich text formatting information will be ignored or cause the function to return an empty variable. You can add HTML commands into the Rich Edit document which are preserved during conversion to HTML. If you store the HTML variable to a file you can display the website in a web browser. Please consult the HTML converter demo for more details.

If a text document is converted to HTML that contains links then the converter calls the *OnLinkClicked* subroutine (specified in the *teOpen* command) and stores in the *Source* variable the keyword "converter". You can use this feature to convert your links to HTML source code commands. The sample application converts linked text in rich text format to a link in HTML that can access NeoBook action commands in the WebBrowser Tool of NeoBook. Note that you can also parse the html code in the variable to insert special NeoBook action commands on keywords. It is easy to insert pictures, buttons or whatever to the converted HTML document if you know basics about the HTML programming language.

(3) Use the keyword "WordAtCursor" for Options to store the word at the current input position in a variable. The result string with the word at the cursor position must be cleaned from word limiters like: ?!"%&\$()/ etc. See the section "How to create a popup menu" for details. The result is an unformatted text. Example:

```
teSetFocus "teRectangle"  
teGetText "teRectangle" "WordAtCursor" "[Word]"  
AlertBox "GetText" "WordAtCursor: [Word]"
```

Example: Store the text paragraph by paragraph in the variable "teGetText".

```
teGetText "Rectangle1" "Unformatted" [teGetText]"
```

This example stores the text line by line in the variable "teGetText". Each line contains the same number of characters as the document in the editor.

```
teGetText "Rectangle1" "LineMode" "[teGetText]"
```

The following example stores the text in the format HTML in the variable "teGetText". The variable "teGetText" is set with the keyword "HTML". The FileWrite command saves the variable to a file in the current publication directory that can be displayed in the WebBrowser tool.

```
teGetText "Rectangle1" "HTML" "[teGetText]"  
FileWrite "[PubDir]FileName.htm" "All"  
"[teGetText]"
```

teGetTextLength

Purpose: Store the number of characters in the current rich text document to a variable.

Category: General Settings

Syntax: teGetTextLength "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Specify a variable that returns the number of characters in the text.

Example: teGetTextLength "Rectangle1" "[teGetTextLength]"

teGetTotalLines

Purpose: Get total number of lines in the text

Category: Line Operations

Syntax: teGetTotalLines "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Specify a variable that returns the total number of lines in the text.

Example: teGetTotalLines "Rectangle1" "[teGetTotalLines]"

teHideSelection

Purpose: Specifies whether a text selection is hidden when the Text Control loses the input focus.

Category: General Settings

Syntax: teHideSelection "Rectangle" "Options"

Specify the name of a NeoBook Rectangle. Enter "True" in options to hide the selection when the Text Control loses the input focus. Enter "False" if you want the selection to stay visible, independent of the input focus. "False" is the default value of the Text Control.

Example: teHideSelection "Rectangle1" "True"

teInsertImage

Purpose: Insert a Jpeg image file (picture) into the document at the current input position

Category: Text Operations

Syntax: teInsertImage "Rectangle" "PictureName"

Specify the name of a NeoBook Rectangle. A Jpeg images file with the name specified in "PictureName" will be loaded into the document at the current cursor position.

Example: The following example inserts a bitmap image file from the publication directory with the name "slide4.jpg" into the text document.

```
teInsertImage "Rectangle1" "[PubDir]slide4.jpg"
```

teInsertObjectDialog

Purpose: Call a Windows standard dialog to insert an object to the rich text document. Objects can be linked or embedded to the document.

Category: Integrated Dialogboxes

Syntax: teInsertObjectDialog "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Specify a variable to receive a notification if the object was successfully inserted. The plug-in returns "TRUE" if the object was inserted or "FALSE" if

not. Note that you can add a picture file in bitmap format with the command `teInsertImage`.

Example: The following example shows the load object dialog and inserts a selected object at the current input position. If the object could not be inserted then the variable `[teInsertObject]` returns false.

```
SetVar "[teInsertObject]" ""
teInsertObjectDialog "teRectangle"
"[teInsertObject]"
If "[teInsertObject]" "=" "False"
    MessageBox "Information" "Object could not be
    inserted!"
endif
ShowObject "teRectangle" "None" "0"
```

teInsertSpecial

Purpose: Insert special functions at the current input position. Currently the function only inserts a non visible page break for printing and export. This function requires a Rich Edit Control 2.0 or higher.

Category: Text Operations

Syntax: `teInsertSpecial "Rectangle" "Options"`

Specify the name of a NeoBook Rectangle. Use the keyword `"PageBreak"` to insert a non visible page break at the current input position. In a more sophisticated word processor like Microsoft ® Word you will see the page break. This option gives you the opportunity to insert a page break programmatically to help creating reports.

Example: The following example inserts a page break.

```
teInsertImage "Rectangle1" "PageBreak"
```

teLeftIndent

Purpose: Get or set left indent of paragraph at the current input position. The measurement unit is pixel.

Category: Paragraph Formatting

Syntax: `teLeftIndent "Rectangle" "Variable"`

Specify the name of a NeoBook Rectangle. If you want to get the value for left indent you must the set variable value to -1. The function returns the pixel value for the right indent. Specify a positive value (or zero) for the variable to set the right indent. The function returns the internal Win32 Rich Text Edit Control values. These internal windows values for LeftIndent and FirstLineIndent are a little bit confusing. The paragraph dialog of Editors Toolbox recalculates the values for the regular usage of a paragraph formatting dialog. The relation is shown in this example:

$\text{MyFirstLineIndent} = \text{LeftIndent} + \text{FirstLineIndent}$

$\text{MyLeftIndent} = - \text{FirstLineIndent}$

Note that screen pixels, font pixels and printer pixels are depending on different values. Therefore word-processors calculate 72 dots per inch as a standard unit. You get fairly good results with this value on different hardware.

Example: The following example stores the value of the right indent in the variable [teRightIndent].

```
SetVar "[teLeftIndent]" "-1"
teLeftIndent "Rectangle1" "[teLeftIndent]"
```

The example sets the value of the left indent to 72 which is the default editor measurement value for the measurement of one inch.

```
SetVar "[teLeftIndent]" "72"
teLeftIndent "Rectangle1" "[teLeftIndent]"
```

teLineSpacing

Purpose: Get or set the line spacing for the paragraph at the current input position. The measurement unit is pixel.

Category: Paragraph Formatting

Syntax: teLeftIndent "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. The function expects a variable to set or get the line spacing value. If you want to get the value you must the set the value for Variable to -1. The function returns the pixel value for the line spacing. Specify a positive value for Variable to set the line spacing. Note that there is a relation between font-size and line-spacing. The specified value is always the minimum value. If text with a big

font size is inserted in a paragraph with a small font size then the line spacing is automatically adjusted.

Line Spacing requires Rich Edit Versions 2.0 or higher.

Example: The following example stores the current font size in a variable and set the line spacing to 1.5 according to the fontsize.

```
teGetFontSize "teRectangle" "[teGetFontSize]"
Math "[teGetFontSize] * 1.5" "0"
"[teLineSpacing]"
teLineSpacing "teRectangle" "[teLineSpacing]"
```

teLink

Purpose: Changes selected text to hyperlink text.
Use this command to add hyperlinks between text in a document and NeoBook. When the mouse hovers over the linked text the mouse pointer changes to a “hand”.

Category: Text Operations
Syntax: teLink “Rectangle” “Options”

Specify the name of a NeoBook Rectangle. Use the keywords “Selection” to change selected text to a hyperlink or use the keyword “Clear” to clear the selection. Two hyperlinks next to another (on one or two lines) need to be separated with a space. Note that hyperlinks are achieved with a trick. The function assumes a hyperlink when it finds blue and underlined characters. You must provide a special dialog to edit hyperlinks. See the CreativeWriter sample application at subroutines “Rectanlge_OnLinkClicked” to learn how to edit and respond to hyperlinks. Note that when you use a subroutine as an event handler for links in your publication an internal function is called on load file operations (teLoadFileDialog) to change blue and underlined text to hyperlinks. If you load a large document this internal search operation takes a long time until the editor window receives the focus.

If you click on the link text in the document the link is passed to a Neobook subroutine with information like the position of the linked text in the document. The name of the subroutine to handle link events is specified in the *teOpen* command. The plug-in creates variables that contain this additional information:

“SubroutineName.Start”: Start position of linked text in the document
“SubroutineName.End”: End position of linked text in the document
“SubroutineName.Text”: Linked text
“SubroutineName.Source”: Information about the caller of the link. If it is the HTML converter that found a link on conversion the variable returns the keyword *“Converter”*. If the mouse button was clicked on linked text the variable contains *“0”* for left mouse button, *“1”* for right mouse button and *“2”* for middle mouse button.

You can use the hyperlink function to run NeoBook action commands on a mouse click in your document. Use the search and replace function to jump to another position in the text. See the Creative Writer sample application and the start up document (click on first line) to learn how to use linked text and respond to a link event in a subroutine. The hyperlink function works for all Rich Edit Versions (1, 2 and 3). If the document is converted to HTML the converter will call the subroutine on each linked text found in the document. In this case the variable *“SubroutineName.Source”* contains the keyword *“Converter”*.

teLoadFileDialog

Purpose: Load a textfile and display an (optional) dialogbox.
Category: Integrated Dialogboxes
Syntax: teLoadFileDialog “Rectangle” “FileName”

If FileName is empty (“”) then a Windows standard file-open dialog is shown where the user can select a textfile in rich text format. If a FileName is specified, the text file will be loaded without dialog. If the user selects a file in the dialog, the global variable *“RectangleName.FileName”* contains the name of the selected file.

Use the Neobook global variable [CurrentDir] to specify the current directory before using the command teLoadFileDialog. Note that the command sets the global variable *“RectangleName.CurrentDir”* to the currently selected folder when using a dialog box. You can use this variable to set the NeoBook global [CurrentDir] variable (see above).

Using the command `tePlainText` with the option “PlainText” or “RichTextFormat” will force the plugin to load a file as unformatted or formatted rich text.

Example: The example shows a load file dialog and loads the text file if the user clicks ok.

```
teLoadFileDialog "Rectangle1" ""
```

The following example loads “MyTextFile.RTF” without showing a dialog. Use the global NeoBook variable `[LastError]` to check if the file was loaded. The plug-in sets this variable with an error message if the file was not loaded successful.

```
teLoadFileDialog "Rectangle1" "MyTextFile.RTF"
If "[LastError]" ">" ""
    MsgBox "Error" "Textfile not
    loaded!|[LastError]"
EndIf
```

teModified

Purpose: The plug-in internally keeps track if the user makes changes to the current document. Use `teModified` to get or set the current state of the modified variable.

Category: General Settings

Syntax: `teModified "Rectangle" "Variable"`

Specify the name of a NeoBook Rectangle. An empty value ("") for *Variable* gets the current state of the modified flag. The function returns the keywords “TRUE” if the text was changed or “FALSE” if the text was not changed.

The keyword “TRUE” sets the modified flag to true and the keyword “FALSE” for variable sets the flag to false. The functions `teLoadFileDialog`, `teSaveFileDialog` and `teClipboard` with the keyword “clear” will set the modified flag to false.

Example: The example assumes that the user wants to close the application. If the text was changed then a “Save File” message shall be displayed.

```
. Check if text was changed
SetVar "[teModified]" ""
teModified "teRectangle" "[teModified]"
```

```

. If changed, then show a "save file" message
If "[teModified]" "=" "TRUE"
    MsgBox "Exit Program" "Text was
    modified. Save File?" "Yes|No"
    "[teValue]"
    If "[teValue]" "=" "1"
        teSaveFileDialog "teRectangle" ""
        teRedraw "teRectangle"
    EndIf
Endif
. Close Rich Edit Object
teClose "teRectangle" "HideErrors"
. Close application
Exit "" ""

```

teOpen

Purpose: Attach a Rich Text Edit Control to an existing NeoBook Rectangle. Editors Toolbox requires NeoBook 4.1.3a or higher. You can add more than one Rich Text Edit Control to a publication page.

Category: General Settings

Syntax: teOpen "Rectangle" "Variable" "Subroutine OnFormat" "Border" "Reserved" "BackgroundColor" "TextColor" "MeasurementUnits" "Subroutine OnLink" "Ruler"

"Rectangle": Specify the name of a NeoBook Rectangle.

"Variable": Specify an (optional) variable name. The plug-in will update the variable whenever the text is changed. Note that you can not use a *"variable"* when you want to process large text files. This slows down the system performance. In this case you must delete a value in the text entry field *"variable to store unformatted text"*.

"Subroutine OnFormat": You can specify an optional subroutine from your publications subroutine Action. The subroutine specified here will automatically execute whenever the text format is changed. Use this format change event to set-up the format toolbar. Subroutines are entered from the Actions page of Nook's Book Properties screen.

If you want to handle events on text change you must draw a Text Entry Field from NeoBooks Toolbar and use the same variable name as specified in teOpen. You can place the Text Entry Field on a hidden publication page. Then you can respond to the *"textchange"* event of the Text Entry Field. Please note

that NeoBook does not support large numbers of action commands to be executed on each keystroke.

The OnFormat event is fired on all formatting styles on the format bar of the sample application and on a change of the text color.

“Border”: Use the keyword “Border” to draw a border. Leave the parameters “Variable”, “Subroutine” and “Border” empty to ignore these functions.

“Scrollbars”: Use the keyword “Scrollbars” to attach scrollbars to the Rich Edit Window.

“BackgroundColor”: You can specify a color for background of the window. See `teTextColor` for more details. Please note that Editors Toolbox Standard Edition does not support different colors for WindowBackground and the Background of the current text line.

“TextColor”: Specify the color of the text. See `teTextColor` for more details.

“MeasurementUnits”: Specify a measurement unit which is used for the paragraph formatting dialog and for the page setup dialog.

“SubroutineOnLink”: You can specify a subroutine that is called if the document contains special Hyperlinks to NeoBook and the user clicks with the mouse on this links. Use the `teLink` command to change selected text in the document to a hyperlink to NeoBook. See `teLink` for more information. See also the CreativeWriter sample application at subroutines

“Rectangle_OnLinkClicked” to learn how to edit and respond to hyperlinks. Note that when you use a subroutine as an event handler for links in your publication an internal function is called on load file operations (`teLoadFileDialog`) to change all blue and underlined text to Hyperlinks to NeoBook. If you load a large document this internal search operation takes a long time until the editor window receives the focus.

“Ruler”: Show a ruler to set paragraph margins, tab stops and page margins together with mouse dragging. You can change the left and right page margins with the mouse on the ruler. This values are used for printing, page setup dialog or export to PDF. To Hide the Ruler leave the option empty (“”). Page margins are not saved within the document. (See global variables for details). The ruler shows the measurement “mm” or “inch” depending on the settings for “MeasurementUnits”. The measurement values shown on the ruler are only approximately values.

Use the following global variables to setup the ruler before calling the `teOpen` command:

- **“RectangleName.Ruler.Flat”**: Use a flat ruler (true/false)

- “*RectangleName.Ruler.RightMargin*”: Set a value for the ruler’s right margin. If you set this value for right margin the value is ignored for printing or page setup dialog. It is visual on screen only. Specify the value in metric millimeters.
- “*RectangleName.Ruler.LeftMargin*”: Set a value for the ruler’s left margin. If you set this value for left margin the value is ignored for printing or page setup dialog. It is visual on screen only. Specify the value in metric millimeters.
- You find more variables to setup the ruler in the global variables section in this UserGuide.

Example: The following example draws a standard Rich Text Edit Control on a NeoBook rectangle. The function uses the default color settings and specifies Inches as measurement unit. Note that naming a “OnLinkClick” subroutine enables hyperlinks. The hyperlink function is intended for small documents. If you want to use large documents you should leave “OnLinkClicked” empty.

```
teOpen "Rectangle1" "[Rectangle1_Text]"
"Rectangle1_OnFormatChange" "" "" "Border" ""
"Inches" "Rectangle1_OnLinkClicked" "Ruler"
```

tePageSetupDialog

Purpose: Call a Windows standard dialog to set-up the page settings for printing. The settings will not be stored inside the text file when the document is saved. THIS DIALOG IS CURRENTLY NOT INTENDED TO BE USED IN FINAL PUBLICATIONS.

Category: Integrated Dialogboxes

Syntax: tePageSetupDialog “Rectangle”

Specify the name of a NeoBook Rectangle. All values for margins, paper size and paper orientation will be used to calculate the resulting number of pages for the document in the print dialog (see tePrintDialog). Note, that the text width in edit mode is set to the width of the (rectangle) window. For printing the paper format of the printer is used to set-up the width of the text. The measurement units of the dialog are set with the teOpen action command.

Example: tePageSetupDialog "Rectangle1"

teParagraphDialog

Purpose: Display a dialogbox to setup the paragraph formatting style
Category: Integrated Dialogboxes
Syntax: teParagraphDialog "Rectangle" "Language" "MeasurementUnitis"

Specify the name of a NeoBook Rectangle. You can specify the keyword "English" for English language or translate the English language string to your native language. Each word must be separated by a semicolon. For more details see the action command dialog. Enter the keywords "Inches", "Millimeters" or "Pixel" to set the dialog values to the desired measurement units. Use the keyword "Default" to use the measurement units specified in the teOpen action command.

The function teParagraphDialog checks which Rich Text Edit Control is installed on the computer system and displays only valid commands for this Rich Edit.

Rich Edit 1.0 - Indentation, Alignment (Left, Center, Right)

Rich Edit 2.0 - Adds line spacing

Rich Edit 3.0 - Adds numbering.

Information: If you compare the Pixel values in the Dialog to the values of teFirstLineIndent or teLeftIndent you will notice that teParagraphDialog recalculates the internal indent value of the Windows Rich Text Edit Control to standard values of a paragraph dialogs in word-processors.

Example: teParagraphDialog "Rectangle1" "English"
"Millimeters"

tePlainText

Purpose: Ignore the internal text format when loading or saving a text file and interpret all formatting information as normal text or richt text.

Category: Generall Settings
Syntax: tePlainText "Rectangle" "Options"

Specify the name of a NeoBook Rectangle. Enter the keyword "PlainText" in options to ignore the internal text format on loading and saving. Enter "RichTextFormat" or leave options

empty to force the normal rich text mode. Note that you cannot perform any formatting on plain text. This command forces the specified text format independent from the file extension (“rtf” or “txt”) on loading and saving. You can also enable formatting of a plain text with this command.

Example: tePlainText "Rectangle1" "PlainText"

tePrintDialog

Purpose: Print current document and display an (optional) dialogbox.

Category: Integrated Dialogboxes

Syntax: tePrintDialog “Rectangle” “Dialog” “Options”

Specify the name of a NeoBook Rectangle. Enter the keyword “ShowDialog” for *Dialog* to show the Windows standard print dialog before printing or specify none.

According to the settings for Page Setup (see tePageSetupDialog) the resulting number of pages are calculated and displayed in the dialog. The user can select the number of pages that will be printed. You can also specify the paper margins for the Left, Top, Right and Bottom of the page with global variables. The left and right page margins can also be set with the ruler. The function expects all values in the measurement unit Millimeter. Use the action command dialog to set the margins. Separate each value with a semicolon. Use the keyword “*Default*” to accept the default settings as specified in the teOpen command.

You can also use the NeoBook print function to print a file in rich text format and add headers and footers. See the CreativeWriter sample.

Example: tePrintDialog "Rectangle1" "ShowDialog" "Default"

tePrinterSetupDialog

Purpose: Display a Windows standard dialog box for printer set-up

Category: Integrated Dialogboxes

Syntax: tePrinterSetup “Rectangle”

Specify the name of a NeoBook Rectangle.

Example: tePrinterSetup "Rectangle1"

teReAlign

Purpose: Aligns a Rich Text Edit Control to the Rectangle. Use this command after the Rectangle was resized with the NeoBook SizeObject command in Neobook versions prior to 4.1.1d. teReAlign does not work with CustomWindows. If you use NeoBook version 4.1.1d or higher the Rich Text Edit Control is resized automatically when the attached rectangle is resized. You don't need to call teReAlign. (See also global variable [Rectangle.AllowResize] to disable automatic resizing).

Category: General Settings

Syntax: teReAlign "Rectangle"

Specify the name of a NeoBook Rectangle.

Example: The example resizes the Neobook rectangle and the Rich Text Edit Control.

```
SizeObject "Rectangle1" "350" "225"  
teReAlign "Rectangle1"
```

teRedraw

Purpose: Redraw the contents of the Text Control.

Category: General Settings

Syntax: teRedraw "Rectangle"

Specify the name of a NeoBook Rectangle. Neobook provides a rectangle as a host for visual content for a plug-in. The rectangle does not know which content is attached. Use teRedraw after displaying standard windows dialogs on top of the Rich Text Edit Control.

Running a word-processor on the back of a NeoBook rectangle may cause in certain situations redraw problems. Therefore you must use the teRedraw command.

Example: teRedraw "Rectangle1"

teRichEditVersion

Purpose: Store the active Rich Text Edit Control version in a variable. The Microsoft ® Rich Text Edit Control 1.0 has a limited functionality and does not support line spacing and numbered paragraphs. Use this function to determine which Rich Text Edit Control is running.

Category: General Settings

Syntax: teRichEditVersion "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. The Variable returns

1 for Rich Text Edit Control 1.0
2 for Rich Text Edit Control 2.0
3 for Rich Text Edit Control 3.0
4 for Rich Text Edit Control 4.1

Example: This example checks if Rich Text Edit Control 2.0 is installed and set the line spacing according to the font size to 1.5 spacing.

```
teRichEditControl "Rectangle1"  
"[teRichEditVersion]"  
If "[teRichEditVersion]" "<>" "1"  
    teGetFontSize "teRectangle" "[teGetFontSize]"  
    Math "[teGetFontSize] * 1.5" "0"  
    "[teLineSpacing]"  
    teLineSpacing "teRectangle" "[teLineSpacing]"  
Endif
```

teRightIndent

Purpose: Get or set the right indentation for the paragraph at the current input position. The measurement unit is pixel.

Category: Paragraph Formatting

Syntax: teRightIndent "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. If you want to get the value for right indent you must the set variable value to -1. The function returns the pixel value for the right indent. Specify a positive value (or zero) for the variable to set the right indent.

Example: The following example stores the value of the right indent in the variable [teRightIndent].

```
SetVar "[teRightIndent]" "-1"
teRightIndent "Rectangle1" "[teRightIndent]"
```

The example sets the value of the right indent to 72 which is the default editor measurement value for one inch.

```
SetVar "[teRightIndent]" "72"
teRightIndent "Rectangle1" "[teRightIndent]"
```

teSaveFileDialog

Purpose: Save a textfile and display an (optional) dialogbox.
Category: Integrated Dialogboxes
Syntax: teSaveFileDialog "Rectangle" "FileName"

Specify the name of a NeoBook Rectangle. If FileName is empty ("") then a standard Windows save file dialog is shown where the user can select a text file in rich text format. If you specify a string for FileName then the text file with this file name will be saved without dialog. If the user selects a file in the dialog box the function returns the filename in the variable:

"RectangleName.FileName". The teModified flag is set to false.

Use the Neobook global variable [CurrentDir] to specify the current directory before using the command teSaveFileDialog.

Note that the command sets the global variable

"RectangleName.CurrentDir" to the currently selected folder when using a dialog box. You can use this variable to set the NeoBook global [CurrentDir] variable (see above).

You can use the *tePlainText* command with the options "PlainText" or "RichTextFormat" to force saving the file as plain text or rich text.

Example: The example shows a save file dialog and saves the textfile if the user clicks ok.

```
teSaveFileDialog "Rectangle1" ""
```

The following example saves "MyTextFile.RTF" without showing a dialog. Use the global NeoBook variable [LastError] to check if an error occurred. If the user selects a file in the dialog, the global variable "RectangleName.FileName" contains the name of the selected file.

```
teSaveFileDialog "Rectangle1" "MyTextFile.RTF"
If "[LastError]" ">" ""
```

```
        MsgBox "Error" "Textfile not  
        saved! |[LastError]"  
    EndIf
```

teSelLength

Purpose: Set or get the number of characters selected in the document
Category: Text Operations
Syntax: teSelLength "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Specify a variable value of -1 to get the length of selected text. Insert a value to set the length for the current selection. Use teSelStart to specify the start position for the selected text. In conjunction with teClipboard the functions teSelStart, teSelLength and teSelText are useful for cut, copy, paste and clear operations.

Example: The example gets the length of a text selection and stores the value in the variable [teSelLength]

```
SetVar "[teSelLength]" "-1"  
teSelLength "Rectangle1" "[teSelLength]"
```

The following example starts a text selection at position 143 and selects 12 Characters. teSelText deletes the selection and inserts "Hello World".

```
SetVar "[teSelLength]" "143"  
teSelStart "Rectangle1" "[teSelLength]"  
SetVar "[teSelLength]" "12"  
teSelLength "Rectangle1" "[teSelLength]"  
teSelText "Rectangle1" "Hello World"
```

teSelStart

Purpose: Set or get start position of selected text or get the current input position
Category: Text Operations
Syntax: teSelStart "Rectangle" "Variable"

Specify the name of a NeoBook Rectangle. Specify a variable value of -1 to get the start of selected text or the current input position. Insert a value to set the current input position or the

start for selected text. Use `teSelLength` to specify the length for the selected text. In conjunction with `teClipboard` the functions `teSelStart`, `teSelLength` and `teSelText` are useful for cut, copy, paste and clear operations.

Example: The example gets the current input position in the text. The result is stored in the variable `[teSelStart]`.

```
SetVar "[teSelStart]" "-1"  
teSelStart "Rectangle1" "[teSelStart]"
```

The following example sets the cursor to position 348.

```
SetVar "[teSelStart]" "348"  
teSelStart "Rectangle1" "[teSelStart]"
```

teSelText

Purpose: Set or get selected text
Category: Text Operations
Syntax: `teSelText "Rectangle" "Variable"`

Set the name of a NeoBook *Rectangle*. Specify an empty string value ("") to store the currently selected text in *Variable*. If no Text is selected the variable returns an empty string. If you specify a string for *Variable* it will be inserted at the `SelStart` position. If text is selected the specified string will replace the selected text. In conjunction with `teClipboard` the functions `teSelStart`, `teSelLength` and `teSelText` are useful for cut, copy, paste and clear operations.

Example: The first example inserts the text string "Hello World" at the current input position in the text. You can change the current input position with `teSelStart`.

```
teSelText "Rectangle1" "Hello World"
```

The second example copies the current selected text to the variable `[teSelText]`, displays the content in an alert box and deletes the selected text in the document. The `"teClipboard"` command with the keyword `"clear"` deletes the currently selected text.

```
SetVar "[teSelText]" ""
```

```
teSetText "Rectangle1" "[teSetText]"
Alertbox "teSetText Sample" "[teSetText]"
teClipboard "Rectangle1" "Clear"
```

teSetAlignment

Purpose: Set paragraph alignment (Left, Center, Right)
Category: Paragraph Formatting
Syntax: teSetAlignment "Rectangle" "Alignment"

Specify the name of a NeoBook Rectangle. Use the keywords Left, Center, Right.

(Unsupported Functionality: The standard Win32 Rich Text Edit Controls Version 1.0 and 2.0 do not support a justified paragraph alignment. The Version 2.0 supports to format (set and get) a justified paragraph but does not show it in the edit window. With a little programming trick this alignment style can be shown when a justified paragraph is set at runtime for Version 1.0 and 2.0. Please note that a justified paragraph is not stored in Version 1.0 when the file is saved. Use the keyword "Justify" to set a justified alignment. Justified paragraphs are not recommended for final publications because in some cases a mouse click on the text document does disable keyboard input. The User has to click a second time.)

Example: teSetAlignment "Rectangle1" "Center"

teSetFocus

Purpose: Set the keyboard input focus to the Text Control
Category: General Settings
Syntax: teSetFocus "Rectangle"

Specify the name of a NeoBook Rectangle.

Example: teSetFocus "Rectangle1"

teSetFontName

Purpose: Set the fontname for current input position or selection
Category: Character Formatting
Syntax: teSetFontName "Rectangle" "FontName"

Specify the name of a NeoBook Rectangle. Specify the name of the font. Take care that the specified font exists on the system when the compiled NeoBook publication is running. The font file associated with the specified font name is not automatically included in the compiled publication by NeoBook.

Example: This example opens a text window, sets a font and shows some text. Place this sample code in the page properties section of NeoBook and run the publication.

```
teOpen "Rectangle1" "" "" "" "" "" "" "Inches" ""  
""  
teSetFontName "Rectangle1" "Comic Sans MS"  
teAddLine "Rectangle1" "-1" "Hello World!"
```

teSetFontSize

Purpose: Set fontsize for current input position or selection
Category: Character Formatting
Syntax: teSetFontSize "Rectangle" "FontSize"

Specify the name of a NeoBook Rectangle. Specify the size for the current font. The default size is 12 points.

Example: teSetFontSize "Rectangle1" "14"

teSetFontStyle

Purpose: Set font style for current input position or text selection
Category: Character Formatting
Syntax: teSetFontStyle "Rectangle" "FontStyle"

Specify the name of a NeoBook Rectangle. Use the keywords "Bold", "Italic", "Underline", "Strikeout", "Subscript", "Superscript" and "Protected" for *FontStyle*. Subscript and Superscript use the half font size for a positive or negative Offset of the line at the current input position or for selected text. You can specify a positive or negative numbered value for a line offset alternatively. The keyword "None" for *FontStyle* sets the style to unformatted characters.

It is recommended that you set the *FontStyle* to "None" before changing the current settings.

Example: This example changes the FontStyle to “bold”.

```
teSetFontStyle "Rectangle1" "None"  
teSetFontStyle "Rectangle1" "Bold"
```

This example formats subscript characters (negative line offset).

```
teSetFontStyle "Rectangle1" "Subscript"
```

The following examples are a little bit more complex and show how to select the complete text in order to change the font settings. The “BasicEditor” sample publication shows the same function but with only a few commands. This code demonstrates how to select text manually.

```
. Get the current text length  
teGetTextLength "Rectangle" "[teGetTextLength]"  
. Set cursor position to end of text  
SetVar "[teSelStart]" "[teGetTextLength]"  
teSelStart "Rectangle" "[teSelStart]"  
. Setup font style. Functions like on and off.  
teSetFontStyle "Rectangle" "Underline"  
teSetFontStyle "Rectangle" "Bold"  
. Setup font size.  
teSetFontSize "Rectangle" "14"  
. Setup font name  
teSetFontName "Rectangle" "Times New Roman"  
. Insert text  
SetVar "[teSelText]" "... I am formatted text."  
teSelText "Rectangle" "[teSelText]"  
. Unselect text  
SetVar "[teSelLength]" "0"  
teSelLength "Rectangle" "[teSelLength]"  
. Set new Cursor position  
StrLen "[teSelText]" "[teTextLen]"  
SetVar "[teTextLen]" "[teSelStart]+[teTextLen]"  
teSelStart "Rectangle" "[teTextLen]"
```

teSetNumbering

Purpose: Set numbering style for the paragraph at the current input position

Category: Paragraph Formatting

Syntax: `teSetBullets "Rectangle" "Options"`

Specify the name of a NeoBook Rectangle. Use the following keyword for Options:

“None”: Set no numbering style.

“Bullets”: Set bullets as numbering style

“SwitchBullets”: Switch between bullets or no numbering style

“ArabicNumbers”: Set numbering to Arabic numbers (0,1,2,..)

“LoCaseLetter”: Set lower case letters (a,b,c,...)

“UpCaseLetter”: Set upper case letters (A,B,C,...)

Rich Edit 1.0 (RichEd32.dll) does only support Bullets. Therefore all other numbering styles will be ignored. Rich Edit 2.0 stores the numbering style but does not display it. Therefore paragraph numbering can only be used with Rich Edit 3.0 that displays and stores line spacing.

Example: The first example switches between “Bullet” and “None”.

```
teSetBullets "Rectangle1" "SwitchBullets"
```

The second example set Arabic numbers.

```
teSetBullets "Rectangle1" "ArabicNumbers"
```

teSetText

Purpose: Write (and replace existing) text into the current rich text document. You can use Neobook variables.

Category: Text Operations

Syntax: `teSetText "Rectangle" "Text"`

Specify the name of a NeoBook Rectangle. The text of the current document is set to the string in *Text*.

Example: `teSetText "Rectangle1" "Text"`

teSpellCheck

Purpose: Activate live spell checking and show red wiggly lines under misspelled words. A right mouse click on a misspelled word shows a popup menu with a wordlist for correction and more options

like setup and activation of dictionaries. Alternatively call a spell checking dialog box.

Categorie: Spell Checking

Syntax: teSpellCheck "Rectangle" "MenuLanguage" "Options"

Specify the name of a NeoBook Rectangle. The variable "MenuLanguage" expects one of the following keywords to setup the language: Afrikaans, American, Aussie, Brazilian Portuguese, British, Czech, Danish, Dutch, Estonian, Finnish, French, German, Hungarian, Italian, Norwegian, Polish, Portuguese, Russian, Spanish and Swedish.

The variable Options allows to switch spell checking on and off. Use the keywords "SpellCheckON" and "SpellCheckOFF".

"RectangleName.SpellCheckHWND": You can specify the window handle of a Microsoft Rich Edit Control to be spell checked before calling teSpellCheck. In this case teSpellCheck checks the window handle (Rich Edit Control) instead of the control specified by the NeoBook rectangle. This function is intended for advanced NeoBook developers and may have unpredicted results. Note that you are not allowed to create your own spell check engine and violate the Addictive Software license agreement.

The following example spell checks the Editors Toolbox Rich Edit Control specified by its window handle:

```
SetVar "[HWND]" "[teRectangle.WinHandle]"
SetVar "[teRectangle.SpellCheckHWND]" "[HWND]"
teSpellCheck "teRectangle" "English" "DialogSpellCheck"
```

Editors Toolbox uses the high quality spell check engine from Addictive Software ®. The user can make all important choices using the popup menu on right mouse click.

The spell check starts at the current input position or is limited to selected text.

Editors Toolbox comes only with the basic English dictionary to keep the setup file small to enable download at low speed. Many dictionaries in many languages are available from Addictive Software ®. Currently supported: Afrikaans, American, Aussie, Brazilian Portuguese, British, Czech, Danish, Dutch, Finnish, French, German, Hungarian, Italian, Norwegian, Polish, Portuguese, Russian, Spanish, Swedish as well as Technical, Medical, Legal, Bible, Latin, etc. Most of these dictionaries are free. You will find the download website from Addictive Software ® at: <http://www.addictivesoftware.com/dicts-extern.htm>

You can also use Microsoft Word (R) custom dictionaries. Note that you can purchase a word list compiler from Addictive Software ® to built your own dictionaries.

On startup the spell checking engine creates two files for a custom dictionary “user.adu” and “user_sp.adl”. The configuration file “spell.cfg” is created when the user opens the popup menu and makes a choice. The default dictionary language is setup in the configuration file which is a simple text file. When the user makes changes in the spellcheck popup menu which shows up on a right mouse click on a misspelled word than the changes made are saved in the config file. You can easily changes the values of this file using the NeoBook “ReadLine” and “WriteLine” commands. The CreativeWriter sample application has a menu option to shows the config file in the Help menu. The spell checking engine supports different dictionaries (*.adm) to be active at the same time.

Calling the command teSpellCheck gives the option to call a setup dialog. You can search for language dictionaries and setup a language. The dialog saves a configuration file (spell.cfg) and saves it in the publication directory. Finally this file must be placed in compiled publication directory together with the dictionary files. If you don’t find it or if you see all words with red wiggly lines you must perform a search for this file and place it in the publication dictionary.

It seems that Neobook changes the value of the [PubDir] variable when you compile your publication. In this case your Neobook publication file can be in a different directory as the compiled publication.

If you want to go shure right click on a misspelled word in the editor window and call the option dialog. This should saves the configuration file in the correct directory.

Please note that Addictive Software can not give support to Neobook User as they are “customers” of a customer.

Example: This example adds spellchecking to an existing Neobook Rectangle. The popup menu (on right mouse clicks) is in the English language. (In some very rare situations teSpellCheck might need a redraw to display wiggly lines).

```
teSpellCheck "teRectangle" "English"  
"SpellCheckON"
```

The second example calls a spell checking dialog box. It assumes that live spell checking was active. The language popup menu language is English.

```
. Close Live Spell Checking
teSpellCheck "teRectangle" "English"
"LiveSpellCheckOFF"
. call dialog box
teSpellCheck "teRectangle" "English"
"DialogSpellCheck"
. Hide selected text after the dialog was closed
SetVar "[teSelLength]" "0"
teSelLength "teRectangle" "[teSelLength]"
. Set Focus to text window
teSetFocus "teRectangle"
```

If you use Neobook 5.0.1 or higher you can use the global variable “[SystemLanguageExt]” to determine which language is active on the windows system and call the dialog with the correct language settings. See the CreativeWriter sample application for Neobook 5.5 or higher for details.

teSpellOptionDialog

Purpose: Call an option dialog to set up spelling options.
Category: Spell Checking
Syntax: teSpellOptionDialog “Rectangle”

Specify the name of a NeoBook Rectangle.

The spell option dialog creates a spell checking configuration file (spell.cfg). This file must be in the same directory as the publication and the language dictionaries.

Example: teSpellOptionDialog "Rectangle1"

teSpellSuggest

Purpose: Get suggestions for a misspelled word or check if a word is correctly spelled.
Category: Spell Checking
Syntax: teSpellSuggest “Rectangle” “CheckThisWord” “SuggestWords”

Specify the name of a NeoBook Rectangle. Specify a misspelled word. Use the teGetText command together with the option "WordAtCursor" to get the word at the current input position.

The command teSpellSuggest stores in SuggestWords:

Empty String	- no suggestions for misspelled word available
-1	- Word is spelled ok (no mistakes found)
List of Words	- suggestions for misspelled word (words are separated with the pipe character " ")

Example: See the section how to create a popup menu for more details.

The following example checks the word "currently" and stores a list with suggestions in the variable [WordList] as a string.

```
teSpellSuggest "RectangleName" "curently" "[WordList]"
```

teTextColor

Purpose: Set the text color

Category: General Settings

Syntax: teTextColor "Rectangle" "Color" "Variable"

Specify the name of a NeoBook Rectangle. Enter a color with RGB values for Red, Green and Blue separated by a comma. Use the action command dialog for teTextColor to specify the color with the select color dialog. If the value for Color is -1 then the variable returns the RGB color value for the text at the current input position.

Example: This example gets the text color at the current input position and stores the result in the variable [teTextColor]. If the text color is black (0,0,0) then the text color is changed to blue (0,0,255).

```
teTextColor "Rectangle1" "-1" "[teTextColor]"
If "[teTextColor]" "=" "0,0,0"
    teTextColor "Rectangle1" "0,0,255"
"[teTextColor]"
Endif
```

teUndo

Purpose: Undo or redo the last text operations

Category: General Settings

Syntax: teUndo "Rectangle" "Options"

Specify the name of a NeoBook Rectangle. Use the keyword "Undo" to undo the last operation or "Redo" to redo the last operation. Use "Clear" to empty the undo/redo buffer. Rich Edit 2.0 (RichEd20.dll) supports multiple undo/ redo and the undo/ redo buffer stores up to 100 operations.

Example: teUndo "Rectangle1" "Undo"

teWordWrap

Purpose: Specify the word wrap style. Wrap words at the window border or handle text as one line.

Category: General Settings

Syntax: teWordWrap "Rectangle" "Options"

Specify the name of a NeoBook Rectangle. Use the keyword "Window" to start a new line when text reaches the window border. Use the keyword "None" to places the text in one line. (Unsupported and limited Functionality: Specify a value in twips for *options* to wrap the words at a fixed position. Note that other windows programs can overwrite this value. In this case a fixed line width will be ignored. A hint for twips calculations: 1440 twips is equal to 1 inch. Windows calculates approximately 96 pixels per inch for screen pixels depending on the system. Editors Toolbox calculates a Standard of 72 pixels per inch which comes close to most printer resolutions. The Example sets the wrapping position to 5 inches: teWordWrap "Rectangle1" "7200". This function is not intended to be used in final publications. It is just a "hack" for very special occasions.)

Example: The first example sets the default value to wrapping lines at the window border

teWordWrap "Rectangle1" "Window"

teWriteMode

Purpose: Specifies the editing mode of the Text Control. Valid modes are read only mode, normal writing and editing mode, a locked window to prevent refreshing and redrawing of the screen, Unlock Window for refreshing and redrawing, Disable or Enable Keyboard Input, Disable or Enable Format Change Events calling the Subroutine 'OnFormatChange', Enable or Disable editing and deleting of protected text, Show or Hide the Ruler.

Category: General Settings

Syntax: teWriteMode "Rectangle" "Options"

Specify the name of a NeoBook Rectangle. Enter the keyword "ReadOnly" in options to display text only. To enable keyboard input use the keyword "EditMode". Use the keyword "LockWindowUpdate" to prevent refreshing of the screen or "UnlockWindowUpdate" to allow refreshing of the screen. Use locking the window update to hide text processing functions from the user. Use it for example to perform search and replace operations to update variables in the text or using hyperlinks. See search and replace function in the CreativeWriter sample application how the teFindText command is used in background. Use the keyword "EnableChangeEvent" or "DisableChangeEvent" to enable or disable calls of the 'Subroutine OnFormatChange' at runtime programmatically. Use the keyword "ShowRuler" and "HideRuler" to show or hide the Ruler bar. Use the keyword "EnableEditProtected" and "DisableEditProtected" to enable 'editing and deleting of protected text'. See the CreativeWrite Sample application for more details how to work with protected text. Use the keyword "EnableKeyboard" and "DisableKeyboard" to enable and disable keyboard input. This is a different way to set a read only mode with a slightly different visual effect.

Example: teWriteMode "Rectangle1" "ReadOnly"

Rich Edit Shortcut Keys

Rich Text Edit Controls support the following shortcut keys. Not all commands are supported by Editors Toolbox for Neobook and available on all systems of Microsoft ® Windows!

Keys

Shift+Backspace
 Ctrl+Tab
 Ctrl+Clear
 Ctrl+Number Pad 5
 Ctrl+A
 Ctrl+E
 Ctrl+J
 Ctrl+R
 Ctrl+L
 Ctrl+C
 Ctrl+V
 Ctrl+X
 Ctrl+Z
 Ctrl+Y
 Ctrl+'+' (Ctrl+Shift+'=')
 Ctrl+'='
 Ctrl+1
 Ctrl+2
 Ctrl+5
 Ctrl+' (apostrophe)
 Ctrl+'` (grave)
 Ctrl+~ (tilde)
 Ctrl+; (semicolon)
 Ctrl+Shift+6
 Ctrl+, (comma)
 Ctrl+Shift+' (apostrophe)
 Backspace

 Ctrl+Backspace
 F16
 Ctrl+Insert
 Shift+Insert
 Insert
 Ctrl+Left Arrow
 Ctrl+Right Arrow
 Ctrl+Left Shift
 Ctrl+Right Shift
 Ctrl+Up Arrow
 Ctrl+Down Arrow
 Ctrl+Home
 Ctrl+End
 Ctrl+Page Up
 Ctrl+Page Down
 Ctrl+Delete
 Shift+Delete
 Esc
 Alt+Esc
 Alt+X

 Alt+Shift+X

 Alt+0xxx (Number Pad)

 Alt+Shift+Ctrl+F12
 Alt+Shift+Ctrl+F11

 Ctrl+Shift+A

Operations

Generate a LRM/LRM on a bidi keyboard
 Tab
 Select all
 Select all
 Select all
 Center alignment
 Justify alignment
 Right alignment
 Left alignment
 Copy
 Paste
 Cut
 Undo
 Redo
 Superscript
 Subscript
 Line spacing = 1 line.
 Line spacing = 2 lines.
 Line spacing = 1.5 lines.
 Accent acute
 Accent grave
 Accent tilde
 Accent umlaut
 Accent caret (circumflex)
 Accent cedilla
 Activate smart quotes
 If text is protected, beep and do not delete it. Otherwise, delete previous character.
 Delete previous word. This generates a VK_F16 code.
 Same as Backspace.
 Copy
 Paste
 Overwrite
 Move cursor one word to the left.
 Move cursor one word to the right.
 Left alignment
 Right alignment
 Move to the line above.
 Move to the line below.
 Move to the beginning of the document.
 Move to the end of the document.
 Move one page up.
 Move one page down.
 Delete the next word or selected characters.
 Cut the selected characters.
 Stop drag-drop.
 Change the active application.
 Converts the Unicode hexadecimal value preceding the insertion point to the corresponding Unicode character.
 Converts the Unicode character preceding the insertion point to the corresponding Unicode hexadecimal value.
 Inserts Unicode values if xxx is greater than 255. When xxx is less than 256, ASCII range text is inserted based on the current keyboard.
 Hex to Unicode.
 Selected text will be output to the debugger window and saved to %temp%\DumpFontInfo.txt.
 Set all caps.

Ctrl+Shift+L
Ctrl+Shift+Right Arrow
Ctrl+Shift+Left Arrow

Fiddle bullet style.
Increase font size.
Decrease font size.

Copyright and Trademarks

Copyright © 2003 - 2006 by C. Giebel. All Rights Reserved. This manual and the software described within are copyrighted with all rights reserved. No part of this publication or the software described may reproduced in any form without written permission.

Trademarks: Microsoft Windows, Windows XP, Word, Excel, WordPad, Rich Text Format, Rich Text Edit Control are trademarks of Microsoft Corp. Adobe Acrobat is a trademark of Adobe Systems Incorporated. NeoBook and NeoBookDB are trademarks of NeoSoft Corp. "Addict" is a trademark of Addictive Software. Adobe ® Acrobat ® PDF ® are trademarks or registered trademarks of Adobe Systems Incorporated. Delphi is a registered trademark of Borland Inc. All other brand and product names are trademarks or registered trademarks of their owners.